networking-cisco Documentation

Release

OpenStack Foundation

Nov 01, 2018

Contents

Contents

The networking-cisco project's goal is to provide support for Cisco networking hardware and software in OpenStack deployments. This includes ML2 drivers and agents for neutron, as well as other pieces of software which interact with neutron to best utilise your Cisco products with OpenStack.

- Free software: Apache license
- Documentation: http://networking-cisco.readthedocs.io/en/latest
- Source: http://git.openstack.org/cgit/openstack/networking-cisco
- Bugs: http://bugs.launchpad.net/networking-cisco

CHAPTER 1

Drivers for Cisco Products

- Nexus 9000 Series Switches
  - ML2 Mechanism driver - cisco_nexus
  - ML2 VXLAN Type driver - nexus_vxlan
- UCS Manager
  - ML2 Mechanism driver - cisco_ucsm
- ASR 1000 Series
  - Neutron Service Plugins - cisco_l3_routing
- Service Advertisement Framework (SAF)
  - Firewall drivers - native, phy_asa
  - Applications - fabric-enabler-server, fabric-enabler-agent, fabric-enabler-cli
- Prime Network Registrar (CPNR)
  - Applications - cpnr-rootwrap, cpnr-dns-relay-agent, cpnr-dns-relay, cpnr-dhcp-relay-agent, cpnr-dhcp-relay
- Application Policy Infrastructure Controller (APIC)
  - *No longer supported.* Removed in release 5.0.0
  - Code removed by commit 10b124d39fde4085a695d5c6652c8fb6e0620ece
  - Driver now hosted in repo https://github.com/noironetworks/apic-ml2-driver
- Network Convergence System (NCS)
  - *No longer supported.* Removed in release 6.0.0
  - Code removed by commit 31e4880299d04ceb399aa38097fc5f2b26e30ab1
- Nexus 1000v
  - *No longer supported.* Removed in release 6.0.0

  - Code removed by commit 0730ec9e6b76b3c1e75082e9dd1af55c9faeb34c
- CSR 1000v series
  - *No longer supported.* Removed in release 6.0.0
  - Code removed by commit 917480566afa2b40dc382bc4f535d173bad7736d

CHAPTER 2

Releases and Version Support

We have a goal to maintain compatibility with each version of OpenStack for as long as possible, so starting with version 4.0.0, networking-cisco was compatible with both the Mitaka and Newton OpenStack releases. As such networking-cisco is branchless and stable releases that are compatible with multiple OpenStack versions will be cut from our master branch.

The latest (6.x) release of networking-cisco is compatible with the Mitaka, Newton, Ocata, Pike and Queens releases of OpenStack.

For full release notes refer to *Full Release Notes for networking-cisco*

CHAPTER 3

Using Networking Cisco

3.1 Installation Guides

3.1.1 Installing the networking-cisco Package

The following lists steps to install the networking-cisco repository:

1. Released versions of networking-cisco are available via either:

```
http://tarballs.openstack.org/networking-cisco
https://pypi.org/project/networking-cisco
```

   The neutron release is http://tarballs.openstack.org/neutron

2. To install networking-cisco, do as follows:

   - When using pip for installs, do either:

```
$ pip install networking-cisco
$ pip install <path to downloaded networking-cisco tarball>
```

   - To install networking-cisco without pip, do:

```
$ tar -zxvf <downloaded networking-cisco tarball>
$ cd ./networking-cisco-<version>
$ python setup.py install
```

   - To install networking-cisco package from system packages, do:

```
$ yum install python-networking-cisco
```

3. Recent additions to networking-cisco package data requires a data base migration to be performed. This can be done by running:

```
$ su -s /bin/sh -c "neutron-db-manage --config-file /etc/neutron/neutron.conf --
↪config-file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade head" neutron
```

Note: If a separate file for cisco configuration exists (ex: ml2_conf_cisco.ini), that file also should be included by following other config files in the command with:

```
--config-file /etc/neutron/plugins/ml2/ml2_conf_cisco.ini
```

3.1.2 ASR1000 L3 Router Service Plugin Installation Guide

This is an installation guide for enabling the ASR1000 L3 Router Service Plugin (L3P) support on OpenStack.

Prerequisites

The prerequisites for installing the ASR1k L3P are as follows:

- Cisco IOS XE image version 16.03.04. Refer to ASR1k docs for upgrade/downgrade instructions. From this link, select the 'Support' option in the middle of the page for information on upgrade/downgrade guides, configuration and technical references.
- The ASR1k L3P has been tested on these OSs.
  - Ubuntu 14.04 or above
- Your ASR1k router must be set-up as described in the next section *ASR1k Router Setup*.
- As the ASR1k L3P uses ncclient the following must also be installed:
  - `Paramiko` library, the SSHv2 protocol library for python
  - The `ncclient` (minimum version v0.4.6) python library for NETCONF clients. Install the ncclient library by using the pip package manager at your shell prompt: **`pip install ncclient >= 0.4.6`**
  - Additionally, the following patch to `ncclient` is needed

ASR1k Router Setup

This section lists what is required to prepare the ASR1k router for operation with the ASR1k L3P.

1. Your ASR1k router must be connected to a management network separate from the OpenStack data network. The configuration agent (CFGA) *must* be able to access this network so it can communicate with the ASR1k router to set up tenant data flows.

   The following exemplifies this for a case where the management subnet is 10.0.10.0/24 with upstream gateway 10.0.10.1/24. The management interface of the ASR1k is `GigabitEthernet0` and its IP address is 10.0.10.39/24.

```
vrf definition Mgmt-intf
 !
 address-family ipv4
 exit-address-family
 !
 address-family ipv6
 exit-address-family
```

```
 !
!
interface GigabitEthernet0
vrf forwarding Mgmt-intf
ip address 10.0.10.39 255.255.255.0
negotiation auto
no mop enabled
!
ip route vrf Mgmt-intf 0.0.0.0 0.0.0.0 10.0.10.1
ip ssh source-interface GigabitEthernet0
!
```

2. A CFGA uses Netconf to apply configurations in an ASR1k router. Netconf therefore has to be enabled by the router administrator along with user credentials for authentication. These credentials must match one of the `hosting_device_credentials` defined for CFGA (*see*).

   The following exemplifies this for a user `stack` and password `cisco`:

```
username stack privilege 15 password 0 cisco
netconf max-sessions 16
netconf ssh
```

3. Some pre-configuration of ASR1k interfaces that will carry data traffic for Neutron routers must be performed by the router administrator.

   - All participating interfaces must be enabled with **`no shutdown`**.
   - Any port-channels to be used must also be pre-configured.

   The following example defines port-channel 10 on interfaces `TenGigabitEthernet0/1/0` and `TenGigabitEthernet0/2/0`:

```
interface Port-channel10
no ip address
!
interface TenGigabitEthernet0/1/0
no ip address
cdp enable
channel-group 10 mode active
!
interface TenGigabitEthernet0/2/0
no ip address
cdp enable
channel-group 10 mode active
!
```

ASR1k L3P Installation

1. Install networking-cisco package as described in the section *Installing the networking-cisco Package*.
2. Configure ASR1k L3 router service plugin, its dependencies, the device manager plugin and the configuration agent. Full details on how to do this are available in the *ASR1000 L3 Router Service Plugin Administration Guide*. For details on each configuration parameters, refer to *ASR1k Configuration Reference*.
3. Restart neutron to pick-up configuration changes. For example on Ubuntu 14.04 use:

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

3.1.3 Nexus Mechanism Driver Installation Guide

This is an installation guide for enabling the Nexus Mechanism Driver (MD) support on OpenStack. This guide only covers details on the Nexus MD install and does not cover OpenStack or Nexus 9K switch installation. The *Prerequisites* section contains links for this.

Prerequisites

The prerequisites for installing the ML2 Nexus MD are as follows:

- Requires neutron installation as described in Neutron Install documentation.
- The ML2 Nexus MD have been tested on these OSs.
  - RHEL 6.1 or above
  - Ubuntu 14.04 or above
- Your Nexus switch must be set-up as described in the next section *Nexus Switch Setup*.
- Cisco Nexus 9K image version - NX-OS 7.0(3)I5(2) (minimum required for REST API Driver). Refer to Nexus 9K documents for upgrade/downgrade instructions. From this link, select the 'Support' option in the middle of the page for information on upgrade/downgrade guides, configuration and technical references.

Nexus Switch Setup

This section lists what is required to prepare the Nexus switch for operation with the Nexus Driver.

1. Your Nexus switch must be connected to a management network separate from the OpenStack data network. The Nexus MD *must* be able to access this network so it can communicate with the switch to set up your data flows.
2. The ML2 Nexus Rest API Driver requires **`feature nxapi`** to be enabled on the switch.
3. Each OpenStack compute host on the cloud must be connected to the switch using an interface dedicated solely to OpenStack data traffic. Connecting the OpenStack Network-Node(s) may also be needed depending on your network configuration. If Network Nodes are connected, you can pre-configure the tenant vlan range on the Nexus switchport; otherwise, like all compute nodes you can configure the Nexus ML2 Driver to manage the switchports by configuring the hostname to Nexus switchport mapping using the configuration option *host_ports_mapping* beneath the section *[ml2_mech_cisco_nexus]* of the neutron start-up config file.
4. Some pre-configuration must be performed by the Nexus switch administrator. For instance:
   - All participating OpenStack hosts interfaces must be enabled with **`no shutdown`**. If additional trunk vlans are needed for an interface, the administrator should manually apply these extra vlans using **`switchport trunk allowed vlan add <vlanid>`** and also include **`switchport mode trunk`**.
   - Possible port-channels pre-configuration:
     - Peer-link configuration must always be pre-configured.
     - The port-channel configuration must be preconfigured for all non-baremetal cases. For baremetal cases, when administrator prefers to have port-channels learned versus automatically creating port-channels using `vpc-pool` Nexus driver config.
     - When port-channels are learned, the interfaces must also be configured as members of the port-channel.
   - For VXLAN Overlay Configurations, enable the feature using the following Nexus CLI:

```
feature nv overlay
feature vn-segment-vlan-based
interface nve1
    no shutdown
    source-interface loopback x
    # where x is same value as Nexus driver config 'nve_src_intf'
```

ML2 Nexus MD Installation

1. Install networking-cisco package as described in the section *Installing the networking-cisco Package*.
2. Configure Nexus ML2 Driver. Once the networking-cisco code is installed, it needs to be configured and enabled in neutron, the *Nexus Mechanism Driver Administration Guide* provides full details on how to create the neutron configs for various use cases. For details on each configuration parameters, refer to *Nexus Configuration Reference*.

   Below is a simple VLAN configuration which can be applied to ML2 neutron config files `ml2_conf.ini` and possibly `ml2_conf_cisco.ini` located in directory `/etc/neutron/plugins/ml2`. The file `ml2_conf_cisco.ini` is optional if you'd like to isolate cisco specific parameters.

```
[ml2]
#- This neutron config specifies to use vlan type driver and use
#  cisco nexus mechanism driver.
type_drivers = vlan
tenant_network_types = vlan
mechanism_drivers = openvswitch,cisco_nexus

#- This extension driver must be enabled when the mechanism
#  driver includes nexus.
extension_drivers = cisco_providernet_ext

#- This neutron config specifies the vlan range to use.
[ml2_type_vlan]
network_vlan_ranges = physnet1:1400:3900

#- Provide Nexus credentials, OpenStack hostname, and nexus interface
[ml2_mech_cisco_nexus:192.168.1.1]
username=admin
password=mySecretPasswordForNexus
host_ports_mapping=host-1:[1/2]
```

3. Restart neutron to pick-up configuration changes.

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

3.1.4 UCSM Mechanism Driver Installation Guide

This installation guide details enabling the Cisco Unified Computing System Manager (UCSM) Mechanism Driver (MD) to configure UCS Servers and Fabric Interconnects managed by one or more UCS Managers in an OpenStack environment.

Prerequisites

The prerequisites for installing the ML2 UCSM mechanism driver are as follows:

- Cisco UCS B or C series servers connected to a Fabric Interconnect running UCS Manager version 2.1 or above. Please refer to UCSM Install and Upgrade Guides for information on UCSM installation.
- Associate UCS servers with a Service Profile or Service Profile Template on the UCS Manager. Instructions on how to do this via the UCSM GUI can be found in UCS Manager Server Management Guide
- Identify the vNICs for OpenStack use beforehand. If this configuration is not provided to the UCSM driver, it will configure vNICs eth0 and eth1 with OpenStack related configuration. If a subset of the vNICs on the UCS Servers need to be reserved for non-OpenStack workloads, the list of vNICs provided to the UCSM driver should not include those vNICs.
- OpenStack networking service neutron installed according to instructions in neutron install guide
- OpenStack running on the OSs:
  - RHEL 6.1 or above OR
  - Ubuntu 14.04 or above
- UCS Manager version 2.2 running on the Fabric Interconnect. This software can be downloaded from UCS Manager Software Download
- A valid SSL certificate can be set up on the UCS Manager by following instructions specified in Cisco UCS Manager Administration Management Guide

ML2 UCSM MD Installation

1. Install networking-cisco repository as described in the section *Installing the networking-cisco Package*.
2. Once the networking-cisco code is installed, configure and enable the Cisco UCSM ML2 driver in neutron. The *UCSM Mechanism Driver Administration Guide* provides full details on how to create the neutron configuration for various use cases.

   Below is a simple VLAN configuration along with UCSM driver configuration which can be applied to ML2 neutron config files `ml2_conf.ini`.

```
[ml2]
#- This neutron config specifies to use vlan type driver and use
#  cisco ucsm mechanism driver.
type_drivers = vlan
tenant_network_types = vlan
mechanism_drivers = openvswitch,cisco_ucsm

#- This neutron config specifies the vlan range to use.
[ml2_type_vlan]
network_vlan_ranges = physnet1:1400:3900

#- Provide UCSM IP and credentials
#  This format can be used when there is 1 UCSM to be configured.
[ml2_cisco_ucsm]
ucsm_ip=10.10.10.10
ucsm_username=admin
ucsm_password=mysecretpassword

# List of vNICs on every UCS Server that can be configured for
# tenant VLAN configuration.
ucsm_virtio_eth_ports=ucs-eth-0, ucs-eth-1

# Hostname to Service Profile mapping for Compute hosts managed by
# this UCS Manager. This config should be specified for hosts configured
```

```
# with only Service Profiles and not Service Profile Templates.
ucsm_host_list=controller-1:Controller-SP, compute-1:Compute-SP

# Service Profile Template config for UCSM. This is a mapping of Service Profile
# Template to the list of UCS Servers (shown as S# below) controlled by this␣
↪template.
sp_template_list = SP_Template1_path:SP_Template1:S1,S2 SP_Template2_path:SP_
↪Template2:S3,S4,S5

# vNIC Template config for UCSM. This is a mapping of vNIC Templates on the UCS
# Manager that control vNICs that are connected to Neutron provider networks.
vnic_template_list = physnet1:vnic_template_path1:vt1 physnet2:vnic_template_
↪path2:vt2
```

3. If the installation consists of multiple Fabric Interconnects with multiple UCS Managers running on them, the UCSM driver will talk to each one of them to configure the set of UCS Servers controlled by them. In that case, the neutron configuration file needs to contain configuration for all these UCS Managers and the UCS servers they control.

   Below is a snippet of configuration that depicts the multi-UCSM configuration format.

```
[ml2_cisco_ucsm]

# This block of config has to repeat for each UCSM in the installation
[ml2_cisco_ucsm_ip:1.1.1.1]
ucsm_username = username
ucsm_password = password

# List of vNICs on every UCS Server that can be configured for
# tenant VLAN configuration.
ucsm_virtio_eth_ports = eth0, eth1

# Hostname to Service Profile mapping for Compute hosts managed by
# this UCS Manager. This config should be specified for hosts configured
# with only Service Profiles and not Service Profile Templates.
ucsm_host_list = Hostname1:Serviceprofile1, Hostname2:Serviceprofile2

# Service Profile Template config per UCSM. This is a mapping of Service Profile
# Template to the list of UCS Servers controlled by this template.
sp_template_list = SP_Template1_path:SP_Template1:S1,S2 SP_Template2_path:SP_
↪Template2:S3,S4

# vNIC Template config per UCSM. This is a mapping of vNIC Templates on the UCS
# Manager that control vNICs that are connected to Neutron provider networks.
vnic_template_list = physnet1:vnic_template_path1:vt1 physnet2:vnic_template_
↪path2:vt2
```

4. Though not recommended, the UCSM SSL certificate checking can be disabled if necessary.

```
[ml2_cisco_ucsm]

ucsm_https_verify = False
```

5. Restart neutron to pick-up configuration changes.

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

Configuring UCSM Driver via TripleO

VLAN Configuration

The Cisco specific implementation is deployed by modifying the tripleo environment file Tripleo Nexus Ucsm Env File and updating the contents with the deployment specific content. Note that with TripleO deployment, the server names are not known before deployment so the MAC address of the server must be used in place of the server name. Descriptions of the parameters can be found at Tripleo Nexus Ucsm Parm file.

```
resource_registry:
  OS::TripleO::AllNodesExtraConfig: /usr/share/openstack-tripleo-heat-templates/
↪puppet/extraconfig/all_nodes/neutron-ml2-cisco-nexus-ucsm.yaml
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-configs/
↪compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-configs/
↪controller.yaml

parameter_defaults:

  NetworkUCSMIp: '10.86.1.10'
  NetworkUCSMUsername: 'neutron'
  NetworkUCSMPassword: 'cisco123'
  NetworkUCSMHostList: '06:00:C0:06:00:E0:bxb6-C6-compute-2,06:00:C0:05:00:E0:bxb6-C5-
↪compute-1,06:00:C0:03:00:E0:bxb6-C3-control-2,06:00:C0:07:00:E0:bxb6-C7-compute-3,
↪06:00:C0:04:00:E0:bxb6-C4-control-3,06:00:C0:02:00:E0:bxb6-C2-control-1'

  ControllerExtraConfig:
    neutron::plugins::ml2::mechanism_drivers: ['openvswitch', 'cisco_ucsm']
```

Note: Multi-UCSM configuration is currently not supported via TripleO.

3.1.5 [DEPRECATED] Prime Network Registrar (PNR)

1. General

This is an installation guide for enabling Cisco Prime Network Registrar (PNR) support on OpenStack.

Please refer to PNR installation guide (http://www.cisco.com/c/en/us/support/cloud-systems-management/prime-network-registrar/tsd-products-support-series-home.html) for how to install and bring up the PNR.

The neutron DHCP agent in the OpenStack environment needs to be setup to communicate with the PNR DHCP server and with the PNR DNS server. The PNR DHCP server performs leasing operations and PNR DNS server resolves DNS queries, these two servers replace dnsmasq.

This guide does not cover OpenStack installation.

2. Prerequisites

The prerequisites for installing the PNR OpenStack enabler are the following:

- Install PNR with required DNS and DHCP licenses.
- Disable dnsmasq or other DNS/DHCP services.

3. PNR plugin Installation

3.1 Using devstack

In this scenario, the PNR plugin will be installed along with OpenStack using devstack.

1. Clone devstack.
2. Add this repo as an external repository:

```
> cat local.conf
[[local|localrc]]
enable_plugin networking-cisco https://git.openstack.org/openstack/networking-
↪cisco.git
enable_service net-cisco
```

3. **`./stack.sh`**

3.2 On a setup with OpenStack already installed

In this scenario, the PNR plugin will be installed on a setup which has OpenStack installed already:

1. Clone networking-cisco.

2. **`cd networking-cisco`**
3. **`sudo python networking_cisco/setup.py install`**
4. The following modifications are needed in the `dhcp_agent.ini` file.

   Change the DHCP driver from dnsmasq to PNR.

   ```
   [DEFAULT]
   #dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
   dhcp_driver = networking_cisco.plugins.cisco.cpnr.dhcp_driver.CpnrDriver
   ```

   Add the following new section to the `dhcp_agent.ini` file with the details for contacting the PNR local server.

   ```
   [cisco_pnr]
   http_server = http://<pnr_localcluster_ipaddress>:8080
   http_username = <pnr_localcluster_username>
   http_password = <pnr_localcluster_password>
   external_interface = eth0
   dhcp_server_addr = <pnr_localcluster_ipaddress>
   dhcp_server_port = 67
   dns_server_addr = <pnr_localcluster_ipaddress>
   dns_server_port = 53
   ```

   Change the <pnr_localcluster_ipaddress> to the IP address of the local PNR VM.

   Change the <pnr_localcluster_username> and <pnr_localcluster_password> to the same username and password provided during PNR installation.

   If you are using HTTPS with a valid SSL certificate, change the scheme in http_server config variable to 'https' and the port number in the address to the appropriate port (default 8443).

   If you do not want to verify SSL certificates, add a config variable to `dhcp_agent.ini` file.

   ```
   [cisco_pnr]
   insecure = True
   ```

Note that using the `insecure` variable is NOT recommended in production.

5. After changing `dhcp_agent.ini`, restart the DHCP agent.

   On Red Hat based server:

   ```
   systemctl restart neutron-dhcp-agent
   ```

   On Ubuntu based server:

   ```
   service restart neutron-dhcp-agent
   ```

6. Start the dhcp and dns relay from command line as a detached background process. The relay files exist in networking_cisco/plugins/cisco/cpnr.

   ```
   nohup python dhcp_relay.py --config-file /etc/neutron/dhcp_agent.ini
   --log-file /var/log/neutron/dhcp-relay.log &
   ```

   ```
   nohup python dns_relay.py --config-file /etc/neutron/dhcp_agent.ini
   --log-file /var/log/neutron/dns-relay.log &
   ```

3.1.6 [DEPRECATED] Nexus StandAlone Fabric (SAF)

1. General

This is an installation guide for enabling nexus fabric support on OpenStack

Please refer to nexus fabric system configuration for how to bring up the fabric using a spine and leaf topology with DCNM as the fabric manager. The compute node in an OpenStack setup should be connected to the nexus fabric leaf switch. This link on the compute node/server is often referred as uplink in this note.

This guide does not cover OpenStack installation.

2. Prerequisites

The prerequisites for installing Nexus fabric OpenStack enabler are the following:

- Install LLDPad
- Install OVS (version 2.3.x)

3. Fabric Enabler Installation

3.1 Using devstack

In this scenario, SAF will be installed along with openstack using devstack

1. Clone devstack.
2. Use `networking-cisco/devstack/saf/local.conf.compute.saf` and `networking-cisco/devstack/saf/local.conf.control.saf` as and example to create local.conf for control and compute nodes and set the required parameters in the local.conf based on the your setup.
3. Run ./stack.sh

3.2 On a setup with OpenStack already installed

In this scenario, SAF will be installed on a setup which has already OpenStack installed:

1. Clone networking-cisco.

2. The following modifications are needed in:

```
2.1 /etc/neutron/plugins/ml2/ml2_conf.ini

[ml2]
type_drivers = local
mechanism_drivers = openvswitch

[ovs]
bridge_mappings = ethd:br-ethd

[agent]
tunnel_types =

Following sections should remain empty:
[ml2_type_flat]
[ml2_type_vlan]
[ml2_type_gre]
[ml2_type_vxlan]


L3 agent  - must be disabled
DHCP service - must be disabled


2.2 neutron.conf:

[DEFAULT]
notification_driver = messaging
notification_topics = cisco_dfa_neutron_notify
rpc_backend = rabbit

[keystone_authtoken]
...
auth_host = <ip address of controller>
auth_port = 35357
admin_tenant_name = service
admin_user = neutron
admin_password = <admin password>
...

2.3 nova.conf:

[keystone_authtoken]
...
admin_password = <admin password>
admin_user = nova
admin_tenant_name = service
auth_uri = http://<ip address of controller>:5000/v2.0
auth_host = <ip address of controller>
...


3.3 keystone.conf:
[DEFAULT]
notification_driver = messaging
notification_topics = cisco_dfa_keystone_notify
admin_endpoint = http://<services ip address>:%(admin_port)s/
rpc_backend = rabbit
```

4. `cd networking-cisco`
5. Edit `networking-cisco/etc/saf/enabler_conf.ini`

   Set the parameters in each section of the enabler_conf.ini based on your setup
6. Run `python tools/saf_prepare_setup.py`
7. Run `sudo python setup.py install`
8. On controller node run:
   - On ubuntu based server:

     `sudo start fabric-enabler-server`
   - On Red Hat based server:

     `sudo systemctl start fabric-enabler-server`
9. On compute node run:
   - On ubuntu based server:

     `sudo start fabric-enabler-agent`
   - On Red Hat based server:

     `sudo systemctl start fabric-enabler-agent`

3.2 Administrator Guides

3.2.1 ASR1000 L3 Router Service Plugin Administration Guide

The ASR1000 (ASR1k) L3 Router Service Plugin (L3P) implements neutron's L3 routing service API on the Cisco ASR1000 family of routers.

Specifically it provides the following features:

- L3 forwarding between subnets on the tenants' neutron L2 networks
- Support for overlapping IP address ranges between different tenants so each tenant could use the same IPv4 address space, **RFC 1918**
- P-NAT overload for connections originating on private subnets behind a tenant's neutron gateway routers connected to external neutron networks
- Floating IP, i.e., static NAT of a private IP address on a internal neutron subnet to a public IP address on an external neutron subnet/network
- Static routes on neutron routers
- HSRP-based high availability (HA) whereby a neutron router is supported by two (or more) ASR1k routers, one actively doing L3 forwarding, the others ready to take over in case of disruptions

Component Overview

To implement neutron routers in ASR1000 routers the ASR1k L3P relies on two additional Cisco components: a device manager plugin (DMP) for neutron server and a configuration agent (CFGA).

The DMP manages a device repository in which ASR1k routers are registered. An ASR1k router in the DMP repository is referred to as a *hosting device*. neutron server should be configured so that it loads both the DMP and the L3P when it starts, covered in section *Configuring Neutron directly for ASR1000*.

The CFGA is a standalone component that needs to be separately started as neutron server cannot be configured to take care of that. The CFGA monitors hosting devices as well as performs configurations in them upon instruction from the L3P or the DMP. That communication is done using the regular AMQP message bus that is used by Openstack services.

Warning: The ASR1k L3P and CFGA assume that nobody else manipulates the configurations the CFGA makes in the ASR1k routers used in the Openstack neutron deployment. If router administrators do not honor this assumption the CFGA may be unable to perform its configuration tasks.

Limitations

- The neutron deployment must use VLAN-based network segmentation. That is, the L2 substrate must be controlled by ML2's VLAN technology driver.
- Access to nova's Metadata service via neutron routers is not supported. The deployment can instead provide access via neutron's DHCP namespaces when IPAM is implemented using neutron DHCP agents. Alternatively, metadata can be provided to nova virtual machines using nova's config drive feature.
- Only one router can be attached to a particular internal neutron network. If a user attempts to attach a router to an internal network that already has another router attached to it the L3P will reject the request.

Configuring Neutron directly for ASR1000

The subsections that follows details the steps to be performed to properly configure and start neutron so that ASR1k devices can host neutron routers.

Configure enabled service plugins in neutron

Update the neutron configuration file commonly named `neutron.conf` so the neutron server will load the device manager and L3 service plugins.

This file is most commonly found in the directory `/etc/neutron`. The `service_plugins` configuration option should contain the following two items:

- `networking_cisco.plugins.cisco.service_plugins.cisco_device_manager_plugin.CiscoDeviceManagerPlugin`
- `networking_cisco.plugins.cisco.service_plugins.cisco_router_plugin.CiscoRouterPlugin`

in addition to any other service plugins.

```
[DEFAULT]
service_plugins = networking_cisco.plugins.cisco.service_plugins.cisco_device_manager_
↪plugin.CiscoDeviceManagerPlugin,networking_cisco.plugins.cisco.service_plugins.
↪cisco_router_plugin.CiscoRouterPlugin
```

Specify ASR credentials

Add credential information to the configuration file under the section `[hosting_device_credentials]`. The format is as follows:

- `[cisco_hosting_device_credential:`*`UUID`*`]` of hosting device credentials
- `name=`*`NAME`* of credentials
- `description=`*`description`* of credentials
- `user_name=`*`USERNAME`*, username of credentials
- `password=`*`PASSWORD`*, password of credentials
- `type=`*`TYPE`*, *not required for ASR1k, can be left empty*

The credentials are used by a CFGA when configuring ASR1k routers. For that reason the router administrator needs to pre-configure those credentials in the ASR1k devices.

The following is an example of credentials defined in a configuration file that neutron server reads:

```
[hosting_device_credentials]
[cisco_hosting_device_credential:1]
name="Universal credential"
description="Credential used for all hosting devices"
user_name=stack
password=cisco
type=
```

Note: As the credential definitions are tightly coupled to Cisco device management they may be placed in the file `cisco_device_manager_plugin.ini`.

Define hosting device templates

Define hosting device templates for ASR1k devices and devices supporting Linux network namespace-based routers. The hosting device template definition should be placed in the `[hosting_device_templates]` section with the following format:

- `[cisco_hosting_device_template:`*`UUID`*`]` of hosting device template
- `name=`*`NAME`* given to hosting devices created using this template
- `enabled=`*`True`*`|`*`False`*, `True` if template enabled, `False` otherwise
- `host_category=`*`VM`*`|`*`Hardware`*`|`*`Network_Node`*
- `service_types=`*`SERVICE_TYPES`*, *not required for ASR1k, can be left empty*
- `image=`*`IMAGE`*, name or UUID of Glance image, *not used for ASR1k*
- `flavor=`*`UUID`* of nova VM flavor, *not used for ASR1k*
- `default_credentials_id=`*`UUID`* of default credentials
- `configuration_mechanism=`*`MECHANISM`*, *not required for ASR1k, can be left empty*
- `protocol_port=`*`PORT`* udp/tcp port for management
- `booting_time=`*`SECONDS`*, typical booting time of devices based on this template

- `slot_capacity=`*`INTEGER`*, abstract metric specifying capacity to host logical resources like neutron routers
- `desired_slots_free=`*`INTEGER`*, desired number of slots to keep available at all times
- `tenant_bound=`*`TENANT_SPEC`*, list of tenant UUIDs to which template is available, if empty available to all tenants
- `device_driver=`*`MODULE`* to be used as hosting device driver
- `plugging_driver=`*`MODULE`* to be used as plugging driver

The hosting device template stores information that is common for a certain type of devices (like the ASR1k). The information is used by the DMP and the CFGA to tailor how to they manage devices of the type in question.

The following is an example with template 1 for devices using namespaces and template 2 for ASR1k devices):

```
[hosting_devices_templates]
[cisco_hosting_device_template:1]
name=NetworkNode
enabled=True
host_category=Network_Node
service_types=router:FW:VPN
image=
flavor=
default_credentials_id=1
configuration_mechanism=
protocol_port=22
booting_time=360
slot_capacity=2000
desired_slots_free=0
tenant_bound=
device_driver=networking_cisco.plugins.cisco.device_manager.hosting_device_drivers.
 ↪noop_hd_driver.NoopHostingDeviceDriver
plugging_driver=networking_cisco.plugins.cisco.device_manager.plugging_drivers.noop_
 ↪plugging_driver.NoopPluggingDriver

[cisco_hosting_device_template:3]
name="ASR1k template"
enabled=True
host_category=Hardware
service_types=router
image=
flavor=
default_credentials_id=1
configuration_mechanism=
protocol_port=22
booting_time=360
slot_capacity=2000
desired_slots_free=0
tenant_bound=
device_driver=networking_cisco.plugins.cisco.device_manager.hosting_device_drivers.
 ↪noop_hd_driver.NoopHostingDeviceDriver
plugging_driver=networking_cisco.plugins.cisco.device_manager.plugging_drivers.hw_
 ↪vlan_trunking_driver.HwVLANTrunkingPlugDriver
```

A normal deployment need not modify any of the values in the example above.

Note: As the hosting device template definitions are tightly coupled to Cisco device management, they may be placed in the file `cisco_device_manager_plugin.ini`.

Add ASR1k devices to device repository

Register ASR1k devices in the device repository. The information that needs to be provided should be placed in the `[hosting_devices]` section and should be formatted as:

- `[cisco_hosting_device:`*`UUID`*`]` of hosting device
- `template_id=`*`UUID`* of hosting device template for this hosting device
- `credentials_id=`*`UUID`* of credentials for this hosting device
- `name=`*`NAME`* of device, e.g., its name in DNS
- `description=`*`DESCRIPTION`* arbitrary description of the device
- `device_id=`*`MANUFACTURER_ID`* of the device, e.g., its serial number
- `admin_state_up=True|False`, `True` if device is active, `False` otherwise
- `management_ip_address=`*`IP ADDRESS`* of device's management network interface
- `protocol_port=`*`PORT`* udp/tcp port of hosting device's management process
- `tenant_bound=`*`UUID`* of tenant allowed to have neutron routers on the hosting device, if empty any tenant can have neutron routers on it
- `auto_delete=`*`True|False`*, only relevant for VM-based hosting devices, so value is ignored for ASR1k devices

If any of the `UUID` values are given as an integer, they will automatically be converted into a proper UUID when the hosting device is added to the database. Hence, `1` becomes `00000000-0000-0000-0000-000000000001`.

Once registered, the L3P starts scheduling neutron routers to those devices that have `admin_state_up` set to True. Neutron routers already scheduled to a disabled hosting device continue to operate as normal.

In the example below, two ASR1k routers are registered as hosting devices based on hosting device template 3 and to use credentials 1 as defined in the earlier *credentials* and *hosting device template* examples:

```
[hosting_devices]
[cisco_hosting_device:3]
template_id=3
credentials_id=1
name="ASR1k device 1"
description="ASR1k in rack 2"
device_id=SN:abcd1234efgh
admin_state_up=True
management_ip_address=10.0.100.5
protocol_port=22
tenant_bound=
auto_delete=False

[cisco_hosting_device:5]
template_id=3
credentials_id=1
name="ASR1k device 2"
description="ASR1k in rack 5"
device_id=SN:efgh5678ijkl
admin_state_up=True
management_ip_address=10.0.100.6
protocol_port=22
tenant_bound=
auto_delete=False
```

The ASR1k routers have to be configured by the router administrator to accept the credentials specified in the hosting device database record.

The plugging driver for VLAN trunking needs to be configured with the ASR1k interfaces to use for tenant data traffic. This information is placed in the section `[plugging_drivers]` and should be structured as follows:

- `[HwVLANTrunkingPlugDriver:`*`UUID`*`]` of hosting device
- `internal_net_interface_`*`NUMBER`*`=`*`NETWORK_SPEC`*`:`*`INTERFACE_NAME`*
- `external_net_interface_`*`NUMBER`*`=`*`NETWORK_SPEC`*`:`*`INTERFACE_NAME`*

The `NETWORK_SPEC` can be `*`, which matches any network UUID, or a specific network UUID, or a comma separated list of network UUIDs.

The example below illustrates how to specify that `Port-channel 10` in for hosting devices 3 and 4 will carry all tenant network traffic:

```
[plugging_drivers]
[HwVLANTrunkingPlugDriver:3]
internal_net_interface_1=*:Port-channel10
external_net_interface_1=*:Port-channel10

[HwVLANTrunkingPlugDriver:5]
internal_net_interface_1=*:Port-channel10
external_net_interface_1=*:Port-channel10
```

Note: As the hosting device definitions and plugging driver configurations are tightly coupled to Cisco device management, they may be placed in the file `cisco_device_manager_plugin.ini`.

Define router types

Define router types for neutron routers to be hosted in devices supporting Linux network namespaces and in ASR1k devices. The information that needs to be provided should be placed in the `[router_types]` section. The following is the format:

- `[cisco_router_type:`*`UUID`*`]` of router type
- `name=`*`NAME`* of router type, should preferably be unique
- `description=`*`DESCRIPTION`* of router type
- `template_id=`*`UUID`* of hosting device template for this router type
- `ha_enabled_by_default=`*`True`*`|`*`False`*, `True` if HA should be enabled by default, False otherwise
- `shared=`*`True`*`|`*`False`*, `True` if routertype is available to all tenants, `False` otherwise
- `slot_need=`*`NUMBER`* of slots this router type consumes in hosting devices
- `scheduler=`*`MODULE`* to be used as scheduler for router of this type
- `driver=`*`MODULE`* to be used by router plugin as router type driver
- `cfg_agent_service_helper=`*`MODULE`* to be used by CFGA as service helper driver
- `cfg_agent_driver=`*`MODULE`* to be used by CFGA agent for device configurations

A router type is associated with a hosting device template. Neutron routers based on a particular router type will only be scheduled to hosting devices based on the same hosting device template.

In the example below a router type is defined for neutron routers implemented as Linux network namespaces and for neutron routers implemented in ASR1k devices. The hosting device templates refers to the ones defined in the earlier *hosting device template example*:

```
[router_types]
[cisco_router_type:1]
name=Namespace_Neutron_router
description="Neutron router implemented in Linux network namespace"
template_id=1
ha_enabled_by_default=False
shared=True
slot_need=0
scheduler=
driver=
cfg_agent_service_helper=
cfg_agent_driver=

[cisco_router_type:3]
name=ASR1k_router
description="Neutron router implemented in Cisco ASR1k device"
template_id=3
ha_enabled_by_default=True
shared=True
slot_need=2
scheduler=networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_
↪scheduler.L3RouterHostingDeviceHARandomScheduler
driver=networking_cisco.plugins.cisco.l3.drivers.asr1k.asr1k_routertype_driver.
↪ASR1kL3RouterDriver
cfg_agent_service_helper=networking_cisco.plugins.cisco.cfg_agent.service_helpers.
↪routing_svc_helper.RoutingServiceHelper
cfg_agent_driver=networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_
↪routing_driver.ASR1kRoutingDriver
```

A normal deployment need not modify any of the values in the example above as long as the templates referred to are correct.

To ensure all neutron routers created by users are scheduled onto the ASR1k devices, the `default_router_type` configuration option in the `[routing]` section should be set to the name of the router type defined for ASR1k devices. For the example above, this would be done by:

```
[routing]
default_router_type = ASR1k_router
```

Note: As the router type definitions are tightly coupled to Cisco ASR1000 L3 router service plugin, they may be placed in the file `cisco_router_plugin.ini`.

Make services use correct configuration files

Include the configuration files on the command line when the neutron-server and configuration agent is started. For example:

```
$ /usr/local/bin/neutron-server --config-file /etc/neutron/neutron
  .conf \
  --config-file /etc/neutron/plugins/ml2/ml2_conf.ini \
  --config-file /etc/neutron/plugins/ml2/ml2_conf_cisco.ini \
```

```
  --config-file /etc/neutron/plugins/cisco/cisco_router_plugin.ini \
  --config-file /etc/neutron/plugins/cisco/cisco_device_manager_plugin.ini
```

It looks similarly for the configuration agent:

```
$ /usr/local/bin/neutron-cisco-cfg-agent \
  --config-file /etc/neutron/neutron.conf \
  --config-file /etc/neutron/plugins/cisco/cisco_cfg_agent.ini \
  --config-file /etc/neutron/plugins/cisco/cisco_router_plugin.ini \
  --config-file /etc/neutron/plugins/cisco/cisco_device_manager_plugin.ini
```

High-Availability for Neutron Routers in ASR1k devices

The HA is implemented using the HSRP feature of IOS XE.

When a user creates a neutron router that has HA enabled, the L3P will automatically create a second neutron router with the same name but with `_HA_backup_1` added to the name. This second router is referred to as a *redundancy router* and it is hidden from non-admin users. The HA-enabled router the user created is referred to as the *user-visible router*.

The router-list command issued by a neutron *admin* user returns both the user-visible and redundancy HA routers (list below has been truncated for clarity):

```
(keystone_admin)$ neutron router-list
+--------------------------------------+-------------------------------+------------
↪-----------------------------------------------------------------------------------
↪-----------------------------------------+
| id                                   | name                          | external_
↪gateway_info                                                                      ⌋
↪                                                 |
+--------------------------------------+-------------------------------+------------
↪-----------------------------------------------------------------------------------
↪-----------------------------------------+
| 0924ad2f-9858-4f2c-b4ea-f2aff15da682 | router1_HA_backup_1           | {"network_
↪id": "09ec988a-948e-42da-b5d1-b15c341f653c", "external_fixed_ips": [{"subnet_id":
↪"e732b00d-027c-45d4-a68a-10f1535000f4",         |
|                                      |                               | "ip_address
↪": "172.16.6.35"}]}                                                               ⌋
↪                                                 |
| 2c8265be-6df1-49eb-b8e9-e8c0aea19f44 | router1                       | {"network_
↪id": "09ec988a-948e-42da-b5d1-b15c341f653c", "external_fixed_ips": [{"subnet_id":
↪"e732b00d-027c-45d4-a68a-10f1535000f4",         |
|                                      |                               | "ip_address
↪": "172.16.6.34"}]}                                                               ⌋
↪                                                 |
                ...                                    ...                         ⌋
↪                                               ...
+--------------------------------------+-------------------------------+------------
↪-----------------------------------------------------------------------------------
↪-----------------------------------------+
```

The same router-list command issued by a *non-admin* user returns only the user-visible HA router:

```
(keystone_regular)$ neutron router-list
+--------------------------------------+---------+--------------------------------
↪-----------------------------------------------------------------------------------
↪-----------------------------------------+
```

```
| id                                   | name    | external_gateway_info                                                                                                                                  |
+--------------------------------------+---------+--------------------------------------------------------------------------------------------------------------------------------------------------------+
| 2c8265be-6df1-49eb-b8e9-e8c0aea19f44 | router1 | {"network_id": "09ec988a-948e-42da-b5d1-b15c341f653c", "external_fixed_ips": [{"subnet_id": "e732b00d-027c-45d4-a68a-10f1535000f4", "ip_address": "172.16.6.34"}]} |
+--------------------------------------+---------+--------------------------------------------------------------------------------------------------------------------------------------------------------+
```

The L3P uses a HA aware scheduler that will schedule the user-visible router and its redundancy router on different ASR1k devices. The CFGAs managing those ASR1k devices apply configurations for the user-visible router and its redundancy router so that they form a HSRP-based HA pair.

External Network Connectivity and Global Routers

Connectivity to external networks for neutron routers in the ASR1k is provided using interfaces in the global VRF of the ASR1k. The L3P represents an ASR1k's global VRF with a special neutron router referred to as a *global* neutron router. Global routers are only visible to admin users.

When a neutron gateway router has been scheduled to an ASR1k device, the L3P automatically creates a global router that is scheduled to that ASR1k. This global router will have regular router ports on every subnet of an external neutron network. Furthermore, the global router can be connected to several external networks if there are neutron gateway routers on the same ASR1k device that are attached to those networks.

Continuing the example above where the HA routers were discussed, the full list of routers are shown below:

```
(keystone_admin)$ neutron router-list
+--------------------------------------+---------------------------------+--------------------------------------------------------------------------------------------------------------------------------------------------+
| id                                   | name                            | external_gateway_info                                                                                                                            |
+--------------------------------------+---------------------------------+--------------------------------------------------------------------------------------------------------------------------------------------------+
| 0924ad2f-9858-4f2c-b4ea-f2aff15da682 | router1_HA_backup_1             | {"network_id": "09ec988a-948e-42da-b5d1-b15c341f653c", "external_fixed_ips": [{"subnet_id": "e732b00d-027c-45d4-a68a-10f1535000f4", "ip_address": |
|                                      |                                 | "172.16.6.35"}]}                                                                                                                                 |
| 2c8265be-6df1-49eb-b8e9-e8c0aea19f44 | router1                         | {"network_id": "09ec988a-948e-42da-b5d1-b15c341f653c", "external_fixed_ips": [{"subnet_id": "e732b00d-027c-45d4-a68a-10f1535000f4", "ip_address": |
|                                      |                                 | "172.16.6.34"}]}                                                                                                                                 |
| 5826d408-1fa3-4e01-b98a-8990060a8902 | Global-router-0000-000000000003 | null                                                                                                                                             |
| 66dba329-468c-4b15-8626-97a86afeaf79 | Global-router-0000-000000000005 | null                                                                                                                                             |
```

```
| 71336018-6390-4142-951a-f18d2f028a77 | Logical-Global-router           | null      ␣
↪                                                                                         ␣
↪                                                         |
+------------------------------------+--------------------------------+------------
↪--------------------------------------------------------------------------------
↪-------------------------------------------------------+
```

It shows two global routers: `Global-router-0000-000000000003` and `Global-router-0000-000000000005`. The table also contains a router named `Logical-Global-router`. HSRP-based HA is also used for the global routers. The logical global router stores HA information for the global routers, most importantly the HSRP VIP addresses. It only exists in the neutron database and is never explicitly seen by the CFGA.

The reason why there are two global routers in this example is the two HA routers (the user-visible one and its redundancy) have the gateway set and are scheduled to different ASR1k devices.

The details of router1 (see below) reveal that it has external gateway set to subnet `e732b00d-027c-45d4-a68a-10f1535000f4`. The `routerhost:hosting_device` field shows that it has been scheduled to hosting device `00000000-0000-0000-0000-000000000003`.

```
(keystone_admin)$ neutron router-show router1
+--------------------------------------------------+----------------------------------
↪--------------------------------------------------------------------------------
↪-------------------------------------------------------+
| Field                                            | Value                            ␣
↪                                                                                 ␣
↪                                                         |
+--------------------------------------------------+----------------------------------
↪--------------------------------------------------------------------------------
↪-------------------------------------------------------+
| admin_state_up                                   | True                             ␣
↪                                                                                 ␣
↪                                                         |
| cisco_ha:details                                 | {"redundancy_routers": [{"priority
↪": 97, "state": "STANDBY", "id": "0924ad2f-9858-4f2c-b4ea-f2aff15da682"}], "probe_
↪connectivity": false, "priority": 100, "state":         |
|                                                  | "ACTIVE", "redundancy_level": 1,
↪"type": "HSRP"}                                                                  ␣
↪                                                           |
| cisco_ha:enabled                                 | True                             ␣
↪                                                                                 ␣
↪                                                         |
| description                                      |                                  ␣
↪                                                                                 ␣
↪                                                         |
| external_gateway_info                            | {"network_id": "09ec988a-948e-
↪42da-b5d1-b15c341f653c", "external_fixed_ips": [{"subnet_id": "e732b00d-027c-45d4-
↪a68a-10f1535000f4", "ip_address": "172.16.6.34"}]}      |
| id                                               | 2c8265be-6df1-49eb-b8e9-
↪e8c0aea19f44                                                                     ␣
↪                                                                  |
| name                                             | router1                          ␣
↪                                                                                 ␣
↪                                                         |
| routerhost:hosting_device                        | 00000000-0000-0000-0000-
↪000000000003                                                                     ␣
↪                                                                  |
| routerrole:role                                  |                                  ␣
↪                                                                                 ␣
↪                                                         |
```

```
| routertype-aware-scheduler:auto_schedule         | True                                 ␣
↪                                                                                        ␣
↪                                                        |
| routertype-aware-scheduler:share_hosting_device | True                                 ␣
↪                                                                                        ␣
↪                                                        |
| routertype:id                                    | 00000000-0000-0000-0000-
↪000000000003
↪                                                                                        ␣
↪                                                                    |
| routes                                           |                                    ␣
↪                                                                                        ␣
↪                                                        |
| status                                           | ACTIVE                             ␣
↪                                                                                        ␣
↪                                                        |
| tenant_id                                        | fb99eb6f915342e399894a35f911b515   ␣
↪                                                                                        ␣
↪                                                        |
+--------------------------------------------------+-------------------------------------
↪-----------------------------------------------------------------------------------------
↪-----------------------------------------------------+
```

The details of `Global-router-0000-000000000003` (see below) show that it is also scheduled to hosting device `00000000-0000-0000-0000-000000000003`.

```
(keystone_admin)$ neutron router-show Global-router-0000-000000000003
+--------------------------------------------------+-------------------------------------
↪---+
| Field                                            | Value                              ␣
↪   |
+--------------------------------------------------+-------------------------------------
↪---+
| admin_state_up                                   | True                               ␣
↪   |
| cisco_ha:enabled                                 | False                              ␣
↪   |
| description                                      |                                    ␣
↪   |
| external_gateway_info                            |                                    ␣
↪   |
| id                                               | 5826d408-1fa3-4e01-b98a-
↪8990060a8902 |
| name                                             | Global-router-0000-000000000003    ␣
↪   |
| routerhost:hosting_device                        | 00000000-0000-0000-0000-
↪000000000003 |
| routerrole:role                                  | Global                             ␣
↪   |
| routertype-aware-scheduler:auto_schedule         | False                              ␣
↪   |
| routertype-aware-scheduler:share_hosting_device | True                               ␣
↪   |
| routertype:id                                    | 00000000-0000-0000-0000-
↪000000000003 |
| routes                                           |                                    ␣
↪   |
| status                                           | ACTIVE                             ␣
↪   |
```

```
| tenant_id                                        |                                                                  ␣
↪    |
+--------------------------------------------------+------------------------------------
↪---+
```

The `external_gateway_info` of `Global-router-0000-000000000003` is empty which is expected since global routers are attached to the external networks using regular router ports.

By listing the router ports of `Global-router-0000-000000000003` (see below), it can be seen that it indeed has a router port on the same subnet as the gateway of `router1`.

```
(keystone_admin)$ neutron router-port-list Global-router-0000-000000000003
+--------------------------------------+------+-------------------+-------------------
↪-----------------------------------------------------------------+
| id                                   | name | mac_address       | fixed_ips        ␣
↪                                                                  |
+--------------------------------------+------+-------------------+-------------------
↪-----------------------------------------------------------------+
| 9f57e5a7-bfda-4ae4-80e1-80528f7c9e1e |      | fa:16:3e:b5:0b:2a | {"subnet_id":
↪"e732b00d-027c-45d4-a68a-10f1535000f4", "ip_address": "172.16.6.38"} |
+--------------------------------------+------+-------------------+-------------------
↪-----------------------------------------------------------------+
```

Although not shown, here the situation is analogous for `router1_HA_backup_1` and `Global-router-0000-000000000005`. They are both scheduled to hosting device `00000000-0000-0000-0000-000000000005`.

Configuration Replay onto ASR1k Router

The CFGA performs a keep-alive against each ASR1k router that it manages. If communication is lost due to router reboot or loss of network connectivity, it continues to check for a sign of life. Once the router recovers, the CFGA will replay all neutron specific configurations for this router. Similarly, if a CFGA is restarted, the neutron specific configuration for all ASR1k routers it manages are replayed. Other configurations in the router are not touched by the replay mechanism.

The time period to perform keep-alives for each router can be altered by the configuration variable `heartbeat_interval` defined under the section header `[cfg_agent]`. If this feature is not wanted, the configuration variable `enable_heartbeat` should be set to `False` which disables it. Refer to the *ASR1000 Configuration Sample* for more details on these settings.

High-Availability for Configuration Agents

Since no configurations can be made to an ASR1k router if the CFGA managing that router is dead, a high-availability mechanism is implemented for CFGA. The CFGA HA requires that at least two CFGA are deployed. If a CFGA dies, the DMP will select another CFGA to take over management of the hosting devices (the ASR1k routers) that were managed by the dead CFGA. The detailed activities are described in the remainder of this section.

Whenever a neutron REST API update operation is performed on a neutron router, a notification will be sent to the CFGA managing the ASR1k that hosts the neutron router. At that point, the status of the CFGA is checked. If the CFGA has not sent a status report recently, it is considered dead and the hosting device will be unassigned from that CFGA. The time interval after which a device is considered dead can be modified using the `cfg_agent_down_time` configuration option.

After that, an attempt to reschedule the hosting devices to another CFGA will be performed. If it succeeds, the hosting device will be assigned to that CFGA and then the notification will be sent. If not, the hosting device will not be assigned to any config agent but new re-scheduling attempts will be performed periodically.

Every 20 seconds (configurable through the configuration option `cfg_agent_monitoring_interval`), any CFGA that has not been checked in the last 20 seconds (because of a notification) will be checked. If the CFGA is determined to be dead, all hosting devices handled by that CFGA will be unassigned from that CFGA.

An attempt to reschedule each of those hosting devices to other CFGA will be performed. Those attempts that succeed will result in the corresponding ASR1k router being assigned to the CFGA returned by the scheduler. Those attempts that fail will result in the ASR1k remaining unassigned.

Hence, an ASR1k will either be rescheduled as a consequence of a neutron router notification or by the periodic CFGA status check.

Scheduling of hosting devices to configuration agents

Two hosting device-to-CFGA schedulers are available. The `configuration_agent_scheduler_driver` configuration option in the `[general]` section determines which scheduler the L3P uses.

Random

- Hosting device is randomly assigned to the first available CFGA
- Two hosting devices can end up being assigned to the same CFGA
- `configuration_agent_scheduler_driver = networking_cisco.plugins.cisco.device_manager.scheduler.hosting_device_cfg_agent_scheduler.HostingDeviceCfgAgentScheduler`

Load-balanced

- Attempts to load-balance hosting devices across available CFGA
- A hosting device is assigned to the CFGA managing the least number of hosting devices
- `configuration_agent_scheduler_driver = networking_cisco.plugins.cisco.device_manager.scheduler.hosting_device_cfg_agent_scheduler.StingyHostingDeviceCfgAgentScheduler`

Troubleshooting

- To triage issues and verify that the L3P, DMP, CFGA, and the ASR1k routers are operating correctly, the following steps can be performed:
  1. Use the `neutron agent-list` command to make sure that at least one CFGA (i.e., `neutron-cisco-cfg-agent`) is running with alive state showing `:-)`. Also ensure that any deployed L3 agent (i.e., `neutron-l3-agent`) is disabled, indicated by alive state of `xxx`:

```
(keystone_admin)$ neutron agent-list
+--------------------------------------+--------------------+----------------
↪-+-------+----------------+---------------------------+
| id                                   | agent_type         | host           ␣
↪ | alive | admin_state_up | binary                    |
+--------------------------------------+--------------------+----------------
↪-+-------+----------------+---------------------------+
| 019fdca0-6310-43f6-ae57-005fbbd1f672 | L3 agent           | tme166.cisco.
↪com | xxx   | True           | neutron-l3-agent          |
| 1595c8ce-3ec5-4a01-a1d8-c53cd0cd4970 | DHCP agent         | tme166.cisco.
↪com | :-)   | True           | neutron-dhcp-agent        |
```

```
| 61971f98-75f0-4d03-a130-88f7228c51a1 | Open vSwitch agent | tme167.cisco.
↪com | :-)   | True           | neutron-openvswitch-agent |
| 8d0de547-a7b8-4c33-849b-b0a7e38198b0 | Metadata agent     | tme166.cisco.
↪com | :-)   | True           | neutron-metadata-agent    |
| cdfc51b4-88b6-4d84-bfa3-2900914375cc | Open vSwitch agent | tme166.cisco.
↪com | :-)   | True           | neutron-openvswitch-agent |
| fbc8f44b-64cd-4ab1-91d8-32dbdf10d281 | Cisco cfg agent    | tme166.cisco.
↪com | :-)   | True           | neutron-cisco-cfg-agent   |
+--------------------------------------+--------------------+-----------------
↪-+-------+----------------+---------------------------+
```

2. If cisco-cfg-agent is not running [xxx] then check the output of **`systemctl status neutron-cisco-cfg-agent.service`** to make sure that its loaded and active or any errors that it shows.
3. Check the logs for config-agent at `/var/log/neutron/cisco-cfg-agent.log` and see if there are any errors or tracebacks.
4. Verify that a hosting-device-template for ASR1k routers is defined:

```
(keystone_admin)$ neutron cisco-hosting-device-template-list
+--------------------------------------+-----------------+---------------+----
↪-----------+---------+
| id                                   | name            | host_category |␣
↪service_types | enabled |
+--------------------------------------+-----------------+---------------+----
↪-----------+---------+
| 00000000-0000-0000-0000-000000000001 | NetworkNode     | Network_Node  |␣
↪router:FW:VPN | True    |
| 00000000-0000-0000-0000-000000000003 | ASR1k template  | Hardware      |␣
↪router        | True    |
+--------------------------------------+-----------------+---------------+----
↪-----------+---------+
```

Note: The above command must be performed as administrator.

If the Cisco extensions to neutronclient are not installed a query to the neutron `cisco_hosting_device_templates` DB table can instead be performed. The following shows how this is done when MySQL is used:

```
$ mysql -e "use neutron; select * from cisco_hosting_device_templates;"
```

5. Verify that the ASR1k routers are registered in the device repository:

```
(keystone_admin)$ neutron cisco-hosting-device-list
+--------------------------------------+----------------+---------------------
↪-----------------+----------------+--------+
| id                                   | name           | template_id         ␣
↪                 | admin_state_up | status |
+--------------------------------------+----------------+---------------------
↪-----------------+----------------+--------+
| 00000000-0000-0000-0000-000000000003 | ASR1k device 1 | 00000000-0000-0000-
↪0000-000000000003 | True           | ACTIVE |
| 00000000-0000-0000-0000-000000000004 | ASR1k device 2 | 00000000-0000-0000-
↪0000-000000000003 | True           | ACTIVE |
+--------------------------------------+----------------+---------------------
↪-----------------+----------------+--------+
```

```
```

Note: The above command must be performed as administrator.

Alternatively, as a DB query:

```
$ mysql -e "use neutron; select * from cisco_hosting_devices;"
```

6. Verify that a router type for ASR1k routers is defined:

```
(keystone_admin)$ neutron cisco-router-type-list
+--------------------------------------+-------------------------+-----------
↪-------------------------------------------+-------------------------------
↪-------+
| id                                   | name                    |␣
↪description                                           | template_id          ␣
↪                  |
+--------------------------------------+-------------------------+-----------
↪-------------------------------------------+-------------------------------
↪-------+
| 00000000-0000-0000-0000-000000000001 | Namespace_Neutron_router | Neutron␣
↪router implemented in Linux network namespace | 00000000-0000-0000-0000-
↪000000000001 |
| 00000000-0000-0000-0000-000000000003 | ASR1k_router             | Neutron␣
↪router implemented in Cisco ASR1k device      | 00000000-0000-0000-0000-
↪000000000003 |
+--------------------------------------+-------------------------+-----------
↪-------------------------------------------+-------------------------------
↪-------+
```

Alternatively, do:

```
$ mysql -e "use neutron; select * from cisco_router_types;"
```

7. Verify that there is ip connectivity between the controllers and the ASR1K routers.
8. Check the netconf sessions on the ASR1K using the `show netconf session` command.
9. Collect logs from `/var/log/neutron/server.log` and `/var/log/neutron/cisco-cfg-agent.log`.
10. If new code is being pulled for bug fixes, run the steps in the section *Installing the networking-cisco Package* and restart neutron and configuration agent services.

- The hosting-device states reported by the CFGA and their meaning are as follows:

  ACTIVE Active means the hosting device is up, responds to pings and is configurable.

  NOT RESPONDING Not responding means the hosting device does not respond to pings but has not yet been determined to be dead or faulty.

  ERROR Error means the hosting device has been determined to be faulty; meaning it may respond to pings but other symptoms indicate it is faulty.

  DEAD Dead means the hosting device has been determined to be dead in that it does not respond to pings even given multiple, repeated attempts.

3.2.2 Nexus Mechanism Driver Administration Guide

There are two ways to configure the Nexus ML2 Mechanism driver either directly in the neutron configuration files or via TripleO config for OpenStack on OpenStack configurations.

This guide focuses on the neutron start-up files then follows up with samples of Tripleo configuration files. You will find similarities between the neutron start-up files and Tripleo sample configurations since tripleo config files ultimately cause the generation of neutron start-up configuration files. These neutron start-up files are most often placed beneath the directory `/etc/neutron/plugins/ml2` on the controller node.

For a description of what activites are performed by the Nexus Driver for VLAN and VXLAN configuration, refer to *Nexus Mechanism Driver Overview and Architecture* documentation.

Configuring neutron directly for Nexus

VLAN Configuration

To configure the Nexus ML2 Mechanism Driver for use with neutron VLAN networks, do the following:

1. Update the neutron configuration file commonly named `ml2_conf.ini` with sample configuration described in this document. This file is most commonly found in the directory `/etc/neutron/plugins/ml2`.

   Note: Cisco specific ML2 configuration may be isolated in the file `ml2_conf_cisco.ini` file while keeping neutron specific configuration parameters in file `ml2_conf.ini`.

2. Add the Nexus switch information to the configuration file. Multiple switches can be configured in this file as well as multiple OpenStack hosts for each switch. This information includes:

   - The IP address of the switch
   - The Nexus switch credential username and password
   - The OpenStack hostname and Nexus port of the node that is connected to the switch (For non-baremetal only)
   - vpc ids pool (baremetal only). It is required when automated port-channel creation is desired.
   - intfcfg_portchannel (baremetal only). This is an optional config which allows the user to custom configure port-channel as they are getting created. The custom config will substitute the default config **`spanning-tree port type edge trunk;no lacp suspend-individual`**. See *VLAN Creation* for more details on what gets configured during port-channel creation.

   For detail description of the Nexus mechanism driver options in the neutron configuration files, refer to *Nexus Configuration Reference*.

3. Include the configuration file on the command line when the neutron-server is started. For example:

```
/usr/local/bin/neutron-server --config-file /etc/neutron/neutron.conf --config-
↪file /etc/neutron/plugins/ml2/ml2_conf.ini  --config-file /etc/neutron/plugins/
↪ml2/ml2_conf_cisco.ini
```

Sample VM configuration with ethernet interfaces

The sample configuration which follows contains configuration for VMs. Both baremetal and VM configurations can co-exist at the same time but this section illustrates config for VMs only. For configuration activities performed during VLAN creation and removal, refer to *VLAN Creation* and *VLAN Removal* sections.

```
[DEFAULT]
service_plugins = trunk
<SNIP other neutron config>

[ml2]
#- This neutron config specifies to use vlan type driver and uses
#  Cisco nexus mechanism driver.
type_drivers = vlan
tenant_network_types = vlan
mechanism_drivers = openvswitch,cisco_nexus

#- This extension driver must be enabled when the mechanism
#  driver includes nexus.
extension_drivers = cisco_providernet_ext

#- This neutron config specifies the vlan range to use.
[ml2_type_vlan]
network_vlan_ranges = physnet1:1400:3900

[ml2_cisco]
#- switch_heartbeat_time is optional since it now defaults to 30 seconds
#  where previously it defaulted to 0 for disabled.  This causes a
#  keep-alive event to be sent to each Nexus switch for the amount of
#  seconds configured. If a failure is detected, the configuration will be
#  replayed once the switch is restored.
switch_heartbeat_time = 30

#- Beneath this section header 'ml2_mech_cisco_nexus:' followed by the IP
#  address of the Nexus switch are configuration which only applies to
#  this switch.
[ml2_mech_cisco_nexus:192.168.1.1]

#- Provide the Nexus login credentials
username=admin
password=mySecretPasswordForNexus

#- Non-baremetal config only - Hostname and port used on the switch for
#  this OpenStack host.  Where 1/2 indicates the "interface ethernet 1/2"
#  port on the switch and host-1 is the OpenStack host name.
host_ports_mapping=host-1:[1/2]

#- Setting the https_verify option below to False is highly discouraged
#  for use in a production setting. This would make the communication
#  path vulnerable to man-in-the-middle attacks.  The default is True
#  for a secure path.  Also provide a location of your chosen CA's
#  certificates to avoid use of a random list of CAs provided by distros.
#  This configuration is set for each Nexus switch.
https_verify=True
https_local_certificate=/home/user/certs
```

Sample VM configuration with vPC interfaces

In addition to supporting ethernet interfaces, multi-homed hosts using vPC configurations are supported. To configure this for non-baremetal case, the administrator must do some pre-configuration on the Nexus switch and the OpenStack host. These prerequisites are as follows:

1. The vPC must already be configured on the Nexus 9K device as described in Nexus9K NXOS vPC Cfg Guide.
2. The data interfaces on the OpenStack host must be bonded. This bonded interface must be attached to the external bridge.

The only variance from the ethernet configuration shown previously is the host to interface mapping so this is the only change shown below for non-baremetal configuration:

```
[ml2_mech_cisco_nexus:192.168.1.1]
host_ports_mapping=host-1:[port-channel2]

[ml2_mech_cisco_nexus:192.168.2.2]
host_ports_mapping=host-1:[port-channel2]
```

Sample VM configuration with multiple ethernet interfaces

There are some L2 topologies in which traffic from a physical server can come into multiple interfaces on the ToR switch configured by the Nexus Driver. In the case of server directly attached to ToR, this is easily taken care of by port-channel/bonding. However, if an intermediary device (e.g. Cisco UCS Fabric Interconnect) is placed between the server and the Top of Rack switch, then server traffic has the possibility of coming into multiple interfaces on the same switch. So the user needs to be able to specify multiple interfaces per host.

The following shows how to configure multiple interfaces per host. Since only the host to interface mapping is the only variance to the ethernet configuration, only the change to host to interface mapping is shown.

```
[ml2_mech_cisco_nexus:192.168.1.1]
host_ports_mapping=host-1:[1/11,1/12]
```

Sample Baremetal configuration for ethernet interfaces

The sample configuration which follows contains configuration for baremetal ethernet interfaces. Baremetal and VM configurations can co-exist at the same time. For baremetal only deployments, the host to interface mapping configurations are omitted. Instead, it does require additional dns configurations to derive host name to interface mappings. The dns changes include 1) adding `L3RouterPlugin` to `service_plugins`, 2) include `dns_domain`, and 3) add `dns` to `extension_drivers` all which are shown below. For configuration activities performed during VLAN creation and removal, refer to *VLAN Creation* and *VLAN Removal* sections.

```
[DEFAULT]
service_plugins = trunk,neutron.services.l3_router.l3_router_plugin.L3RouterPlugin
dns_domain = sample-dns-domain.com.
<SNIP other neutron config>

[ml2]
#- This neutron config specifies to use vlan type driver and uses
#  Cisco nexus mechanism driver.
type_drivers = vlan
tenant_network_types = vlan
mechanism_drivers = openvswitch,cisco_nexus


#- The dns extension driver is a baremetal only configuration.
#  It is used to acquire host name for every transaction set.
extension_drivers = dns,cisco_providernet_ext

#- This neutron config specifies the vlan range to use.
```

```
[ml2_type_vlan]
network_vlan_ranges = physnet1:1400:3900

[ml2_cisco]
#- switch_heartbeat_time is optional since it now defaults to 30 seconds
#  where previously it defaulted to 0 for disabled.  This causes a
#  keep-alive event to be sent to each Nexus switch for the amount of
#  seconds configured. If a failure is detected, the configuration will be
#  replayed once the switch is restored.
switch_heartbeat_time = 30

#- Beneath this section header 'ml2_mech_cisco_nexus:' followed by the IP
#  address of the Nexus switch are configuration which only applies to
#  this switch.
[ml2_mech_cisco_nexus:192.168.1.1]

#- Provide the Nexus login credentials
username=admin
password=mySecretPasswordForNexus

#- Setting the https_verify option below to False is highly discouraged
#  for use in a production setting. This would make the communication
#  path vulnerable to man-in-the-middle attacks.  The default is True
#  for a secure path.
https_verify=True
```

Sample baremetal configuration for vPC

The snippet of configuration in this section is additional config which can be added to the previous baremetal ethernet config. It enables the Nexus Driver to create port-channels and apply channel-groups to the ethernet interfaces on the Nexus switch.

Nexus driver only configures the vPC bonding on the Nexus switch. It does not configure bonding on the baremetal server. This must be pre-configured in one of two ways:

1. The network config is passed into the instance using config-drive from nova/ironic. Therefore, if the instance has something like cloud-init or glean which can read the config-drive it'll set up the bond.
2. If the instance image doesn't have one of those tools then it is down to the tenant/owner of the instance to set it up manually.

When the baremetal server interfaces are bonded, multiple ethernet interfaces are sent in the Ml2 transactions to the Nexus driver. The driver checks these interfaces against the Nexus switch to determine if the administrator has preconfigured channel-groups. If preconfigured, the driver learns that these interfaces are already configured. The alternative to preconfiguring port-channels and channel-groups, the administrator can configure the options `vpc_pool` and optionally `intfcfg_portchannel` below which will cause the driver to create a port-channel and apply channel-groups for you. For more details on these parameters, refer to the *Nexus Configuration Reference*.

For configuration activities performed during VLAN creation and removal, refer to *VLAN Creation* and *VLAN Removal* sections.

```
#- Beneath this section header 'ml2_mech_cisco_nexus:' followed by the IP
#  address of the Nexus switch are configuration which only applies to
#  this switch.
[ml2_mech_cisco_nexus:192.168.1.1]

#- Provide the Nexus login credentials
```

```
username=admin
password=mySecretPasswordForNexus

#- Baremetal config only - Provide pool of vpc ids for use when creating
#  port-channels.  The following allows for a pool of ids 1001 thru 1025
#  and also 1030.
vpc_pool=1001-1025,1030

#- Baremetal config only - Provide custom port-channel Nexus 9K commands
#  for use when creating port-channels for baremetal events.
intfcfg_portchannel=no lacp suspend-individual;spanning-tree port type edge trunk
```

VXLAN Overlay Configuration

Limitations

VXLAN Overlay Configuration is supported on normal VM configurations and not baremetal. Because of this, host to interface mapping in the ML2 Nexus configuration section is always required.

Prerequisites

The Cisco Nexus ML2 driver does not configure the features described in the "Considerations for the Transport Network" section of Nexus9K NXOS VXLAN Cfg Guide. The administrator must perform such configuration before configuring the Nexus driver for VXLAN. Do all of the following that are relevant to your installation:

- Configure a loopback IP address
- Configure IP multicast, PIM, and rendezvous point (RP) in the core
- Configure the default gateway for VXLAN VLANs on external routing devices
- Configure VXLAN related feature commands: **`feature nv overlay`** and **`feature vn-segment-vlan-based`**
- Configure NVE interface and assign loopback address

Nexus Driver VXLAN Configuration

To support VXLAN configuration on a top-of-rack Nexus switch, add the following additional Nexus Driver configuration settings:

1. Configure an additional setting named `physnet` under the `ml2_mech_cisco_nexus` section header.
2. Configure the VLAN range in the `ml2_type_vlan` section as shown in the Sample which follows. The `ml2_type_vlan` section header format is defined in the `/etc/neutron/plugins/ml2/ml2_conf.ini`.
3. Configure the network VNI ranges and multicast ranges in the `ml2_type_nexus_vxlan` section. These variables are described in more detail in *Nexus Configuration Reference*.

Sample VXLAN configuration with Ethernet interfaces

```
[ml2]
#- This neutron config specifies to use nexus_vxlan,vlan type driver
#  and use cisco nexus mechanism driver.
type_drivers = nexus_vxlan,vlan
tenant_network_types = nexus_vxlan
mechanism_drivers = openvswitch,cisco_nexus

#- This extension driver must be enabled when the mechanism
#  driver includes nexus.
extension_drivers = cisco_providernet_ext

[ml2_type_vlan]
network_vlan_ranges = physnet1:100:109

[ml2_mech_cisco_nexus:192.168.1.1]
# Provide the Nexus log in information
username=admin
password=mySecretPasswordForNexus

# Hostname and port used on the switch for this OpenStack host.
# Where 1/2 indicates the "interface ethernet 1/2" port on the switch.
host_ports_mapping=host-1:[1/2]

# Where physnet1 is a physical network name listed in the ML2 VLAN
# section header [ml2_type_vlan].
physnet=physnet1

# Setting the https_verify option below to False is highly discouraged
# for use in a production setting. This would make the communication
# path vulnerable to man-in-the-middle attacks.  The default is True
# for a secure path. Also provide a location of your chosen CA's
#  certificates to avoid use of a random list of CAs provided by distros.
https_verify=True
https_local_certificate=/home/user/certs

[ml2_type_nexus_vxlan]
# Comma-separated list of <vni_min>:<vni_max> tuples enumerating
# ranges of VXLAN VNI IDs that are available for tenant network allocation.
vni_ranges=50000:55000

# Multicast groups for the VXLAN interface. When configured, will
# enable sending all broadcast traffic to this multicast group.
# Comma separated list of min:max ranges of multicast IP's
# NOTE: must be a valid multicast IP, invalid IP's will be discarded
mcast_ranges=225.1.1.1:225.1.1.2
```

Additional configuration when the DHCP agent is not running on the Network Node

If a DHCP Agent is not running on the network node then the network node physical connection to the Nexus switch must be added to all OpenStack hosts that require access to the network node. As an example, if the network node is physically connected to Nexus switch 192.168.1.1 port 1/10 then the following configuration is required.

```
<SKIPPED Other Config defined in VLAN/VXLAN sections>
[ml2_mech_cisco_nexus:192.168.1.1]
```

```
host_ports_mapping=ComputeHostA:[1/8,1/10],ComputeHostB:[1/9,1/10]
username=admin
password=secretPassword
physnet=physnet1
https_verify=True  # for secure path if certificate available
https_local_certificate=/home/user/certs  # location of CA certificates

[ml2_mech_cisco_nexus:192.168.1.2]
host_ports_mapping=ComputeHostC:[1/10]
username=admin
password=secretPassword
physnet=physnet1
https_verify=True  # for secure path if certificate available
https_local_certificate=/home/user/certs  # location of CA certificates
```

Configuring neutron via OpenStack on OpenStack (TripleO) for Nexus

VLAN Configuration

The Cisco specific implementation is deployed by modifying the tripleo environment file Tripleo Nexus Ucsm Env File and updating the contents with the deployment specific content. Note that with TripleO deployment, the server names are not known before deployment so the MAC address of the server must be used in place of the server name. Descriptions of the parameters can be found at Tripleo Nexus Ucsm Parm file. In this file, you can see how the parameters below are mapped to neutron variables. More details on these neutron variable names can be found in *Nexus Configuration Reference*.

```
resource_registry:
  OS::TripleO::AllNodesExtraConfig: /usr/share/openstack-tripleo-heat-templates/
↪puppet/extraconfig/all_nodes/neutron-ml2-cisco-nexus-ucsm.yaml

parameter_defaults:
  NeutronMechanismDrivers: 'openvswitch,cisco_nexus'
  NetworkNexusConfig: {
    "N9K-9372PX-1": {
        "ip_address": "192.168.1.1",
        "nve_src_intf": 0,
        "password": "mySecretPasswordForNexus",
        "physnet": "datacentre",
        "servers": {
            "54:A2:74:CC:73:51": {
                "ports": "1/2"
            }
        },
        "username": "admin",
        "vpc_pool": "1001-1025,1030",
        "intfcfg_portchannel": "no lacp suspend-individual;spanning-tree port type␣
↪edge trunk",
        "https_verify": "true",
        "https_local_certification": "/home/user/certs"
    }
  }
  NetworkNexusManagedPhysicalNetwork: datacentre
  NetworkNexusSwitchHeartbeatTime: 30
  NetworkNexusProviderVlanAutoCreate: 'true'
  NetworkNexusProviderVlanAutoTrunk: 'true'
```

```
  NetworkNexusVxlanGlobalConfig: 'false'
  NeutronNetworkVLANRanges: 'datacentre:2000:2500'
  NetworkNexusVxlanVniRanges: '0:0'
  NetworkNexusVxlanMcastRanges: '0.0.0.0:0.0.0.0'
  NeutronPluginExtensions: 'cisco_providernet_ext'
```

VXLAN Configuration

The Cisco specific implementation is deployed by modifying the tripleO environment file Tripleo Nexus Ucsm Env File and updating the contents with the deployment specific content. Note that with TripleO deployment, the server names are not known before deployment. Instead, the MAC address of the server must be used in place of the server name. Descriptions of the parameters can be found at Tripleo Nexus Ucsm Parm file. In this file, you can see how the parameters below are mapped to neutron variables. With these neutron variable names, more details can be found in *Nexus Configuration Reference*.

```
resource_registry:
  OS::TripleO::AllNodesExtraConfig: /usr/share/openstack-tripleo-heat-templates/
↪puppet/extraconfig/all_nodes/neutron-ml2-cisco-nexus-ucsm.yaml

parameter_defaults:
  NeutronMechanismDrivers: 'openvswitch,cisco_nexus'
  NetworkNexusConfig: {
    "N9K-9372PX-1": {
        "ip_address": "192.168.1.1",
        "nve_src_intf": 0,
        "password": "secretPassword",
        "physnet": "datacentre",
        "servers": {
            "54:A2:74:CC:73:51": {
                "ports": "1/10"
            }
        },
        "username": "admin",
        "https_verify": "true",
        "https_local_certification": "/home/user/certs"
    }
   "N9K-9372PX-2": {
        "ip_address": "192.168.1.2",
        "nve_src_intf": 0,
        "password": "secretPassword",
        "physnet": "datacentre",
        "servers": {
            "54:A2:74:CC:73:AB": {
                "ports": "1/10"
            }
           "54:A2:74:CC:73:CD": {
                "ports": "1/11"
            }
        },
        "username": "admin",
        "https_verify": "true",
        "https_local_certification": "/home/user/certs"
    }
  }

  NetworkNexusManagedPhysicalNetwork: datacentre
```

```
  NetworkNexusSwitchHeartbeatTime: 30
  NetworkNexusProviderVlanAutoCreate: 'true'
  NetworkNexusProviderVlanAutoTrunk: 'true'
  NetworkNexusVxlanGlobalConfig: 'false'
  NeutronNetworkVLANRanges: 'datacentre:2000:2500'
  NetworkNexusVxlanVniRanges: '50000:55000'
  NetworkNexusVxlanMcastRanges: '225.1.1.1:225.1.1.2'
  NeutronPluginExtensions: 'cisco_providernet_ext'
```

Note: If setting `NetworkNexusManagedPhysicalNetwork`, the per-port `physnet` value needs to be the same as `NetworkNexusManagedPhysicalNetwork`.

Additional configuration when the DHCP agent is not running on the Network Node

The following is the Tripleo version of configuring what was described in the section *Additional configuration when the DHCP agent is not running on the Network Node*.

```
<Skipped other config details defined in VLAN/VXLAN sections>

parameter_defaults:
  NeutronMechanismDrivers: 'openvswitch,cisco_nexus'
  NetworkNexusConfig: {
    "N9K-9372PX-1": {
        "ip_address": "192.168.1.1",
        "nve_src_intf": 0,
        "password": "secretPassword",
        "physnet": "datacentre",
        "servers": {
            "54:A2:74:CC:73:51": {
                "ports": "1/10"
            }
        },
        "username": "admin",
        "https_verify": "true",
        "https_local_certification": "/home/user/certs"
    }
    "N9K-9372PX-2": {
        "ip_address": "192.168.1.2",
        "nve_src_intf": 0,
        "password": "secretPassword",
        "physnet": "datacentre",
        "servers": {
            "54:A2:74:CC:73:AB": {
                "ports": "1/10"
           }
           "54:A2:74:CC:73:CD": {
                "ports": "1/11"
            }
        },
        "username": "admin",
        "https_verify": "true",
        "https_local_certification": "/home/user/certs"
    }
  }
```

```
<Skipped other config details defined in VLAN/VXLAN sections>
```

Configuration Replay applied to the Nexus Switch

The Nexus mechanism driver performs a keep-alive against each known Nexus switch every 30 seconds. If communication is lost due to switch reboot or loss of network connectivity, it continues to check for a sign of life. Once the switch recovers, the Nexus driver will replay all known configuration for this switch. If neutron restarts, configuration for all known Nexus switches is replayed. The time period to perform keep-alives for each switch can be altered by the configuration variable `switch_heartbeat_time` defined under the section header `[ml2_cisco]`. If this feature is not wanted, the variable should be set to 0 which disables it. Refer to the *Nexus Configuration Reference* for more details on this setting.

Provider Network Limited Operations

The OpenStack/network administrator may want to control how the OpenStack create, update and delete port events program the Nexus switch for provider networks. Two configuration variables are available to address limiting the actions taken for provider networks during port events. The variables are defined under the `[ml2_cisco]` section header. These variables depend on the *extension_drivers* being set to *cisco_providernet_ext* beneath the `[ml2]` section header.

```
[ml2_cisco]
# Provider VLANs are automatically created as needed on the Nexus switch.
provider_vlan_auto_create=[True|False]

# Provider VLANs are automatically trunked as needed on the ports of the
# Nexus switch.
provider_vlan_auto_trunk=[True|False]
```

For more information on provider networks, refer to the Provider Networks OpenStack documentation.

Neutron Trunk Support

Nexus driver support for the neutron trunk feature consists of the driver programming the trunk parent port's and all subport's network segmentation ID(s) on the switch. (See *VLAN Creation* for VLAN programming details.)

The VLAN IDs described in this section are the same IDs used for all Layer-2 configuration. The segmentation ID assigned to a VLAN network segment is used to program the switch on neutron port events. These port events are triggered when Nova instances are created, updated or deleted.

Note that the segmentation IDs assigned from the `openstack network trunk set` command are not used to configure the nexus top-of-rack switch. Example:

```
$ openstack network trunk set --subport port=<port ID>, segmentation-type=vlan,
  segmentation-id=<vlan ID> <trunk ID>
```

These VLAN IDs are used by instances attached to a virtual switch (ex. OVS).

In baremetal deployments, the trunk parent port's network segmentation ID will be programmed on the nexus switch as both `switchport trunk native` and `switchport trunk allowed`. For trunk subports, only `switchport trunk allowed` is programmed. For VM deployments, `switchport trunk allowed` is programmed on the switch for both the parent and subports of the trunk.

There are no specific nexus configuration variables required for trunk support. To enable neutron trunking, the neutron `service_plugin` configuration variable must include the `trunk` plugin.

For more configuration and usage information on the neutron trunk feature refer to the Neutron/TrunkPort and neutron Trunking OpenStack documentation.

Troubleshooting

Introduction

The details in this section identify common problems which can be encountered, error messages that can be seen for each problem, and the actions the user can take to resolve each problem. A good place to start triaging is to peruse the neutron log file for error messages by searching for strings such as `ERROR` or `Traceback`. If you find a Nexus driver error in the log file, you can search this guide for snippets from the log message.

All Nexus Mechanism Driver log messages appear in the same log file as neutron. To isolate Nexus log messages from other neutron log entries, grep on 'nexus'. The location of OpenStack log messages vary according to each install implementation.

At times, the problems can not be resolved by the administrator and requires intervention by Cisco Tech Support. If this is the only recourse left, then gather the following information to provide to Tech Support so they can better assist you.

- If an installer is being used for deployment, identify what installer is being used and provide a copy of its log files.
- Provide compressed OpenStack log files:

```
tar -xvfz openstack-log-files.tar.gz {OpenStack log directory}
```

- Provide a copy of the current configuration of all participating Nexus Switches in your network. This can be done with the Nexus command:

```
copy run off-load-nexus-config-for-viewing
```

- Dump content of Nexus driver databases into files using commands defined in *How to view Nexus Driver databases*.
- Provide a network diagram with connection details.

How to view Nexus Driver databases

To help triage issues, it may be helpful to peruse the following database tables:

1. To view the content of the Nexus Driver port binding database table: In addition to port entries, the switch state is also saved in here. These special entries can be identified with an instance_id of `RESERVED_NEXUS_SWITCH_DEVICE_ID_R1`.

```
mysql -e "use neutron; select * from cisco_ml2_nexusport_bindings;"
```

2. To view the content of the Nexus Driver port mapping database table:

```
mysql -e "use neutron; select * from cisco_ml2_nexus_host_interface_mapping;"
```

3. To view the content of the Nexus Driver VPC ID port database table:

```
mysql -e "use neutron; select * from cisco_ml2_nexus_vpc_alloc;"
```

4. To view the content of the Nexus Driver VNI allocation port database table:

```
mysql -e "use neutron; select * from ml2_nexus_vxlan_allocations;"
```

5. To view the content of the Nexus Driver Mcast mapping database table:

```
mysql -e "use neutron; select * from ml2_nexus_vxlan_mcast_groups;"
mysql -e "use neutron; select * from cisco_ml2_nexus_nve;"
```

Create Event Failures

Description

As events for port creation are received, the Nexus Driver makes sure at least one of the switches for each event are active. If it fails to reach a switch, Message 1 below will appear. After checking all switches and it is determined there are no active switches needed for this event, then the exception (message 2 below) will appear and the event is rejected.

Message

```
1. Failed to ping switch ip {switch_ip} error {exp_err}
2. NexusConnectFailed: <snip> Create Failed: Port event can not be
   processed at this time.
```

Corrective Action

Refer to *corrective actions* defined in *Connection loss with Nexus Switch* for steps to narrow down why switch(s) are not active.

Update/Delete Event Failures

Description

As Update or Delete configuration events are received, there are a couple exceptions which can be raised by Nexus Driver. When events are sent to the configuration driver, they can fail during the authorization phase with the exception `NexusConnectFailed` or during the actual configuration with the exception `NexusConfigFailed`. The following illustrates what appears for these exceptions:

1. `NexusConnectFailed`: Unable to connect to Nexus {switch-ipaddr}. `Reason`: {error returned from underlying REST API or the Nexus switch}
2. `NexusConfigFailed`: Failed to configure Nexus switch: {switch-ipaddr} `Config`: REST API path: REST API body `Reason`: {error returned from underlying REST API or the Nexus switch}

Notice the `NexusConfigFailed` exception has a `Config:` parameter. This provides information of what object the driver was trying to configure (REST API path) and what value(s) the driver was trying to change (REST API body).

The exception is accompanied by a `Reason:` parameter which returns the exact error received by the Nexus REST API driver from one of two sources:

- The lower layer REST API code could return an error. See the section *Connection loss with Nexus Switch* for an example of an error from the lower layer REST API driver as well as Message 2 below.

- The error comes from the Nexus Switch itself. See the section *Missing Nexus Switch VXLAN Prerequisite Config* for an example of an error generated by Nexus Switch.

The `Reason` clause provides the details needed to narrow down the error. Since the `Reason` clause contains the specific details to the error message, it will be reduced to the following for the remainder of the *Troubleshooting* section.

Message

```
1. NexusConnectFailed: <SNIP>, Reason: Update Port Failed: Nexus Switch is
   down or replay in progress.
2. NexusConfigFailed: <SNIP>, Reason: HTTPConnectionPool(
   host={switch-ipaddr}, port=80): Read timed out. (
   read timeout=30)
```

Corrective Action

1. Check the section *Connection loss with Nexus Switch* for the most likely lower layer REST API error. Message 2 above is an example of the output you would see.
2. Errors returned by the Nexus switch cannot be documented in this section. You can determine what object failed to update by analyzing what's in the `Config:` clause of the exception and manually applying the same action using the Nexus switch CLI.
3. The `NexusConnectFailed` error (message 1 above) is a special case where the reason is generated by Nexus Driver. In this case, the Nexus driver receives update events from neutron but configuration replay has not fully initialized or in process of reconfiguring the switch, or the switch is down. This may be a temporary glitch. Updates are resent to Nexus Driver and the switch is configured when the switch becomes active.

Connection loss with Nexus Switch

Description

The most likely error to encounter is loss of connectivity to the Nexus switch either due to Nexus switch rebooting or breakage in the network itself. One or either of the exceptions shown below can occur during configuration events. The first occurs if the driver was performing an authorization request prior to configuration. The latter occurs if the driver was attempting a configuration request. Either case will fail with a timeout error as shown in the messages listed below.

Message

```
1. NexusConnectFailed: <SNIP>, Reason: HTTPConnectionPool(
    host={switch-ipaddr}, port=80): Max retries exceeded with url:
    /api/aaaLogin.json (Caused by ConnectTimeoutError(
    Connection to {switch-ipaddr} timed out.  (connect timeout=60))
2. NexusConfigFailed: <SNIP>, Reason: HTTPConnectionPool(
    host={switch-ipaddr}, port=80): Read timed out. (read timeout=30)
```

Corrective Action

- Check if the Nexus switch is accessible from the OpenStack Controller node by issuing a ping to the Nexus Switch ip address.
- If the switch is accessible, check the Nexus port binding database as described in section *How to view Nexus Driver databases* and look for `RESERVED_NEXUS_SWITCH_DEVICE_ID_R1`. Check the following if the switch is shown as `INACTIVE`.
  1. Check the credentials configured for this switch in the neutron start-up configuration file. Make sure the switch IP address is correct and the credential information is correct. See the various configuration examples in the section *Configuring neutron directly for Nexus* for details.
  2. Check that `feature nxapi` is configured on the Nexus Switch since it is required for Nexus Mechanism driver to use the REST API Config driver.
- If the switch is not accessible, isolate where in the network a failure has occurred.
  1. Is Nexus Switch management interface down?
  2. Is there a failure in intermediary device between the OpenStack Controller and Nexus Switch?
  3. Can the next hop device be reached?
- Check if the switch is running by accessing the console.

Configuration Replay Messages

Description

The Nexus driver periodically performs a get request to the Nexus switch to make sure the communication path is open. A log message (See Message 1 below) is generated the first time the get request fails. The Nexus Driver will indefinitely continue to send the get request until it is successful as indicated by log message 2 below. Once connectivity is established, the configuration for this Nexus switch is replayed and successful completion of the reconfiguration is show in the log message 3 below. If there were no port bindings found for a switch, message 4 will be seen. This may be due to no port events received for this switch and the switch state has toggled. For failures during the replay of the switch configuration, refer to the section *Replay of Configuration Data Failed*.

Message

```
1. Lost connection to switch ip {switch_ip}
2. Re-established connection to switch  ip {switch_ip}
3. Restore of Nexus switch ip {switch_ip} is complete
4. No port entries found for switch ip {switch_ip} during replay
```

Corrective Action

1. To monitor the state of the target switch from the perspective of the Nexus Driver, database commands can be used. Refer to section *How to view Nexus Driver databases* and look for `RESERVED_NEXUS_SWITCH_DEVICE_ID_R1`.
2. Fix any failed connectivity issues as described in *Connection loss with Nexus Switch*.

Replay of Configuration Data Failed

Description

The Nexus driver has detected the Nexus switch is up and it is attempting to reconfigure. Occasionally configurations will fail since the switch is not fully ready to handle configurations. Any number of the messages listed below can be seen for this failure.

Message

```
1. Unexpected exception while replaying entries for switch {switch_ip}
   Reason: {Contains error details from lower layers}
2. Unable to initialize interfaces to switch {switch_ip}
3. Replay config failed for ip {switch_ip}
4. Error encountered restoring vlans for switch {switch_ip}
5. Error encountered restoring vxlans for switch {switch_ip}
```

Corrective Action

This may be a temporary glitch and should recover on next replay retry. If the problem persists, contact Tech Support for assistance.

Nexus Switch is not getting configured

Description

The only difference between this case and what is described in the section *Connection loss with Nexus Switch* is the Nexus switch has never been successfully configured after neutron start-up. Refer to the connection loss section for more details to triage this case.

Message

There's no specific error message for this other than some shown in *Connection loss with Nexus Switch* section.

Corrective Action

There are a couple possible reasons for this issue:

- It may be due to a connection loss or never having a connection with the switch. See the *Connection loss with Nexus Switch* for more triage hints details like how to check the state of the switch and configuration errors that can occur.
- It is possible the hostname is not correctly configured in the neutron start-up file beneath the nexus switch section named *ml2_mech_cisco_nexus*. Depending on the configuration of the OpenStack host, the hostname to configure is the long name *hostname.domainname* which can be derived by running *hostname -f* on the host itself. Additionally if you enable debug in neutron start-up config file and search for the log entry *Attempting to bind port {port} on host {hostname}*, the *hostname* in this message is the same name used in Nexus look-ups. Configure this name in the neutron start-up file and restart neutron.

No Nexus Configuration in the neutron start-up file

Description

If there are no Nexus switches configured in the neutron start-up configuration file, the error message below will be seen in the neutron log file.

Message

```
No switch bindings in the port database
```

Corrective Action

1. Check Sample configurations throughout this guide on configuring switch details. Specifically look for the section header *ml2_mech_cisco_nexus*. Also refer to the *Nexus Configuration Reference*.
2. When neutron is started, make sure the Nexus configuration is in the configuration file provided to neutron at start-up.

Nexus Switch not defined in the neutron start-up file

Description

If there is Nexus configuration defined in the neutron start-up but there is nothing found for a specific switch, these messages below will be seen. Message 1 is generated for baremetal port events while message 2 is generated for non-baremetal events.

Message

```
1. Skip switch {switch_ip}.  Not configured in ini file
2. Host {switch_ip} not defined in switch configuration section.
```

Corrective Action

Check Sample configurations throughout this guide on configuring switch details. Specifically look for the section header *ml2_mech_cisco_nexus*. Also refer to the *Nexus Configuration Reference*.

Missing Nexus Switch VXLAN Prerequisite Config

Description

An attempt was made to configure **`member vni <vni-id> mcast-group <mcast-ip>`** beneath **`int nve 1`** but an error was returned by the REST API configuration driver used by the Nexus Driver. Possible reasons are:

1. Nexus switch can't find configured object. See message listed below for sample detail in reason space of exception.

2. loss of connectivity with switch. See *Connection loss with Nexus Switch*.

Message

```
Failed to configure nve_member for switch {switch_ip}, vni {vni}
    Reason: NexusConfigFailed: <SNIP>, Reason:
    {"imdata":[{ "error": { "attributes": { "code": "102",
    "text": "configured object ((Dn0)) not found
    Dn0=sys\/epId-1\/nws\/vni-70037, "}
```

Corrective Action

Some general VXLAN configuration must be in place prior to Nexus Driver driver attempting to configure vni and mcast-group configuration. Refer to the *Prerequisite* section of *VXLAN Overlay Configuration* and the section *Nexus Switch Setup* for more details.

Invalid `vpc_pool` config error

Description

The `vpc_pool` configuration parameter is a pool used for automatically creating port-channel ids for baremetal events. As *vpc_pool* is parsed, a number of errors can be detected and are reported in the messages below. For a detail description of configuring `vpc-pool` parameter, refer to *Nexus Configuration Reference*.

Message

```
1. Unexpected value {bad-one} configured in vpc-pool config
   {full-config} for switch {switchip}. Ignoring entire config.
2. Incorrectly formatted range {bad-one} config in vpc-pool
   config {full-config} for switch {switchip}. Ignoring entire config.
3. Invalid Port-channel range value {bad-one} received in vpc-pool
   config {full-config} for switch {switchip}. Ignoring entire config.
```

Corrective Action

In each message, the `{bad-one}` field is the portion of the `{full-config}` field which is failing the parsing. The `{full-config}` is what the user configured for a given `{switchip}` in the `vpc_pool` configuration parameter. Possible issues for each message can be:

1. Values in the range are not numeric. Ex: 2-abc
2. There should only be a min-max value provided. More than two values separated by '-' can not be processed. Ex: 3-5-7
3. Values in range must meet valid port-channel range on Nexus where smallest is 1 and largest is 4096. ex: 0-5 or 4090-4097

Learned Port-channel Configuration Failures for Baremetal Events

Description

If a baremetal event is received with multiple ethernet interfaces, the first in the list indicates how the rest will be treated. If it is determined the first interface is preconfigured as a member of a port-channel, the expectation is the remaining interfaces should also be preconfigured as members of the same port-channel. If this is not the case, the exception below will be raised.

Message

```
1. NexusVPCLearnedNotConsistent: Learned Nexus channel group
   not consistent on this interface set: first interface
   {first}, second interface {second}.  Check Nexus
   Config and make consistent.
2. NexusVPCExpectedNoChgrp: Channel group state in baremetal
   interface set not consistent: first interface %(first)s,
   second interface %(second)s. Check Nexus Config and make consistent.
```

Corrective Action

The message fields `{first}` and `{second}` each contain the host, interface and the channel-group learned. The `{first}` is the basis interface compared to and the `{second}` is the interface that does not match the channel-group of the `{first}`.

- Message 1 is seen when the `{first}` is a member of a channel group and `{second}` does not match channel group of the `{first}`.
- Message 2 is seen when the `{first}` is not a member of a channel group while the `{second}` is.

Log into each switch identified in `{first}` and `{second}` fields and make sure each interface is a member of the same port-channel if learning is desired. If automated port-channel creation is preferred, see *Automated Port-channel Creation Failures for Baremetal Events*.

Automated Port-channel Creation Failures for Baremetal Events

Description

Baremetal events received with multiple ethernet interfaces are treated as port-channel interfaces. The first interface in the list indicates how the rest will be treated. If all interfaces are currently not members of a port-channel, then the Nexus Driver will try and create a port-channel provided the Nexus Driver configuration parameter `vpc-pool` has been defined for each switch. For details on the activity the Nexus Driver performs to configure the port-channel, refer to *VLAN Creation*.

Message

```
1. NexusVPCAllocFailure: Unable to allocate vpcid for all switches
   {ip_list}
2. NexusVPCExpectedNoChgrp: Channel group state in baremetal
```

```
interface set not consistent: first interface {first},
{second} interface %(second)s.  Check Nexus Config and make consistent.
```

Corrective Action

1. The first exception `NexusVPCAllocFailure` will be raised if the `vpc-pool` is not configured or the pool of one of the participating switches has been depleted. The pools can be viewed using port mapping database query command as shown in *How to view Nexus Driver databases*. For details on configuring `vpc-pool` parameter, refer to *Nexus Configuration Reference*.
2. Exception 2 is raised when the `{first}` is not a member of a channel group while the `{second}` is. Log into each switch identified in `{first}` and `{second}` fields and make sure each interface is not a member of port-channel. If learning the port-channel is preferred, make sure all interfaces are configured as members to the same port-channel.

Invalid Baremetal Event

Description

A baremetal event has been received but the Nexus Driver was unable to decode the `switch_info` data in the port event. As a result, the event is ignored by the Nexus driver.

Message

```
switch_info can't be decoded {reason}
```

Corrective Action

This error should not occur and suggest looking for earlier errors in the log file. If unable to triage further from log messages, contact Tech Support for assistance.

Trunk Configuration Conflict on Nexus Switch

Description

During interface initialization, the Nexus driver collects trunking information for interfaces from the Nexus switch. This occurs at start-up for statically configured ports and on receipt of a port event for baremetal ports. The driver looks for trunking vlans configured using **`switchport trunk allowed vlan <vlanid(s)>`** and also checks if the mode type in **`switchport mode <type>`** is `trunk`.

The Nexus driver logs a warning if there are trunking vlans configured but the trunk mode is not `trunk`. The driver does not try to resolve the conflict since the correction can be done in a number of ways which requires attention from the administrator. The driver does continue to add and remove vlans to this interface. However, since the trunk mode is missing, the data traffic does not pass on this interface.

Message

Found trunk vlans but switchport mode is not trunk on Nexus switch {switch} interface {interface}. Recheck config.

Corrective Action

Look at the interface on the Nexus Switch identified in the message and check for the following possible errors.

- For VM deployments, ensure the OpenStack Nexus driver is configured with the correct interface for the intended OpenStack host.
- Ensure **`switchport mode trunk`** is configured on the interface.
- Ensure only vlans required as provider vlans or within your tenant vlan range are configured as `allowed` on the interface, and any additional vlans are removed.

Insecure Communication Path with Nexus Switch

Description

The configuration option *https_verify* became available in 5.4.0 with the default to `False` (insecure) to allow administrators to acquire certificates. The default has changed to `True` causing certificates to be verified. It is highly recommended not to disable certificate verification in production since the communication path is insecure leaving the path vunerable to man-in-the-middle attacks. If the default is changed to `False` (insecure), the warning message below is seen in the neutron log file identifying the ip address of the insecure Nexus switch.

Message

HTTPS Certificate verification is disabled. Your connection to Nexus Switch {ip} is insecure.

Corrective Action

The {ip} in the error message targets which switch is insecure and needs one or more of the following actions to secure it.

- If a publically known certificate is not currently available, apply for one from a public Certificate Authority (CA).
- If the certificate and key files have not been configured on the target Nexus switch, configure them using the Nexus Management CLI *nxapi certificate* and *enable* the certificate. For Nexus details, refer to the section *NX-API Management Commands* in the Nexus NXAPI Programmability Guide.
- Remove *https_verify=False* from the neutron start-up configuration beneath the section header [ml2_mech_cisco_nexus:your-switch-ip] for the target switch. Removing https_verify config will cause it to default to *True* to verify the public certificate.
- Add *https_local_certificate=/path/to/ca-certificates* for path to certificates for Certicate Authorities(CAs) you trust instead of a random list of certificate authorities (CAs) provided by distros.

DBDuplicate Entry - Failed Insert into cisco_ml2_nexus_host_interface_mapping

Description

When the same port-channel is configured for multiple hosts beneath the same switch, a *DBDuplicateEntry* error is seen as shown in the Message section below. This type of configuration is seen with static configurations only and not ironic. An example of such a configuration is as follows:

```
[ml2_mech_cisco_nexus:<snipped-switch-ip-addr>]
host_ports_mapping=compute-host-1:[port-channel:300],
                   compute-host-2:[port-channel:300]
```

Note: The above was represented by the now unsupported format:

```
[ml2_mech_cisco_nexus:<snipped-switch-ip-addr>]
compute-host-1 = port-channel:300
compute-host-2 = port-channel:300
```

This anomaly can also occur when there are multiple controllers which are attempting to initialize the cisco_ml2_nexus_host_interface_mapping db table at the same time.

Message

```
DBDuplicateEntry: (pymysql.err.IntegrityError)
(1062, u"Duplicate entry '<your-switch-ip>-<your-port-channel-interface>'
for key 'PRIMARY'")
[SQL: u'INSERT INTO cisco_ml2_nexus_host_interface_mapping
<SNIP>
```

Corrective Action

Both error cases described were introduced in Cisco Release 5.1.0. To eliminate these errors, upgrade to a more recent release of the networking-cisco package.

Neutron trunking feature not supported in Openstack Newton branches

Description

Cisco Nexus ML2 Mechanism driver supports trunking from tag 5.3.0; however, Openstack neutron in Newton branches and lower do not. As a result, an error can occur if baremetal configurations are attempted with these combined branches/tags. The error message which could be seen is shown below.

Message

```
TypeError: get_object() got an unexpected keyword argument 'port_id'
```

Corrective Action

Upgrade networking-cisco package or apply the changes found in https://review.openstack.org/#/c/542877/.

Exception NexusPortBindingNotFound seen in update_port_postcommit

Description

An exception NexusPortBindingNotFound is seen in update_port_postcommit when attempting to get port binding by calling get_nexusvlan_binding. This is a result of a spurious update event received while deletes are occurring for same event. It is more likely to occur when there are multiple threads and/or multiple controllers.

Message

```
networking_cisco.ml2_drivers.nexus.exceptions.NexusPortBindingNotFound:
    Nexus Port Binding (switch_ip=1.1.1.1,vlan_id=265) is not present
```

Corrective Action

The solution is to log a warning instead of raising an exception to be consistent with other ml2 drivers. To eliminate this exception, upgrade the networking-cisco package to pick-up latest fixes.

Unable to find baremetal host/interface mapping when clearing vpc resources

Description

When a port is being deleted, the vpcid is put back into the configured `vpc_pool` when it is no longer in use. During this procedure, an attempt was made to locate the ethernet interface using the dns name as hostid but it failed.

Message

```
Switch 1.1.1.1 hostid 'dns-name' host_mapping not found. Skipping port-channel clean-
↪up.
```

Corrective Action

1. If you have `None` instead of a `dns-name` in the message, perhaps you omitted the configuration to include dns. Refer to *Sample Baremetal configuration for ethernet interfaces* for dns changes needed.
2. Otherwise, this error should not occur and recommend looking for earlier errors in the log file. If unable to triage further from log messages, contact Tech Support for assistance.

Baremetal vpc id not released back into vpc pool

Description

When a port is being deleted, the vpcid is put back into the configured `vpc_pool`. In attempting to free the vpcid, it failed because it wasn't found.

Message

```
Failed to free vpcid 500 for switch 1.1.1.1 since it did not exist in table.
```

Corrective Action

This error should not occur and suggest looking for earlier errors in the log file. If unable to triage further from log messages, contact Tech Support for assistance.

3.2.3 UCSM Mechanism Driver Administration Guide

The configuration parameters for the ML2 UCSM Mechanism Driver can be specified in a configuration file along with other neutron configuration parameters. Another approach could be to use TripleO config for OpenStack on OpenStack installations.

For a description of functionalities supported by the UCSM Driver for VLAN and SR-IOV configuration, please refer to *UCSM Mechanism Driver Overview and Architecture*.

UCSM Driver configuration along with neutron parameters

1. Configuration for Cisco specific ML2 mechanism drivers can be added to the file containing neutron specific configuration, by specifying it under a driver specific section header. UCSM driver configuration needs to be under the section header `[ml2_cisco_ucsm]`. This neutron configuration file is often called `ml2_conf.ini` and frequently resides under the directory `/etc/neutron/plugins/ml2`.

   Note: It is also possible to place this configuration into a separate file for example `ml2_conf_cisco.ini` to keep these configurations separate from existing configuration in file `ml2_conf.ini`.

2. This configuration file needs to be provided on the command line when neutron-server process is started. For example:

   ```
   /usr/local/bin/neutron-server --config-file /etc/neutron/neutron.conf \
       --config-file /etc/neutron/plugins/ml2/ml2_conf.ini  \
       --config-file /etc/neutron/plugins/ml2/ml2_conf_cisco.ini
   ```

3. In a OpenStack setup with a single UCSM, it may be sufficient to use the single-UCSM format to specify the UCSM driver config with the following parameters:

   - Management IP address of the UCSM
   - Admin username to login to the UCSM
   - Admin password

- Hostname to Service Profile Mapping for all the servers that are controlled by this UCSM and are part of the OpenStack cloud.

  Note: The Service Profile (SP) associated with a server can also be a Service Profile Template (SPT). If the SP or the SPT are not created at the root level on the UCSM, the path to the SP or SPT needs to be provided as part of the above configuration.

4. List of ethernet ports or vNICs on the UCS Servers that can be used for neutron virtual port configurations. Of all the ethernet ports or vNICs available on the UCS Servers, provide only the ones that are set aside for neutron virtual port use.
5. List of vNIC Templates that are associated with neutron physical networks. This is an optional config and needs to be specified only when vNICs spread across multiple UCS Servers are all connected to a common physical network and need to be configured identically by the UCSM driver.
6. List of supported SR-IOV devices specified as a list of vendor and product IDs. This is an optional parameter and will default to the vendor and product IDs for Cisco VICs and Intel NICs.
7. For use cases where a SR-IOV port attached to a nova VM can potentially carry a list of application specific VLANs. For this configuration, the UCSM driver expects a mapping between a neutron network and the list of application specific VLANs that can be expected on a SR-IOV port on this neutron network. This is also an optional config.

   Note: The VLAN IDs associated with a neutron network should not be confused with the VLAN-id range of the neutron network itself. SR-IOV ports created on these neutron networks essentially act as trunk ports that can carry application specific traffic on VLANs specified in this config.

8. In a setup that utilizes multiple UCSMs, UCSM specific configuration parameters need to be repeated for each UCSM under a repeatable section starting with the UCSM IP specified in this format: `[ml2_cisco_ucsm_ip:<UCSM IP address>]`
9. The UCSM driver connects to all the UCS Managers provided in its configuration via a HTTPS connection where the SSL certificate on the UCS Manager is checked for validity. If there is a need to opt out of this default behavior, the parameter `ucsm_https_verify` needs to be explicity set to False. This is a global configuration and is applicable to all the UCS Manager configurations provided to the driver. Disabling SSL certificate checking makes the connection insecure and is not recommended.

Enabling SR-IOV support

The UCSM driver allows the Cisco VM-FEX capability available on UCS Managers to be leveraged in the OpenStack context. OpenStack users that have UCS servers with supported Cisco and Intel NICs can bring up VMs with SR-IOV port to carry their tenant traffic at greater speeds. The following sections provide more details about this feature.

Prerequisites for SR-IOV port support

Before the UCS Servers are purposed as compute hosts with SR-IOV ports, these hosts need to be pre-configured to have SR-IOV VFs (Virtual Functions) enabled and ready for OpenStack use and UCSM driver configuration. Here is the list of pre-requisites for SR-IOV port support:

1. UCS Servers with any of the following VICs:
   - Cisco UCS VIC 1240 (with vendor_id:product_id as 1137:0071)
   - Cisco UCS VIC 1340 (with vendor_id:product_id as 1137:0071)

- Intel 92599 10 Gigabit Ethernet Controller (with vendor_id:product_id as 8086:10ed)

2. Cisco UCS Python SDK version 0.8.2 installed on the OpenStack controller nodes. More information about the UCS SDK can be found here: Cisco UCS SDK information
3. A dynamic vNIC connection policy needs to be defined on the UCSM specifying the number of VFs the physical function (PF) should be split into. This profile also needs to specify if the VFs would be created in `direct` or `macvtap` modes. Detailed instructions for creating a Dynamic vNIC connection policy and applying it on a UCS Server vNIC can be found in UCS Manager VM-FEX configuration guide
4. Associate the Dynamic vNIC connection policy with a PF by updating its Service Profile.
5. Intel VT-x and VT-d processor extensions for virtualization must be enabled in the host BIOS. This can be achieved by adding `intel_iommu=on` to `GRUB_CMDLINE_LINUX` in `/etc/sysconfig/grub` [in RHEL] or `/etc/default/grub` [in Ubuntu].
6. After this grub.conf files on the SR-IOV capable compute hosts need to be regenerated by running **`grub2-mkconfig -o /boot/grub2/grub.cfg`** on BIOS systems or **`grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg`** on UEFI systems.
7. These SR-IOV capable compute hosts need to be rebooted. Due to this operation it is better to install OpenStack on these compute hosts after this list of pre-requisites have been completed.
8. Make sure that IOMMU is activated by running **`dmesg | grep -iE "dmar|iommu"`**. The output should include the following lines:

```
[ 0.000000] Kernel command line: BOOT_IMAGE=/vmlinuz-3.13.0-24-generic root=/dev/
↪mapper/devstack--38--vg-root ro quiet intel_iommu=on
[ 0.000000] Intel-IOMMU:enabled
```

9. Make sure the SR-IOV capable VFs are visible to kernel by running **`lspci -nn | grep Cisco`**. The output should contain several lines that look like:

```
0a:00.1 Ethernet controller [0200]: Cisco Systems Inc VIC SR-IOV VF [1137:0071]␣
↪(rev a2)
```

Configuring nova for SR-IOV

1. For nova to schedule VMs requesting SR-IOV ports, it needs to be made aware of compute hosts that have SR-IOV capable devices. This is achieved by adding the following configuration to `nova.conf` on each compute host capable of hosting SR-IOV based VMs.

```
[default]
pci_passthrough_whitelist = { "vendor_id": "<id>", "product_id": "<id>",
    "physical_network": "physnet2"}
```

2. Also, for nova to schedule VMs that request SR-IOV port(s) on a compute host, nova's scheduler should be able to filter compute hosts based on their SR-IOV capability. This is achieved by adding the following config to `nova.conf` on the controller node(s).

```
[DEFAULT]
scheduler_default_filters = RetryFilter, AvailabilityZoneFilter, RamFilter,␣
↪ComputeFilter, ComputeCapabilitiesFilter, ImagePropertiesFilter,␣
↪ServerGroupAntiAffinityFilter, ServerGroupAffinityFilter, PciPassthroughFilter
```

Troubleshooting

Introduction

This section lists some issues that could be encountered during the installation and operation of the UCS Manager driver. For each of these scenarios, there is an attempt to identify the probable root cause for that issue and a way to return to a successful state.

The UCS Manager driver logs important information regarding its operation in the neutron server log file. Please refer to this log file while trying to troubleshoot your driver installation.

The UCS Manager driver prefixes any configuration it adds to the UCS Manager with `OS-`. This driver creates VLAN Profiles, Port-Profiles and updates Service Profiles and Service Profile Templates in addition to updating vNIC Templates. For this to be successful, the UCS Manager driver should be able to connect to the UCS Manager and push down the configuration. Listed below are some common reasons why the configuration might be missing on the UCS Manager. Please refer to the neutron server log file for error messages reported by the UCS Manager driver to root cause the issue.

Connection to UCS Manager Failed: Certificate verify failed

Description

If you see that the driver is reporting a UCS Manager connection failure with the following error message, then the SSL Certificate verification on the UCS Manager has failed and this would prevent the driver from connecting to the UCS Manager.

Error Message

```
UcsmConnectFailed: Unable to connect to UCS Manager <IP address>. Reason: <urlopen␣
↪error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:590)>.
```

Corrective Action

If you want the SSL certificate check to proceed, please make sure UCS Manager has a valid SSL certificate associated with it. Instructions can be found at:

Cisco UCS Manager Administration Management Guide 3.1

SSL certificate checking can be disabled by setting the configuration variable `ucsm_https_verify` to False. This will be available starting from release 5.4.

UCS Manager network driver failed to get login credentials for UCSM

Description

The UCSM driver needs IP address and login credentials for all the UCS Managers that it needs to configure. Any issues with providing this configuration would result in connectivity issues to the UCS Manager(s).

Error Message

```
UCS Manager network driver failed to get login credentials for UCSM <IP address>
```

Corrective Action

Please check if the UCS Manager IP address, username and password are provided in the configuration file passed to neutron server process and that they are accurate. Ping the UCS Manager(s) IP address(s) from the OpenStack controller to check if you have network connectivity.

VLAN Profile with the id OS-<VLAN-id> not configured on the UCS Manager

Description

If the connection to the UCS Manager is successful, when a neutron network is created with DHCP service enabled, a VLAN Profile should be created on the UCS Manager. This configuration will program the Fabric Interconnect to send traffic on the VLAN associated with the neutron network to the TOR switch.

Corrective Action

Make sure that the neutron Network created is of type VLAN and neutron is configured to use the VLAN type driver. This configuration can be provided as follows:

```
[ml2]
type_drivers = vlan
tenant_network_types = vlan
```

VLAN configuration missing on the vNICs on either the Controller or Compute nodes

Description

Once the VLAN profiles are created, vNICs on the UCS Servers acting as OpenStack Controllers would also be updated with VLAN configuration. The vNICs on UCS Servers acting as compute hosts will be updated with VLAN configuration when VMs are created on those compute hosts.

Corrective Action

1. Check if the hostname to Service Profile mapping provided to the UCSM driver via the `ucsm_host_list` are accurate.
2. Check if the Service Profile on the UCS Manager is at the root or in a sub directory. If it is in a subdirectory, please provide the full path in the `ucsm_host_list` config.
3. Check if the Service Profile is still attached to a Service Profile Template. In that case, for the UCSM driver to be able to modify this SP, it should be unbound from the Template.
4. If the UCSM driver is required to modify the Service Profile Template, then the driver needs to be provided with the `sp_template_list` configuration.

5. The next configuration parameter to check would be the `ucsm_virtio_eth_ports`. This configuration should contain the list of vNICS on the Service Profile or the Service Profile Template that is available for the UCSM driver to configure tenant VLANs on.

VLAN configuration not deleted on the UCS Manager

Description

Just like VLAN configuration was added to the UCS Manager at different stages of Network and VM configuration, the deletion process also follows its own state machine.

Issue

Deleting a VM did not result in the removal of VLAN configuration on the UCS Manager.

Corrective Action

1. If there are other VMs still active on the compute host on the same network (hence these VMs are on the same VLAN as the one being deleted), the VLAN-id configured on vNICs on the compute hosts will not be deleted. In other words, VLAN configuration on the compute hosts will not be deleted until all the VMs on the compute host on the VLAN are deleted.
2. The global VLAN profile will be deleted only when the neutron Network associated with that VLAN-id is deleted.

Port Profiles not created on the UCS Manager

Description

When a VM is launched with an SR-IOV port, the UCSM driver responds to that request by creating Port Profiles (PP) on the UCS Manager. The PPs created by the driver are always named `OS-PP-<Vlan-id>`.

Issue

Port profile is not created on the UCS Manager.

Corrective Action

1. Run the command `lspci -nn | grep -i Cisco` on the compute nodes containing the SR-IOV capable Cisco NICs. The output should contain lines that look as follows:

```
0a:00.1 Ethernet controller [0200]: Cisco Systems Inc VIC SR-IOV VF [1137:0071]␣
↪(rev a2)
```

2. If there are no rows for Virtual Functions with vendor and product ids 1137 and 0071, it is an indication that the Dynamic vNIC Profile for that Physical Function has not been setup properly on the UCS Manager.

3. The ethernet port a.k.a as the Physical Function needs to be split into SR-IOV Virtual Functions that can be consumed by the UCSM driver. This can be achieved by attaching a Dynamic vNIC Profile where the `direct` or `macvtap` values are set. In addition, the parameter to specify the number of Virtual Functions to split the Physical Function into also needs to be provided.

   Note: Attaching a Dynamic vNIC Profile to a ethernet port on a UCS Server would result in a server reboot.

4. Check that the `intel_iommu` kernel parameter is set to `on` in the grub files on the compute node with the SR-IOV ports by running the following command:

   ```
   dmesg | grep -e DMAR -e IOMMU
   ```

   The output of the command should contain a line that says `Intel-IOMMU: enabled`.

5. Lastly, make sure that a Port Profile for that VLAN-id does not exist prior to OpenStack use. If so, OpenStack will not be able to create one for the same VLAN-id or re-use the pre-existing Port Profile.

Port Profiles not deleted on the UCS Manager

Description

The Port Profile created on the UCS Manager in response to a SR-IOV based VM, is aware of all the VMs that are currently using that Port Profile. UCS Manager learns this information by polling the UCS Servers that are attached to it. This polling interval is approximately 15 mins and is not user configurable. The Port Profile can be deleted only when they are no longer in use by any VM.

Issue

Port Profile still exists on the UCS Manager when all VMs using that Port Profile have been deleted.

Corrective Action

No manual intervention required.

Even when all the VMs using a specific Port Profile are deleted, it takes some time for the UCS Manager to learn this information because of the polling interval. The UCS Manager will not allow the UCSM driver to delete the Port Profile before this.

The UCSM driver maintains a list of Port Profiles to delete from the various UCS Managers that it is connected to. The driver also has a timer thread that wakes up every 10 minutes and attempts to delete the Port Profiles in this list. So, although the Port Profile might not get deleted right away, the UCS driver will take care of eventually deleting Port Profiles that it created when they are not in use.

3.3 Configuration Reference

This section provides a list of all configuration options for various networking_cisco drivers and plugins. These are auto-generated from networking_cisco code when this documentation was built.

The filenames listed below are just examples. Often times the neutron configuration options are combined into one or two configuration files (ex. neutron.conf, ml2_conf.ini) which are added on the neutron-server command line via the "–config-file" option.

3.3.1 Configuration Options

Nexus Mechanism Driver

The following list of variables are used by the cisco_nexus mechanism driver. A variable's default value can be changed by adding it to the configuration file included on the neutron-server command line. The sample nexus configuration file is described here, *Sample Nexus Mechanism Driver*.

ml2_cisco

managed_physical_network

Type string

Default `<None>`

When "managed_physical_network" is configured, it restricts the network segment that the nexus driver supports. Setting it to a specific network name will limit the actions taken by this driver to only that network. The network name must match a name defined in the "network_vlan_ranges" configuration. When "managed_physical_network" is not set, events for all network segments will be processed by the driver.

provider_vlan_auto_create

Type boolean

Default `true`

A flag indicating whether the Nexus driver should manage the creation and removal of VLANs for provider networks on the Nexus switches. When this flag is False, the Nexus driver will not create or remove VLANs for provider networks and the administrator needs to manage these interfaces manually or by external orchestration.

provider_vlan_auto_trunk

Type boolean

Default `true`

A flag indicating whether Nexus driver should manage the adding and removing of provider VLANs from trunk ports on the Nexus switches. When this flag is False, the Nexus driver will not add or remove provider VLANs from trunk ports and the administrator needs to manage these operations manually or by external orchestration.

switch_heartbeat_time

Type integer

Default `30`

Configuration replay is enabled by default by defining the time interval to 30 seconds. This is the amount of time to check the state of all known Nexus device(s). To disable the replay feature, set this "switch_heartbeat_time" to 0 seconds.

vxlan_global_config

Type boolean

Default `false`

A flag indicating whether the Nexus driver should manage the creating and removing of the Nexus switch VXLAN global settings of "feature nv overlay", "feature vn-segment-vlan-based", "interface nve 1" and the NVE subcommand "source-interface loopback #". When set to the default of False, the Nexus driver will not add or remove these VXLAN settings and the administrator needs to manage these operations manually or by external orchestration.

ml2_mech_cisco_nexus:<ip_address>

https_verify

Type boolean

Default `true`

This configuration option defaults to True. When https_verify is True, the directory for certification authority (CA) certificates or self-signed certificate file must be locally defined. This location is configured in https_local_certificate. To skip https certification checking, set https_verify to False though this is strongly discouraged since it makes the connection insecure. This configuration is set for each Nexus switch.

https_local_certificate

Type string

Default `<None>`

Configure either a directory to Certificate Authority certificates or a self-signed certificate file to be used with https requests. Self-signed certificates should be used temporarily when an official certificate from a trusted Certificate Authority is not yet available. In this case, the Nexus switch must also be configured and enabled with both the certificate and key files. For further details, refer to the "nxapi certificate" commands as defined in the "Nexus 9K NXAPI Programmability Guide". If the default configuration is None and if https_verify is True, the underlying SSL package will attempt to locate CA bundles using environment variables or DISTRO default locations. It is recommended that the user explicitly configure this value to avoid trusting a random list of CAs. An example configuration for certificate authorities would look like https_local_certificate=/path/to/ca_certificates. An example configuration of a self-signed certificate would look like https_local_certificate=/path/to/cafile.crt. This configuration is set for each Nexus switch.

intfcfg_portchannel

Type string

Default `<None>`

intfcfg_portchannel is a list of Nexus port-channel config CLI used when baremetal port-channels are created by the Nexus driver. It is dependent on "vpc_pool" being configured. Any number of Nexus port-channel commands separated by ";" can be provided. When there are multiple interfaces in a baremetal event, the nexus driver checks to determine whether a port-channel is already applied to the interfaces; otherwise, it creates a port channel. This optional configuration allows the administrator to custom configure the port-channel. When not configured, the nexus driver defaults to configuring "spanning-tree port type edge trunk;no lacp suspend-individual" beneath the port-channel. An example of this configuration is "intfcfg_portchannel=no lacp suspend-individual;spanning-tree port type edge trunk".

nve_src_intf

Type string

Default `<None>`

Only valid if VXLAN overlay is configured and vxlan_global_config is set to True. The NVE source interface is a loopback interface that is configured on the switch with valid /32 IP address. This /32 IP address must be known by the transient devices in the transport network and the remote VTEPs. This is accomplished by

advertising it through a dynamic routing protocol in the transport network. If nve_src_intf is not defined, a default setting of 0 is used to create "loopback0". This is configured for non-baremetal only.

password

Type string

Default <None>

The password of the Nexus Switch Administrator is required to allow configuration access to the Nexus switch.

physnet

Type string

Default <None>

This is required if Nexus VXLAN overlay feature is configured. It should be the physical network name defined in "network_vlan_ranges" (defined beneath the "ml2_type_vlan" section) that this switch is controlling. The configured "physnet" is the physical network domain that is connected to this switch. The vlan ranges defined in "network_vlan_ranges" for a physical network are allocated dynamically and are unique per physical network. These dynamic vlans may be reused across physical networks. This configuration applies to non-baremetal only.

host_ports_mapping

Type unknown type

Default <None>

A list of key:value pairs describing which host is connected to which physical port or portchannel on the Nexus switch. The format should look like: host_ports_mapping=<your-hostname>:[<intf_type><port>,<intf_type><port>], <your-second-host>:[<intf_type><port>] For example: host_ports_mapping=host-1:[ethernet1/1, ethernet1/2], host-2:[ethernet1/3], host-3:[port-channel20] Lines can be broken with indentation to ensure config files remain readable. All compute nodes must be configured while controllers are optional depending on your network configuration. Depending on the configuration of the host, the hostname is expected to be the full hostname (hostname.domainname) which can be derived by running "hostname -f" on the host itself. Valid intf_types are "ethernet" or "port-channel". The default setting for <intf_type> is "ethernet" and need not be added to this setting. This configuration applies to VM deployments only.

username

Type string

Default <None>

The username of the Nexus Switch Administrator is required to allow configuration access to the Nexus switch.

vpc_pool

Type string

Default <None>

This is port-channel/VPC allocation pool of ids used with baremetal deployments only. When there is a list of ethernet interfaces provided by Ironic to neutron in a port event, these are assumed to be a port-channel type configuration. Ironic only knows about ethernet interfaces so it is up to the Nexus Driver to either learn the port channel if the user preconfigured the channel-group on the ethernet interfaces; otherwise, the driver will create a new port-channel and apply the channel-group to the ethernet interfaces. This pool is the reserved port-channel IDs available for allocation by the Nexus driver for each switch. The full format for "vpc_pool" is vpc_pool=<start_vpc_no-end_vpc_no> | <vpc_no> {,<start_vpc_no-end_vpc_no> | <vpc_no>}. The "-" in <start_vpc_no,end_vpc_no> allows you to configure a range from start to end and <vpc_no> allows just individual numbers. There can be any number of ranges and numbers separated by commas. There is no default value. If not configured, the port-channel will only handle learned cases and attempts to create port-channels

will fail since there is no id pool available from which to allocate an id. Once defined, it can be redefined by changing "vpc_pool" and restarting neutron. Existing VPC ids in the database are gathered and compared against the new "vpc_pool" config. New configured vpcids not found in the database are added. Inactive entries in the database not found in the new configured vpcids list are removed. An example of this configuration is *vpc_pool=1001-1025,1028*.

Nexus VXLAN Type Driver

The following list of variables are used by the nexus VXLAN type driver. A variable's default value can be changed by adding it to the configuration file included on the neutron-server command line. The Nexus VXLAN type driver is only supported for VM deployments. The sample nexus configuration file is described here, *Sample Nexus VXLAN Type Driver*.

ml2_type_nexus_vxlan

vni_ranges

Type list

Default

Comma-separated list of <vni_min>:<vni_max> tuples enumerating ranges of VXLAN Network IDs that are available for tenant network allocation. Example format: vni_ranges = 100:1000,2000:6000

mcast_ranges

Type list

Default

Multicast groups for the VXLAN interface. When configured, will enable sending all broadcast traffic to this multicast group. Comma separated list of min:max ranges of multicast IP's. NOTE: Must be a valid multicast IP, invalid IP's will be discarded. Example format: mcast_ranges = 224.0.0.1:224.0.0.3, 224.0.1.1:224.0.1.3

UCSM Mechanism Driver

The following list of variables are used by the cisco_ucsm mechanism driver. A variable's default value can be changed by adding it to the configuration file included on the neutron-server command line. The sample ucsm configuration file is described here, *Sample UCSM Mechanism Driver*.

ml2_cisco_ucsm

ucsm_ip

Type string

Default `<None>`

Cisco UCS Manager IP address. This is a required field to communicate with a Cisco UCS Manager.

supported_pci_devs

Type list

Default `1137:0071,8086:10c9`

SR-IOV and VM-FEX vendors to be handled by the driver. xxxx:yyyy represents vendor_id:product_id of the PCI networking devices that the driver needs to handle. It is implicit that the SR-IOV capable devices specified here should be supported on the UCS platform.

ucsm_https_verify

Type boolean

Default `true`

The UCSM driver will always perform SSL certificate checking on the UCS Managers that it is connecting to. This checking can be disabled by setting this global configuration to False. Disabling this check will leave the connection to UCS Manager insecure and vulnerable to man-in-the-middle attacks.

ucsm_username

Type string

Default `<None>`

Username for UCS Manager. This is a required field to communicate with a Cisco UCS Manager.

ucsm_password

Type string

Default `<None>`

Password for UCS Manager. This is a required field to communicate with a Cisco UCS Manager.

ucsm_virtio_eth_ports

Type list

Default `/ether-eth0,/ether-eth1`

Ethernet port names to be used for virtio ports. This config lets the Cloud Admin specify what ports on the UCS Servers can be used for OpenStack virtual port configuration. The names should match the names on the UCS Manager.

ucsm_host_list

Type dict

Default `<None>`

Hostname to Service profile mapping for UCS Manager controlled hosts. This Service profile should not be associated with a Service Profile Template. If the Service Profile is not specified with a path, the driver assumes that it is at the root level on the UCSM. For example: Hostname1:Serviceprofile1, Hostname2:Serviceprofile2

sriov_qos_policy

Type string

Default `${ml2_cisco_ucsm.sriov_qos_policy}`

A pre-defined QoS policy name. This optional config allows the cloud admin to pre-create a QoS policy on the UCSM. If this config is present, the UCSM driver will associate this QoS policy with every Port profile it creates for SR-IOV ports.

sp_template_list

Type unknown type

Default `<None>`

Service Profile Template config for this UCSM. The configuration to be provided should be a list where each element in the list represents information for a single Service Profile Template on that UCSM. Each element is mapping of a Service Profile Template's path, its name and a list of all UCS Servers controlled by this template. For example: sp_template_list = SP_Template1_path:SP_Template1:Host1,Host2 SP_Template2_path:SP_Template2:Host3,Host4 This is an optional config with no defaults

vnic_template_list

Type unknown type

Default `<None>`

VNIC Profile Template config per UCSM. Allows the cloud admin to specify a VNIC Template on the UCSM that is attached to every vNIC connected to a specific physical network. Each element in this list has 3 parts: the physical network that is defined in neutron configuration, the VNIC Template with its path in UCSM, the vNIC on the UCS Servers that is connected to this physical network. For example: vnic_template_list = physnet1:vnic_template_path1:vt1 physnet2:vnic_template_path2:vt2 This is an optional config with no defaults.

ml2_cisco_ucsm_ip:<ip_address>

ucsm_username

Type string

Default `<None>`

Username for UCS Manager. This is a required field to communicate with a Cisco UCS Manager.

ucsm_password

Type string

Default `<None>`

Password for UCS Manager. This is a required field to communicate with a Cisco UCS Manager.

ucsm_virtio_eth_ports

Type list

Default `/ether-eth0,/ether-eth1`

Ethernet port names to be used for virtio ports. This config lets the Cloud Admin specify what ports on the UCS Servers can be used for OpenStack virtual port configuration. The names should match the names on the UCS Manager.

ucsm_host_list

Type dict

Default `<None>`

Hostname to Service profile mapping for UCS Manager controlled hosts. This Service profile should not be associated with a Service Profile Template. If the Service Profile is not specified with a path, the driver assumes that it is at the root level on the UCSM. For example: Hostname1:Serviceprofile1, Hostname2:Serviceprofile2

sriov_qos_policy

Type string

Default `${ml2_cisco_ucsm.sriov_qos_policy}`

A pre-defined QoS policy name. This optional config allows the cloud admin to pre-create a QoS policy on the UCSM. If this config is present, the UCSM driver will associate this QoS policy with every Port profile it creates for SR-IOV ports.

sp_template_list

Type unknown type

Default `<None>`

Service Profile Template config for this UCSM. The configuration to be provided should be a list where each element in the list represents information for a single Service Profile Template on that UCSM. Each element is mapping of a Service Profile Template's path, its name and a list of all UCS Servers controlled by this template. For example: sp_template_list = SP_Template1_path:SP_Template1:Host1,Host2 SP_Template2_path:SP_Template2:Host3,Host4 This is an optional config with no defaults

vnic_template_list

Type unknown type

Default `<None>`

VNIC Profile Template config per UCSM. Allows the cloud admin to specify a VNIC Template on the UCSM that is attached to every vNIC connected to a specific physical network. Each element in this list has 3 parts: the physical network that is defined in neutron configuration, the VNIC Template with its path in UCSM, the vNIC on the UCS Servers that is connected to this physical network. For example: vnic_template_list = physnet1:vnic_template_path1:vt1 physnet2:vnic_template_path2:vt2 This is an optional config with no defaults.

sriov_multivlan_trunk

network_vlans

Type unknown type

Default `<None>`

SR-IOV Multi-VLAN trunk config section is an optional config section to accomodate the scenario where an application using an SR-IOV port to communicate would like to send traffic on multiple application specific VLANs not known to OpenStack. This config section is applicable across all UCSMs specified as part of the OpenStack cloud. The names of the neutron networks on which the SR-IOV ports are going to be created have to be known ahead of time and should be associated with a list or range of application VLANs using the following format: <neutron network name>=<comma separated list of VLAN-ids or VLAN-id ranges> For example: test_network1=5,7-9

3.3.2 Sample Configuration Files

Sample ASR1000 L3 Router Service Plugin

L3 Router Service Plugin

The sample configuration below for the L3 router service plugin can also be viewed as raw text `cisco_router_plugin.ini` file.

```
[routing]
# Name of default router type to create. Must be a unique name.
default_router_type = ASR1k_router
```

```
# Name of router type for Linux network namespace-based routers
# namespace_router_type_name = NetworkNamespace_router

# Time in seconds between renewed scheduling attempts of non-scheduled routers
# backlog_processing_interval = 10

# Driver to use for routertype-aware scheduling of router to a default L3 agent
# router_type_aware_scheduler_driver = networking_cisco.plugins.cisco.l3.schedulers.
↪l3_routertype_aware_agent_scheduler.L3RouterTypeAwareScheduler

# Set 'auto_schedule' to True if routers are to be automatically scheduled by default
# auto_schedule = True

# Set 'share_hosting_device' to True if routers can share hosts with routers owned by␣
↪other tenants by default
# share_hosting_device = True


[router_types]
# Cisco router type definitions.
# In addition to defining router types using the neutron client,
# router types can be defined here to be immediately available
# when Neutron is started.
# NOTE! All fields must be included (even if left empty).

# Cisco router type format.
# [cisco_router_type:<UUID of router type>]
# name=<router type name, should preferably be unique>
# description=<description of router type>
# template_id=<template to use to create hosting devices for this router type>
# ha_enabled_by_default=<True if HA should be enabled by default>
# shared=<True if if routertype is available to all tenants, False otherwise>
# slot_need=<Number of slots this router type consume in hosting device>
# scheduler=<module to be used as scheduler for router of this type>  (1)
# driver=<module to be used by router plugin as router type driver>    (2)
# cfg_agent_service_helper=<module to be used by configuration agent
#                           as service helper driver                  (3)
# cfg_agent_driver=<module to be used by configuration agent for
#                   device configurations>                            (4)

# (1) --(4): Leave empty for routers implemented in network nodes

# Example:
# [cisco_router_type:1]
# name=Namespace_Neutron_router
# description="Neutron router implemented in Linux network namespace"
# template_id=1
# shared=True
# slot_need=0
# scheduler=
# driver=
# cfg_agent_service_helper=
# cfg_agent_driver=

# [cisco_router_type:2]
# name=Hardware_Neutron_router
# description="Neutron router implemented in Cisco ASR1k device"
# template_id=2
```

```
# ha_enabled_by_default=True
# shared=True
# slot_need=1
# scheduler=networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_
↪scheduler.L3RouterHostingDeviceHARandomScheduler
# driver=networking_cisco.plugins.cisco.l3.drivers.asr1k.asr1k_routertype_driver.
↪ASR1kL3RouterDriver
# cfg_agent_service_helper=networking_cisco.plugins.cisco.cfg_agent.service_helpers.
↪routing_svc_helper.RoutingServiceHelper
# cfg_agent_driver=networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.
↪asr1k_routing_driver.ASR1kRoutingDriver

[ha]
# Enables high-availability support for routing service
# ha_support_enabled = True

# Default number of routers added for redundancy when high-availability
# by VRRP, HSRP, or GLBP is used (maximum is 4)
# default_ha_redundancy_level = 1

# Default mechanism used to implement high-availability. Can be one of HSRP,
# VRRP, or GLBP
# default_ha_mechanism = HSRP

# List of administratively disabled high-availability mechanisms (one or
# several of VRRP, HSRP, GBLP)
# disabled_ha_mechanisms = []

# Enables connectivity probing for high-availability even if (admin) user does
# not explicitly request it
# connectivity_probing_enabled_by_default = False

# Host that will be probe target for high-availability connectivity probing
# if (admin) user does not specify it
# default_probe_target = None

# Time (in seconds) between probes for high-availability connectivity probing
# if user does not specify it
# default_ping_interval = 5
```

Device Manager Plugin

The sample configuration below for the device manager plugin can also be viewed as raw text `cisco_device_manager_plugin.ini` file.

```
[general]
# Name of the L3 admin tenant
# l3_admin_tenant = L3AdminTenant

# Name of management network for hosting device configuration
# management_network = osn_mgmt_nw

# Default security group applied on management port
# default_security_group = mgmt_sec_grp

# Maximal time (in seconds) between checks of config agent status
```

```
# cfg_agent_monitoring_interval = 20

# Seconds of no status update until a cfg agent is considered down
# cfg_agent_down_time = 30

# Driver to use for scheduling hosting device to a Cisco configuration agent
# configuration_agent_scheduler_driver = networking_cisco.plugins.cisco.device_
↪manager.scheduler.hosting_device_cfg_agent_scheduler.HostingDeviceCfgAgentScheduler

# Path to templates for hosting devices
# templates_path = /opt/stack/data/neutron/cisco/templates

# Path to config drive files for service VM instances
# service_vm_config_path = /opt/stack/data/neutron/cisco/config_drive

# Ensure that Nova is running before attempting to create any router VM
# ensure_nova_running = True

# IP address of primary domain name server for hosting devices
# domain_name_server_1 = 8.8.8.8

# IP address of secondary domain name server for hosting devices
# domain_name_server_2 = 8.8.4.4

[hosting_device_credentials]
# Cisco hosting device credentials specifications.
# Credentials for hosting devices must be defined here.
# NOTE! All fields must be included (even if left empty).

# Hosting device credential format.
# [cisco_hosting_device_credential:<UUID of hosting device credential>]  (1)
# name=<name of credential>                                             (2)
# description=<description of credential>                               (3)
# user_name=<username string>
# password=<password string>
# type=<type of credential>                                             (4)

# (1) The UUID can be specified as an integer.
# (2),(3),(4): currently ignored. Can be left empty.

# Example:
# [cisco_hosting_device_credential:1]
# name="Universal credential"
# description="Credential used for all hosting devices"
# user_name=device_administrator
# password=fE#m%%92
# type=

[hosting_devices_templates]
# Cisco hosting device template definitions.
# In addition to defining templates using the neutron client,
# templates can be defined here to be immediately available
# when Neutron is started.
# NOTE! All fields must be included (even if left empty).

# Hosting device template format.
# [cisco_hosting_device_template:<UUID of hosting device template>]   (1)
# name=<name given to hosting devices created using this template>
```

```
# enabled=<template enabled if True>
# host_category=<can be 'VM', 'Hardware', or 'Network_Node'>          (2)
# service_types=<list of service types this template supports>        (3)
# image=<the image name or UUID in Glance>                            (4)
# flavor=<the VM flavor or UUID in Nova>                              (5)
# default_credentials_id=<UUID of default credentials>                (6)
# configuration_mechanism=<indicates how configurations are made>     (7)
# protocol_port=<udp/tcp port of hosting device>
# booting_time=<Typical booting time (in seconds)>
# slot_capacity=<abstract metric specifying capacity to host logical resources>
# desired_slots_free=<desired number of slots to keep available at all times>
# tenant_bound=<list of tenant UUIDs to which template is available>  (8)
# device_driver=<module to be used as hosting device driver>
# plugging_driver=<module to be used as plugging driver >

# (1) The UUID can be specified as an integer.
# (2) Specify 'VM' for virtual machine appliances, 'Hardware' for hardware
#     appliances, and 'Network_Node' for traditional Neutron network nodes.
# (3) Write as string of ':' separated service type names. Can be left empty
#     for now.
# (4) Leave empty for hardware appliances and network nodes.
# (5) Leave empty for hardware appliances and network nodes.
# (6) UUID of credential. Can be specified as an integer.
# (7) Currently ignored. Can be left empty for now.
# (8) A (possibly empty) string of ':'-separated tenant UUIDs representing the
#     only tenants allowed to own/place resources on hosting devices created
#     using this template. If string is empty all tenants are allowed.

# Example:
# [cisco_hosting_device_template:1]
# name=NetworkNode
# enabled=True
# host_category=Network_Node
# service_types=router:FW:VPN
# image=
# flavor=
# default_credentials_id=1
# configuration_mechanism=
# protocol_port=22
# booting_time=360
# slot_capacity=2000
# desired_slots_free=0
# tenant_bound=
# device_driver=networking_cisco.plugins.cisco.device_manager.hosting_device_drivers.
↪noop_hd_driver.NoopHostingDeviceDriver
# plugging_driver=networking_cisco.plugins.cisco.device_manager.plugging_drivers.noop_
↪plugging_driver.NoopPluggingDriver

# [cisco_hosting_device_template:2]
# name="ASR1kv template"
# enabled=True
# host_category=Hardware
# service_types=router:FW:VPN
# image=
# flavor=
# default_credentials_id=1
# configuration_mechanism=
# protocol_port=22
```

```
# booting_time=360
# slot_capacity=2000
# desired_slots_free=0
# tenant_bound=
# device_driver=networking_cisco.plugins.cisco.device_manager.hosting_device_drivers.
↪noop_hd_driver.NoopHostingDeviceDriver
# plugging_driver=networking_cisco.plugins.cisco.device_manager.plugging_drivers.hw_
↪vlan_trunking_driver.HwVLANTrunkingPlugDriver

[hosting_devices]
# Cisco hosting device specifications.
# In addition to specifying hosting devices using the neutron client,
# devices can be specified here to be immediately available when Neutron is
# started.
# NOTE! All fields must be included (even if left empty).

# Hosting device format.
# [cisco_hosting_device:<UUID of hosting device>]                          (1)
# template_id=<UUID of hosting device template for this hosting device>
# credentials_id=<UUID of credentials for this hosting device>
# name=<name of device, e.g., its name in DNS>
# description=<arbitrary description of the device>
# device_id=<manufacturer id of the device, e.g., its serial number>
# admin_state_up=<True if device is active, False otherwise>
# management_ip_address=<IP address of device's management network interface>
# protocol_port=<udp/tcp port of hosting device's management process>
# tenant_bound=<Tenant UUID or empty string>                              (2)
# auto_delete=<True or False>                                             (3)

# (1) The UUID can be specified as an integer.
# (2) UUID of the only tenant allowed to own/place resources on this hosting
#     device. If empty any tenant can place resources on it.
# (3) If True, a VM-based hosting device is subject to deletion as part of
#     hosting device pool management and in case of VM failures. If set to
#     False, the hosting device must be manually unregistered in the device
#     manager and any corresponding VM must be deleted in Nova.

# Example:
# [cisco_hosting_device:2]
# template_id=1
# credentials_id=1
# name=dragon
# description=Main ASR1k serving region 1
# device_id=SN:abcd1234efgh
# admin_state_up=True
# management_ip_address=10.0.100.5
# protocol_port=22
# tenant_bound=
# auto_delete=False

[plugging_drivers]
# Cisco plugging driver configurations.
# Plugging driver specific settings are made here.

# For the hw_vlan_trunking_driver.HwVLANTrunkingPlugDriver plugging driver
# it is expected that for each hosting device the network interfaces to be used
# to reach different Neutron networks are specified.
```

```
# Specifically the format for this plugging driver is as follows
# [HwVLANTrunkingPlugDriver:<UUID of hosting device>]                    (1)
# internal_net_interface_<int number>=<network_uuid_spec>:<interface_name> (2)
# external_net_interface_<int number>=<network_uuid_spec>:<interface_name> (3)
# [zero or more additional internal or external specifications] ...

# (1) The UUID can be specified as an integer.
# (2),(3) <network_uuid_spec> can be '*' or a UUID, or a comma separated list
#         of UUIDs.

# Example:
# [HwVLANTrunkingPlugDriver:3]
# internal_net_interface_1=*:GigabitEthernet1
# external_net_interface_1=*:GigabitEthernet2

# [HwVLANTrunkingPlugDriver:4]
# internal_net_interface_1=*:GigabitEthernet1
# external_net_interface_1=d7b2eac2-1ade-444e-edc5-81fd4267f53a:GigabitEthernet2
# external_net_interface_2=a36b533a-fae6-b78c-fe11-34aa82b12e3a,45c624b-ebf5-c67b-
↪df22-43bb73c21f4e:GigabitEthernet3
```

Configuration Agent

The sample configurations below for the configuration agent can also be viewed as raw text `cisco_cfg_agent.ini` file.

```
[cfg_agent]
# (IntOpt) Interval in seconds for processing of service updates.
# That is when the config agent's process_services() loop executes
# and it lets each service helper to process its service resources.
# rpc_loop_interval = 10

# (BoolOpt)  If enabled, the agent will maintain a heartbeat against
# its hosting-devices. If a device dies and recovers, the agent will
# then trigger a configuration resync.
# enable_heartbeat = True

# (IntOpt)  Interval in seconds when the config agent runs the
# backlog / hosting-device heart beat task.
# heartbeat_interval = 5

# (IntOpt)  Maximum number of attempts for a device sync.
# max_device_sync_attempts = 6

# (IntOpt) The largest number of routers to fetch in one RPC call.
# max_device_sync_batch_size = 64

# (StrOpt) Period-separated module path to the routing service helper class.
# routing_svc_helper_class = networking_cisco.plugins.cisco.cfg_agent.service_helpers.
↪routing_svc_helper.RoutingServiceHelper

# (IntOpt) Timeout value in seconds for connecting to a hosting device.
# device_connection_timeout = 30

# (IntOpt) The time in seconds until a backlogged hosting device is
# presumed dead or booted to an error state.
```

```
# hosting_device_dead_timeout = 300

# (IntOpt) Interval in seconds when the config agent sents a report to the
# plugin. This is used to keep tab on the liveliness of the cfg agent.
# This value should be more than 0, otherwise cfg agent will be considered
# as dead.
# keepalive_interval = 10

# (IntOpt)The iteration where the config agent sends a full status report to
# the plugin. The default is every 6th iteration of the keep alive interval.
# This means with default value of keepalive_interval (10sec), a full report
# is sent once every 6*10 = 60 seconds.
# report_iteration = 6
```

Sample Nexus Mechanism Driver

This sample configuration file can also be viewed in file form.

```
[DEFAULT]


[ml2_cisco]

#
# From networking_cisco.nexus
#

# When "managed_physical_network" is configured, it restricts the network
# segment that the nexus driver supports. Setting it to a specific network name
# will limit the actions taken by this driver to only that network. The network
# name must match a name defined in the "network_vlan_ranges" configuration.
# When "managed_physical_network" is not set, events for all network segments
# will be processed by the driver. (string value)
#managed_physical_network = <None>

# A flag indicating whether the Nexus driver should manage the creation and
# removal of VLANs for provider networks on the Nexus switches. When this flag
# is False, the Nexus driver will not create or remove VLANs for provider
# networks and the administrator needs to manage these interfaces manually or
# by external orchestration. (boolean value)
#provider_vlan_auto_create = true

# A flag indicating whether Nexus driver should manage the adding and removing
# of provider VLANs from trunk ports on the Nexus switches. When this flag is
# False, the Nexus driver will not add or remove provider VLANs from trunk
# ports and the administrator needs to manage these operations manually or by
# external orchestration. (boolean value)
#provider_vlan_auto_trunk = true

# Configuration replay is enabled by default by defining the time interval to
# 30 seconds.  This is the amount of time to check the state of all known Nexus
# device(s). To disable the replay feature, set this "switch_heartbeat_time" to
# 0 seconds. (integer value)
#switch_heartbeat_time = 30

# A flag indicating whether the Nexus driver should manage the creating and
# removing of the Nexus switch VXLAN global settings of "feature nv overlay",
```

```
# "feature vn-segment-vlan-based", "interface nve 1" and the NVE subcommand
# "source-interface loopback #". When set to the default of False, the Nexus
# driver will not add or remove these VXLAN settings and the administrator
# needs to manage these operations manually or by external orchestration.
# (boolean value)
#vxlan_global_config = false


[ml2_mech_cisco_nexus:<ip_address>]

#
# From networking_cisco.nexus
#

# This configuration option defaults to True. When https_verify is True, the
# directory for certification authority (CA) certificates or self-signed
# certificate file must be locally defined. This location is configured in
# https_local_certificate. To skip https certification checking, set
# https_verify to False though this is strongly discouraged since it makes the
# connection insecure. This configuration is set for each Nexus switch.
# (boolean value)
#https_verify = true

# Configure either a directory to Certificate Authority certificates or a self-
# signed certificate file to be used with https requests. Self-signed
# certificates should be used temporarily when an official certificate from a
# trusted Certificate Authority is not yet available. In this case, the Nexus
# switch must also be configured and enabled with both the certificate and key
# files.  For further details, refer to the "nxapi certificate" commands as
# defined in the "Nexus 9K NXAPI Programmability Guide". If the default
# configuration is None and if https_verify is True, the underlying SSL package
# will attempt to locate CA bundles using environment variables or DISTRO
# default locations. It is recommended that the user explicitly configure this
# value to avoid trusting a random list of CAs. An example configuration for
# certificate authorities would look like
# https_local_certificate=/path/to/ca_certificates. An example configuration of
# a self-signed certificate would look like
# https_local_certificate=/path/to/cafile.crt. This configuration is set for
# each Nexus switch. (string value)
#https_local_certificate = <None>

# intfcfg_portchannel is a list of Nexus port-channel config CLI used when
# baremetal port-channels are created by the Nexus driver.  It is dependent on
# "vpc_pool" being configured.  Any number of Nexus port-channel commands
# separated by ";" can be provided.  When there are multiple interfaces in a
# baremetal event, the nexus driver checks to determine whether a port-channel
# is already applied to the interfaces; otherwise, it creates a port channel.
# This optional configuration allows the administrator to custom configure the
# port-channel.  When not configured, the nexus driver defaults to configuring
# "spanning-tree port type edge trunk;no lacp suspend-individual" beneath the
# port-channel. An example of this configuration is "intfcfg_portchannel=no
# lacp suspend-individual;spanning-tree port type edge trunk". (string value)
#intfcfg_portchannel = <None>

# Only valid if VXLAN overlay is configured and vxlan_global_config is set to
# True. The NVE source interface is a loopback interface that is configured on
# the switch with valid /32 IP address. This /32 IP address must be known by
# the transient devices in the transport network and the remote VTEPs.  This is
```

```
# accomplished by advertising it through a dynamic routing protocol in the
# transport network. If nve_src_intf is not defined, a default setting of 0 is
# used to create "loopback0".  This is configured for non-baremetal only.
# (string value)
#nve_src_intf = <None>

# The password of the Nexus Switch Administrator is required to allow
# configuration access to the Nexus switch. (string value)
#password = <None>

# This is required if Nexus VXLAN overlay feature is configured.  It should be
# the physical network name defined in "network_vlan_ranges" (defined beneath
# the "ml2_type_vlan" section) that this switch is controlling.  The configured
# "physnet" is the physical network domain that is connected to this switch.
# The vlan ranges defined in "network_vlan_ranges" for a physical network are
# allocated dynamically and are unique per physical network. These dynamic
# vlans may be reused across physical networks.  This configuration applies to
# non-baremetal only. (string value)
#physnet = <None>

# A list of key:value pairs describing which host is connected to which
# physical port or portchannel on the Nexus switch. The format should look
# like:
# host_ports_mapping=<your-hostname>:[<intf_type><port>,<intf_type><port>],
#                    <your-second-host>:[<intf_type><port>]
# For example:
# host_ports_mapping=host-1:[ethernet1/1, ethernet1/2],
#                    host-2:[ethernet1/3],
#                    host-3:[port-channel20]
# Lines can be broken with indentation to ensure config files remain readable.
# All compute nodes must be configured while controllers are optional depending
# on your network configuration. Depending on the configuration of the host,
# the hostname is expected to be the full hostname (hostname.domainname) which
# can be derived by running "hostname -f" on the host itself. Valid intf_types
# are "ethernet" or "port-channel".  The default setting for <intf_type> is
# "ethernet" and need not be added to this setting. This configuration applies
# to VM deployments only. (dict value)
#host_ports_mapping = <None>

# The username of the Nexus Switch Administrator is required to allow
# configuration access to the Nexus switch. (string value)
#username = <None>

# This is port-channel/VPC allocation pool of ids used with baremetal
# deployments only.  When there is a list of ethernet interfaces provided by
# Ironic to neutron in a port event, these are assumed to be a port-channel
# type configuration.  Ironic only knows about ethernet interfaces so it is up
# to the Nexus Driver to either learn the port channel if the user
# preconfigured the channel-group on the ethernet interfaces; otherwise, the
# driver will create a new port-channel and apply the channel-group to the
# ethernet interfaces.  This pool is the reserved port-channel IDs available
# for allocation by the Nexus driver for each switch. The full format for
# "vpc_pool" is vpc_pool=<start_vpc_no-end_vpc_no> | <vpc_no> {,<start_vpc_no-
# end_vpc_no> | <vpc_no>}. The "-" in <start_vpc_no,end_vpc_no> allows you to
# configure a range from start to end and <vpc_no> allows just individual
# numbers.  There can be any number of ranges and numbers separated by commas.
# There is no default value.  If not configured, the port-channel will only
# handle learned cases and attempts to create port-channels will fail since
```

```
# there is no id pool available from which to allocate an id. Once defined, it
# can be redefined by changing "vpc_pool" and restarting neutron. Existing VPC
# ids in the database are gathered and compared against the new "vpc_pool"
# config.  New configured vpcids not found in the database are added.  Inactive
# entries in the database not found in the new configured vpcids list are
# removed. An example of this configuration is `vpc_pool=1001-1025,1028`.
# (string value)
#vpc_pool = <None>
```

Sample Nexus VXLAN Type Driver

This sample configuration file can also be viewed in file form.

```
[DEFAULT]


[ml2_type_nexus_vxlan]

#
# From networking_cisco.nexus_vxlan_type_driver
#

# Comma-separated list of <vni_min>:<vni_max> tuples enumerating ranges of
# VXLAN Network IDs that are available for tenant network allocation. Example
# format: vni_ranges = 100:1000,2000:6000 (list value)
#vni_ranges =

# Multicast groups for the VXLAN interface. When configured, will enable
# sending all broadcast traffic to this multicast group. Comma separated list
# of min:max ranges of multicast IP's. NOTE: Must be a valid multicast IP,
# invalid IP's will be discarded. Example format: mcast_ranges =
# 224.0.0.1:224.0.0.3, 224.0.1.1:224.0.1.3 (list value)
#mcast_ranges =
```

Sample UCSM Mechanism Driver

This sample configuration file can also be viewed in raw text `ml2_ucsm.ini.sample`.

```
[DEFAULT]


[ml2_cisco_ucsm]

#
# From networking_cisco.ucsm
#

# Cisco UCS Manager IP address. This is a required field to communicate with a
# Cisco UCS Manager. (string value)
#ucsm_ip = <None>

# SR-IOV and VM-FEX vendors to be handled by the driver. xxxx:yyyy represents
# vendor_id:product_id of the PCI networking devices that the driver needs to
# handle. It is implicit that the SR-IOV capable devices specified here should
# be supported on the UCS platform. (list value)
```

```
#supported_pci_devs = 1137:0071,8086:10c9

# The UCSM driver will always perform SSL certificate checking on the UCS
# Managers that it is connecting to. This checking can be disabled by setting
# this global configuration to False. Disabling this check will leave the
# connection to UCS Manager insecure and vulnerable to man-in-the-middle
# attacks. (boolean value)
#ucsm_https_verify = true

# Username for UCS Manager. This is a required field to communicate with a
# Cisco UCS Manager. (string value)
#ucsm_username = <None>

# Password for UCS Manager. This is a required field to communicate with a
# Cisco UCS Manager. (string value)
#ucsm_password = <None>

# Ethernet port names to be used for virtio ports. This config lets the Cloud
# Admin specify what ports on the UCS Servers can be used for OpenStack virtual
# port configuration. The names should match the names on the UCS Manager.
# (list value)
#ucsm_virtio_eth_ports = /ether-eth0,/ether-eth1

# Hostname to Service profile mapping for UCS Manager controlled hosts. This
# Service profile should not be associated with a Service Profile Template. If
# the Service Profile is not specified with a path, the driver assumes that it
# is at the root level on the UCSM. For example: Hostname1:Serviceprofile1,
# Hostname2:Serviceprofile2 (dict value)
#ucsm_host_list = <None>

# A pre-defined QoS policy name. This optional config allows the cloud admin to
# pre-create a QoS policy on the UCSM. If this config is present, the UCSM
# driver will associate this QoS policy with every Port profile it creates for
# SR-IOV ports. (string value)
#sriov_qos_policy = ${ml2_cisco_ucsm.sriov_qos_policy}

# Service Profile Template config for this UCSM. The configuration to be
# provided should be a list where each element in the list represents
# information for a single Service Profile Template on that UCSM. Each element
# is mapping of a Service Profile Template's path, its name and a list of all
# UCS Servers controlled by this template. For example:
# sp_template_list = SP_Template1_path:SP_Template1:Host1,Host2
#                    SP_Template2_path:SP_Template2:Host3,Host4
# This is an optional config with no defaults (SPTemplateList)
#sp_template_list = <None>

# VNIC Profile Template config per UCSM. Allows the cloud admin to specify a
# VNIC Template on the UCSM that is attached to every vNIC connected to a
# specific physical network. Each element in this list has 3 parts: the
# physical network that is defined in neutron configuration, the VNIC Template
# with its path in UCSM, the vNIC on the UCS Servers that is connected to this
# physical network. For example:
# vnic_template_list = physnet1:vnic_template_path1:vt1
#                      physnet2:vnic_template_path2:vt2
# This is an optional config with no defaults. (VNICTemplateList)
#vnic_template_list = <None>
```

```
[ml2_cisco_ucsm_ip:<ip_address>]

#
# From networking_cisco.ucsm
#

# Username for UCS Manager. This is a required field to communicate with a
# Cisco UCS Manager. (string value)
#ucsm_username = <None>

# Password for UCS Manager. This is a required field to communicate with a
# Cisco UCS Manager. (string value)
#ucsm_password = <None>

# Ethernet port names to be used for virtio ports. This config lets the Cloud
# Admin specify what ports on the UCS Servers can be used for OpenStack virtual
# port configuration. The names should match the names on the UCS Manager.
# (list value)
#ucsm_virtio_eth_ports = /ether-eth0,/ether-eth1

# Hostname to Service profile mapping for UCS Manager controlled hosts. This
# Service profile should not be associated with a Service Profile Template. If
# the Service Profile is not specified with a path, the driver assumes that it
# is at the root level on the UCSM. For example: Hostname1:Serviceprofile1,
# Hostname2:Serviceprofile2 (dict value)
#ucsm_host_list = <None>

# A pre-defined QoS policy name. This optional config allows the cloud admin to
# pre-create a QoS policy on the UCSM. If this config is present, the UCSM
# driver will associate this QoS policy with every Port profile it creates for
# SR-IOV ports. (string value)
#sriov_qos_policy = ${ml2_cisco_ucsm.sriov_qos_policy}

# Service Profile Template config for this UCSM. The configuration to be
# provided should be a list where each element in the list represents
# information for a single Service Profile Template on that UCSM. Each element
# is mapping of a Service Profile Template's path, its name and a list of all
# UCS Servers controlled by this template. For example:
# sp_template_list = SP_Template1_path:SP_Template1:Host1,Host2
#                    SP_Template2_path:SP_Template2:Host3,Host4
# This is an optional config with no defaults (SPTemplateList)
#sp_template_list = <None>

# VNIC Profile Template config per UCSM. Allows the cloud admin to specify a
# VNIC Template on the UCSM that is attached to every vNIC connected to a
# specific physical network. Each element in this list has 3 parts: the
# physical network that is defined in neutron configuration, the VNIC Template
# with its path in UCSM, the vNIC on the UCS Servers that is connected to this
# physical network. For example:
# vnic_template_list = physnet1:vnic_template_path1:vt1
#                      physnet2:vnic_template_path2:vt2
# This is an optional config with no defaults. (VNICTemplateList)
#vnic_template_list = <None>


[sriov_multivlan_trunk]

#
```

```
# From networking_cisco.ucsm
#

# SR-IOV Multi-VLAN trunk config section is an optional config section to
# accomodate the scenario where an application using an SR-IOV port to
# communicate would like to send traffic on multiple application specific VLANs
# not known to OpenStack. This config section is applicable across all UCSMs
# specified as part of the OpenStack cloud. The names of the neutron networks
# on which the SR-IOV ports are going to be created have to be known ahead of
# time and should be associated with a list or range of application VLANs using
# the following format:
# <neutron network name>=<comma separated list of VLAN-ids or VLAN-id ranges>
# For example:
# test_network1=5,7-9 (dict value)
#network_vlans = <None>
```

3.4 Contributor Guides

3.4.1 How to Contribute

If you would like to contribute to the development of networking-cisco, you must follow the steps as outlined by the OpenStack page:

> https://docs.openstack.org/infra/manual/developers.html

Once those steps have been completed, changes to networking-cisco 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, not GitHub:

> https://bugs.launchpad.net/networking-cisco

Tox environments provided in networking-cisco:

- py27, py34 - Unit tests run against Mitaka neutron, on different python2.7 and python3.4
- newton - Unit tests run against Newton neutron with python2.7
- master - Unit tests run against master neutron with python2.7
- coverage - provides a report on the test coverage
- compare-coverage - compares coverage reports from before and after the current changes
- pep8 - Checks code against the pep8 and OpenStack hacking rules
- bandit - Performs static analysis on selected python source code
- genconfig - Generate sample configuration files included in the documentation
- docs - Generates documentation for viewing (hint: Run *genconfig* first)

DevStack is used by developers to install OpenStack and is not intended for production use. To get details on using DevStack, refer to other documentation links such as:

- For general DevStack information, refer to DevStack
- For general ML2 DevStack details, refer to ML2_DevStack

As discussed in these links, `local.conf` is DevStack's configuration file for defining OpenStack installations. To include installing the networking-cisco repository, add the following configuration.

```
[[local|localrc]]
enable_plugin networking-cisco https://github.com/openstack/networking-cisco
```

For further Cisco feature configuration details using DevStack, look for other plugin/driver subsections in the Cisco contributor guide for sample DevStack configurations.

3.4.2 Contributor Frequently Asked Questions

How do I...

...know if a release note is needed for my change?

Reno documentation contains a description of what can be added to each section of a release note. If, after reading this, you're still unsure about whether to add a release note for your change or not, keep in mind that it is intended to contain information for deployers, so changes to unit tests or documentation are unlikely to require one.

...create a new release note?

By running `reno` command via tox, e.g:

```
$ tox -e venv -- reno new brief-description-cool-new-release-note
  venv create: /home/foo/networking-cisco/.tox/venv
  venv installdeps: -r/home/foo/networking-cisco/test-requirements.txt
  venv develop-inst: /home/foo/networking-cisco
  venv runtests: PYTHONHASHSEED='0'
  venv runtests: commands[0] | reno new brief-description-cool-new-release-note
  Created new notes file in releasenotes/notes/brief-description-cool-new-release-
↪note-ecb3875dc1cbf6d9.yaml
    venv: commands succeeded
    congratulations :)

$ git status
  On branch test
  Untracked files:
    (use "git add <file>..." to include in what will be committed)

    releasenotes/notes/brief-description-cool-new-release-note-ecb3875dc1cbf6d9.yaml
```

Then edit the result file. Note that:

- we prefer to use present tense in release notes. For example, a release note should say "Adds support for feature foo", not "Added support for feature foo". (We use 'adds' instead of 'add' because grammatically, it is "ironic adds support", not "ironic add support".)
- any variant of English spelling (American, British, Canadian, Australian...) is acceptable. The release note itself should be consistent and not have different spelling variants of the same word.
- Release notes for different plugins should be identified by writing each release note in the following format:

```
plugin_id: A short summary of the note

Full release note text explaining the impact of the change.
```

```
https://link-to-launchpad.com/bug-or-rfe-if-there-is-one
```

To check your release notes you can build the documentation locally by running the command:

```
$ tox -e docs
  Documentation should build out successfully, or errors about the formatting
  should appear.
```

If successful then you can open your browser and view your release notes rendered out with the rest by going to:

```
file:///<your workspace>/networking-cisco/doc/build/html/reference/releasenotes/index.
↪html
```

For more information see the reno documentation.

3.4.3 ASR1000 L3 Router Service Plugin Contributor Guide

Using DevStack

DevStack is used by developers to install Openstack. It is not intended for production use. For introductory details on DevStack, refer to *How to Contribute*.

To install the ASR1k L3 router service plugin along with OpenStack using DevStack, do as follows:

1. Clone DevStack and checkout the branch (ex: stable/ocata, stable/newton, etc) you want to install.
2. Configure the ASR1k L3 router service plugin in `local.conf` file as shown in examples which follow.
3. Run **`./stack.sh`** to install and **`./unstack.sh`** to uninstall.

DevStack Configuration Examples

This section describes how to extend the `local.conf` file with ASR1k-based L3 routing details for DevStack deployment. These details should follow the section which installs networking-cisco repository as described in *How to Contribute*.

Append the following lines to `local.conf` to enable the L3P, DMP and CFGA:

```
Q_CISCO_ASR1K_ENABLED=True

enable_service ciscocfgagent
enable_service q-ciscorouter
enable_service q-ciscodevicemanager

[[post-config|/etc/neutron/neutron.conf]]

[DEFAULT]
api_extensions_path = extensions:/opt/stack/networking-cisco/networking_cisco/plugins/
↪cisco/extensions
```

Defining credentials, hosting device templates, hosting devices and router types

DevStack can automatically include definitions of credentials, hosting device templates, hosting devices and router types in configuration files that are given as arguments to neutron server and the CFGA when they are started.

The actual definitions to be included has to be provided to DevStack. This is done using two text files:

- `cisco_device_manager_plugin.inject`
- `cisco_router_plugin.inject`

If these files exist in the DevStack root directory when the **`./stack.sh`** command is executed, DevStack will append their contents to configuration files that neutron server consumes when it starts.

A cisco_device_manager_plugin.inject sample file

The sample inject file below can be viewed as raw text `cisco_device_manager_plugin.inject` file.

```
[hosting_device_credentials]
[cisco_hosting_device_credential:1]
name="Universal credential"
description="Credential used for all hosting devices"
user_name=stack
password=cisco
type=

[hosting_devices_templates]
[cisco_hosting_device_template:1]
name=NetworkNode
enabled=True
host_category=Network_Node
service_types=router:FW:VPN
image=
flavor=
default_credentials_id=1
configuration_mechanism=
protocol_port=22
booting_time=360
slot_capacity=2000
desired_slots_free=0
tenant_bound=
device_driver=networking_cisco.plugins.cisco.device_manager.hosting_device_drivers.
 ↪noop_hd_driver.NoopHostingDeviceDriver
plugging_driver=networking_cisco.plugins.cisco.device_manager.plugging_drivers.noop_
 ↪plugging_driver.NoopPluggingDriver

[cisco_hosting_device_template:2]
name="ASR1k template"
enabled=True
host_category=Hardware
service_types=router:FW:VPN
image=
flavor=
default_credentials_id=1
configuration_mechanism=
protocol_port=22
booting_time=360
slot_capacity=4000
desired_slots_free=0
tenant_bound=
device_driver=networking_cisco.plugins.cisco.device_manager.hosting_device_drivers.
 ↪noop_hd_driver.NoopHostingDeviceDriver
plugging_driver=networking_cisco.plugins.cisco.device_manager.plugging_drivers.hw_
 ↪vlan_trunking_driver.HwVLANTrunkingPlugDriver
```

```
[hosting_devices]
[cisco_hosting_device:2]
template_id=2
credentials_id=1
device_id=SN:abcd1234efgh
admin_state_up=True
management_ip_address=10.86.7.54
protocol_port=22
tenant_bound=
auto_delete=False

[cisco_hosting_device:3]
template_id=2
credentials_id=1
device_id=SN:efgh5678ijkl
admin_state_up=True
management_ip_address=10.86.7.55
protocol_port=22
tenant_bound=
auto_delete=False

[plugging_drivers]
[HwVLANTrunkingPlugDriver:2]
internal_net_interface_1=*:GigabitEthernet0/0/0
external_net_interface_1=*:GigabitEthernet0/0/0

[HwVLANTrunkingPlugDriver:3]
internal_net_interface_1=*:GigabitEthernet0/2/1
external_net_interface_1=*:GigabitEthernet0/2/1
```

A `cisco_router_plugin.inject` sample file

The sample inject file below can be viewed as raw text `cisco_router_plugin.inject` file.

```
[router_types]
[cisco_router_type:1]
name=Namespace_Neutron_router
description="Neutron router implemented in Linux network namespace"
template_id=1
ha_enabled_by_default=False
shared=True
slot_need=0
scheduler=
driver=
cfg_agent_service_helper=
cfg_agent_driver=

[cisco_router_type:2]
name=ASR1k_router
description="Neutron router implemented in Cisco ASR1k device"
template_id=2
ha_enabled_by_default=True
shared=True
slot_need=1
scheduler=networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_
↪scheduler.L3RouterHostingDeviceHARandomScheduler
```

```
driver=networking_cisco.plugins.cisco.l3.drivers.asr1k.asr1k_routertype_driver.
↪ASR1kL3RouterDriver
cfg_agent_service_helper=networking_cisco.plugins.cisco.cfg_agent.service_helpers.
↪routing_svc_helper.RoutingServiceHelper
cfg_agent_driver=networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_
↪routing_driver.ASR1kRoutingDriver
```

Source Code Location

Code locations for the ASR1k L3 router service plugin, the device manager plugin and the configuration agent are found in the following directory:

networking-cisco install directory/networking_cisco/plugins/cisco

Typically devstack clone the source code to /opt/stack/networking-cisco.

3.4.4 Nexus Mechanism Driver Contributor Guide

DevStack Configuration Examples

For introductory details on DevStack, refer to *How to Contribute*. This section focuses on Nexus VLAN and VXLAN feature specific changes to DevStack's configuration file local.conf. These changes should follow the section which installs networking-cisco repository as described in *How to Contribute*.

VLAN Configuration

The following sample configuration will provide you with Nexus VLAN Configuration. This configuration supports both normal VM as well as Baremetal. As you can see there is a lot of similarity between the neutron config file and the local.conf file so details in the neutron start-up config file sections *Configuring neutron directly for Nexus* apply here.

```
[[local|localrc]]
enable_plugin networking-cisco https://github.com/openstack/networking-cisco

# Set openstack passwords here.  For example, ADMIN_PASSWORD=ItsASecret

# disable_service/enable_service here. For example,
# disable_service tempest
# enable_service q-svc

# bring in latest code from repo.  (RECLONE=yes; OFFLINE=False)

Q_PLUGIN=ml2
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch,cisco_nexus
Q_ML2_TENANT_NETWORK_TYPE=vlan
ML2_VLAN_RANGES=physnet1:100:109
ENABLE_TENANT_TUNNELS=False
ENABLE_TENANT_VLANS=True
PHYSICAL_NETWORK=physnet1
OVS_PHYSICAL_BRIDGE=br-eth1

[[post-config|/etc/neutron/plugins/ml2/ml2_conf.ini]]
[ml2]
extension_drivers = cisco_providernet_ext
```

```
[ml2_cisco]
switch_heartbeat_time = 30

[ml2_mech_cisco_nexus:192.168.1.1]
host_ports_mapping=ComputeHostA:[1/10]  # deprecates config `ComputeHostA=1/10`
username=admin
password=mySecretPasswordForNexus
vpc_pool=1001-1025,1030
intfcfg_portchannel=no lacp suspend-individual;spanning-tree port type edge trunk

[ml2_mech_cisco_nexus:192.168.2.2]
host_ports_mapping=ComputeHostB:[1/10]  # deprecates config `ComputeHostB=1/10`
username=admin
password=mySecretPasswordForNexus
vpc_pool=1001-1025,1030
intfcfg_portchannel=no lacp suspend-individual;spanning-tree port type edge trunk
```

VXLAN Configuration

In addition to the standard OpenStack settings, follow the `local.conf` file example below to configure the Nexus switch for VXLAN Terminal End Point (VTEP) support. As you can see there is a lot of similarity between the neutron config file and the `local.conf` file so details in the neutron start-up config file sections *VXLAN Overlay Configuration* apply here.

```
[[local|localrc]]
enable_plugin networking-cisco https://github.com/openstack/networking-cisco

Q_PLUGIN=ml2
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch,cisco_nexus
Q_ML2_PLUGIN_TYPE_DRIVERS=nexus_vxlan,vlan
Q_ML2_TENANT_NETWORK_TYPE=nexus_vxlan
ML2_VLAN_RANGES=physnet1:100:109
ENABLE_TENANT_TUNNELS=False
ENABLE_TENANT_VLANS=True
PHYSICAL_NETWORK=physnet1
OVS_PHYSICAL_BRIDGE=br-eth1

[[post-config|/etc/neutron/plugins/ml2/ml2_conf.ini]]
[agent]
minimize_polling=True
tunnel_types=

[ml2]
extension_drivers = cisco_providernet_ext

[ml2_cisco]
switch_hearbeat_time = 30  # No longer required since 30 is now the default in this␣
↪release.

[ml2_mech_cisco_nexus:192.168.1.1]
host_ports_mapping=ComputeHostA:[1/10]  # deprecates config `ComputeHostA=1/10`
username=admin
password=secretPassword
physnet=physnet1
```

```
[ml2_mech_cisco_nexus:192.168.1.2]
host_ports_mapping=ComputeHostB:[1/10]  # deprecates config `ComputeHostB=1/10`
NetworkNode=1/11
username=admin
password=secretPassword
physnet=physnet1

[ml2_type_nexus_vxlan]
vni_ranges=50000:55000
mcast_ranges=225.1.1.1:225.1.1.2

[ml2_type_vlan]
network_vlan_ranges = physnet1:100:109
```

Source Code Location

Code location for the ML2 Nexus Mechanism Driver are found in the following directory:

`{networking-cisco install directory}/networking_cisco/ml2_drivers/nexus`

3.4.5 UCSM Mechanism Driver Contributor Guide

DevStack Configuration Examples

For introductory details on DevStack, refer to *How to Contribute*. This section focuses on how to set the UCSM driver related configuration within DevStack's configuration file `local.conf`. These changes should follow the section which installs networking-cisco repository.

Configuration required for neutron virtual port support

The following parameters need to be provided to DevStack so that the UCSM driver can be initialized with its configuration. The parameters provided to `local.conf` are similar to the configuration options provided to neutron and described in section *UCSM Driver configuration along with neutron parameters*.

Common configuration

The following snippet refers to configuration that is common to all VLAN based mechanism drivers.

```
[[local|localrc]]
enable_plugin networking-cisco https://github.com/openstack/networking-cisco

# Set openstack passwords here.  For example, ADMIN_PASSWORD=ItsASecret

# disable_service/enable_service here. For example,
# disable_service tempest
# enable_service q-svc

# bring in latest code from repo.  (RECLONE=yes; OFFLINE=False)

Q_PLUGIN=ml2
Q_ML2_PLUGIN_MECHANISM_DRIVERS=openvswitch,cisco_ucsm
Q_ML2_TENANT_NETWORK_TYPE=vlan
```

```
ML2_VLAN_RANGES=physnet1:100:109
ENABLE_TENANT_TUNNELS=False
ENABLE_TENANT_VLANS=True
PHYSICAL_NETWORK=physnet1
OVS_PHYSICAL_BRIDGE=br-eth1

Q_PLUGIN_CONF_FILE=/path/to/driver/config/file/ml2_conf.ini

NOVA_CONF=/etc/nova/nova.conf
```

Driver configuration for a single UCSM

When the UCSM driver config needs to be specified in the single UCSM format, the following configuration options need to be specified.

```
[[post-config|/$Q_PLUGIN_CONF_FILE]]
[ml2_cisco_ucsm]

# Single UCSM Config format
ucsm_ip=1.1.1.1
ucsm_username=user
ucsm_password=password

# Hostname to Service profile mapping for UCS Manager
# controlled compute hosts
ucsm_host_list=Hostname1:/Path1/Serviceprofile1, Hostname2:Serviceprofile2

# Service Profile Template config per UCSM. This is a mapping of Service Profile
# Profile Template to the list of UCS Servers (shown as S# below) controlled by
# this template.
sp_template_list = SPT1_path:SPT1:S1,S2 SPT2_path:SPT2:S3,S4

# Ethernet port names to be used for virtio ports
ucsm_virtio_eth_ports = neutron-eth0, neutron-eth1

# vNIC Template config per UCSM. This configuration can be used to specify
# which vNICs are physically connected to a neutron provider network. The
# configuration comprises of a mapping between the neutron provider network,
# the path for vNIC Template and the vNIC Template itself.
vnic_template_list = physnet1:vt_path1:vt11 physnet2:vt_path2:vt21
```

Driver configuration in multi-UCSM format

When the UCSM driver config needs to be specified in the multi-UCSM format, the following configuration options need to be specified.

```
[[post-config|/$Q_PLUGIN_CONF_FILE]]
[ml2_cisco_ucsm]

# If there are multiple UCSMs in the setup, then the below
# config needs to be specified in the multi-UCSM format
# for each UCSM
[ml2_cisco_ucsm_ip:1.1.1.1]
ucsm_username = username
```

```
ucsm_password = password
ucsm_virtio_eth_ports = eth0, eth1
ucsm_host_list=Hostname1:Serviceprofile1, Hostname2:Serviceprofile2
sp_template_list = SPT1_path:SPT1:S1,S2,S3 SPT2_path:SPT2:S4,S5
vnic_template_list = physnet1:vt_path1:vt11 physnet2:vt_path2:vt21
```

Driver configuration to turn off SSL certificate checking

When the UCSM driver is attempting to connect to UCS Manager(s) that do not have a valid SSL certificate, this configuration can be used to simultaneously disable checking of SSL certificates on all UCS Manager(s). However, this is not recommended in production since it leaves the communication path insecure and vulnerable to man-in-the-middle attacks. To setup a valid SSL certificate, use information provided in section *UCSM SSL Certificate Setup*.

```
[[post-config|/$Q_PLUGIN_CONF_FILE]]
[ml2_cisco_ucsm]

ucsm_https_verify = False
```

SR-IOV specific configuration

1. On the controller nodes, update the list of available scheduler filters to include the `PciPassthroughFilter`.

   ```
   [[post-config|/$NOVA_CONF]]
   [DEFAULT]
   scheduler_default_filters = RetryFilter, AvailabilityZoneFilter, RamFilter,␣
   ↪ComputeFilter, ComputeCapabilitiesFilter, ImagePropertiesFilter,␣
   ↪ServerGroupAffinityFilter, PciPassthroughFilter
   ```

2. On each of the compute nodes, additional configuration should be specified to allow a list of PCI devices. This whitelist is consumed by nova-compute to determine which PCI devices can used as SR-IOV devices. The following snippet shows how this configuration can be specified within the `local.conf` files of compute nodes. The vendor and product IDs for Cisco VICs are `1137` and `0071` respectively.

   ```
   [[post-config|/$NOVA_CONF]]
   [DEFAULT]
   pci_passthrough_whitelist = {"vendor_id":"1111","product_id":"aaaa","physical_
   ↪network":"physnet1"}
   ```

3. To specify the list of PCI devices that need to be configured by the UCSM driver, use the following configuration options. The UCSM driver supports SR-IOV configuration on Cisco VICs and Intel NICs by default. This parameter can be omitted if the SR-IOV NICs to be supported are one of the defaults. In the multi-UCSM format this configuration needs to be specified per UCSM.

   ```
   # SR-IOV and VM-FEX vendors supported by this driver
   # xxxx:yyyy represents vendor_id:product_id
   # This config is optional.
   supported_pci_devs=['2222:3333', '4444:5555']
   ```

4. The configuration option to specify the list of application specific VLANs per physical network carrying SR-IOV traffic is as follows.

   ```
   # SR-IOV Multi-VLAN trunk config section
   [sriov_multivlan_trunk]
   ```

```
test_network1=5,7-9
test_network2=500,701 - 709
```

3.4.6 Python Module Docs

networking_cisco

networking_cisco package

Subpackages

networking_cisco.apps package

Subpackages

networking_cisco.apps.saf package

Subpackages

networking_cisco.apps.saf.agent package

Subpackages

networking_cisco.apps.saf.agent.topo_disc package

Submodules

networking_cisco.apps.saf.agent.topo_disc.pub_lldp_api module

This file contains the public API's for interacting with LLDPAD.

class networking_cisco.apps.saf.agent.topo_disc.pub_lldp_api.**LldpApi**(*root_helper*)

Bases: object

LLDP API Class.

enable_lldp(*port_name*, *is_ncb=True*, *is_nb=False*)

Function to enable LLDP on the interface.

get_lldp_tlv(*port_name*, *is_ncb=True*, *is_nb=False*)

Function to Query LLDP TLV on the interface.

get_remote_chassis_id_mac(*tlv_data*)

Returns Remote Chassis ID MAC from the TLV.

get_remote_evb_cfgd(*tlv_data*)

Returns IF EVB TLV is present in the TLV.

get_remote_evb_mode(*tlv_data*)

Returns the EVB mode in the TLV.

get_remote_mgmt_addr(*tlv_data*)
Returns Remote Mgmt Addr from the TLV.

get_remote_port(*tlv_data*)
Returns Remote Port from the TLV.

get_remote_port_id_local(*tlv_data*)
Returns Remote Port ID Local from the TLV.

get_remote_port_id_mac(*tlv_data*)
Returns Remote Port ID MAC from the TLV.

get_remote_sys_desc(*tlv_data*)
Returns Remote Sys Desc from the TLV.

get_remote_sys_name(*tlv_data*)
Returns Remote Sys Name from the TLV.

run_lldptool(*args*)
Function for invoking the lldptool utility.

networking_cisco.apps.saf.agent.topo_disc.topo_disc module

This file contains the implementation of Topology Discovery of servers and their associated leaf switches using Open source implementation of LLDP. www.open-lldp.org

class networking_cisco.apps.saf.agent.topo_disc.topo_disc.**TopoDisc**(*cb*, *root_helper*, *intf_list=None*, *all_intf=True*)
Bases: *networking_cisco.apps.saf.agent.topo_disc.topo_disc.TopoDiscPubApi*

Topology Discovery Top level class once.

cfg_intf(*protocol_interface*, *phy_interface=None*)
Called by application to add an interface to the list.

cfg_lldp_interface(*protocol_interface*, *phy_interface=None*)
Cfg LLDP on interface and create object.

cfg_lldp_interface_list(*intf_list*)
This routine configures LLDP on the given interfaces list.

cmp_store_tlv_params(*intf*, *tlv_data*)
Compare and store the received TLV.

Compares the received TLV with stored TLV. Store the new TLV if it is different.

create_attr_obj(*protocol_interface*, *phy_interface*)
Creates the local interface attribute object and stores it.

get_attr_obj(*intf*)
Retrieve the interface object.

periodic_discovery_task()
Periodic task that checks the interface TLV attributes.

uncfg_intf(*intf*)
Called by application to remove an interface to the list.

From an applications perspective, it makes sense to have this function. But, here no action can be taken for the following reasons, but just having it as a place-holder for tomorrow. => Can't remove interface

from the list since DB in server may appear stale. self.intf_list.remove(intf) => One can just remove the interface DB, but need to retry that till it succeeds, so it has to be in periodic loop. => So, currently leaving it as is, since LLDP frames won't be obtained over the bridge, the periodic handler will automatically remove the DB for this interface from server

class networking_cisco.apps.saf.agent.topo_disc.topo_disc.**TopoDiscPubApi**
Bases: object

classmethod get_lldp_status(*intf*)
Retrieves the LLDP status.

classmethod store_obj(*intf*, *obj*)
Stores the topo object.

topo_intf_obj_dict = {}

class networking_cisco.apps.saf.agent.topo_disc.topo_disc.**TopoIntfAttr**(*protocol_interface*, *phy_interface*)
Bases: object

Class that stores the interface attributes.

cmp_update_bond_intf(*bond_interface*)
Update the bond interface and its members.

Update the bond interface, if this interface is a part of bond Return True if there's a change.

get_db_retry_status()
Retrieve the RPC retru status.

This retrieves the number of times RPC was retried with the server.

get_lldp_status()
Retrieve the LLDP cfg status.

get_phy_interface()
Retrieves the physical interface.

get_topo_disc_send_cnt()
Retrieve the topology status send count for this interface.

incr_topo_disc_send_cnt()
Increment the topology status send count for this interface.

init_params(*protocol_interface*, *phy_interface*)
Initializing parameters.

remote_chassis_id_mac_uneq_store(*remote_chassis_id_mac*)
This function saves the Chassis MAC, if different from stored.

remote_evb_cfgd_uneq_store(*remote_evb_cfgd*)
This saves the EVB cfg, if it is not the same as stored.

remote_evb_mode_uneq_store(*remote_evb_mode*)
Saves the EVB mode, if it is not the same as stored.

remote_mgmt_addr_uneq_store(*remote_mgmt_addr*)
This function saves the MGMT address, if different from stored.

remote_port_id_mac_uneq_store(*remote_port_id_mac*)
This function saves the port MAC, if different from stored.

remote_port_uneq_store(*remote_port*)
This function saves the port, if different from stored.

remote_sys_desc_uneq_store(*remote_system_desc*)
This function saves the system desc, if different from stored.

remote_sys_name_uneq_store(*remote_system_name*)
This function saves the system name, if different from stored.

reset_topo_disc_send_cnt()
Reset the topology status send count for this interface.

store_db_retry_status(*status*)
This stores the number of times RPC was retried with the server.

update_lldp_status(*status*)
Update the LLDP cfg status.

networking_cisco.apps.saf.agent.topo_disc.topo_disc_constants module

Module contents

networking_cisco.apps.saf.agent.vdp package

Submodules

networking_cisco.apps.saf.agent.vdp.dfa_vdp_mgr module

class networking_cisco.apps.saf.agent.vdp.dfa_vdp_mgr.**VdpMgr**(*config_dict*, *rpc_client*, *host-name*)

Bases: object

Responsible for Handling VM/Uplink requests.

check_periodic_bulk_vm_notif_rcvd()
Bulk VM check handler called from periodic uplink detection.

This gets called by the 'normal' stage of uplink detection. The bulk VM event sends all the VM's running in this agent. Sometimes during upgrades, it was found that due to some race condition, the server does not send the Bulk VM event. Whenever, a save_uplink is done by the agent, the server sends the Bulk VM event. If Bulk VM event is not received after few attempts, save_uplink is done to request the Bulk VM list. It's not protected with a mutex, since worst case, Bulk VM event will be sent twice, which is not that bad. When uplink is detected for the first time, it will hit the below else case and there a save_uplink is anyways done.

dfa_uplink_restart(*uplink_dict*)

is_openstack_running()
Currently it just checks for the presence of both the bridges.

is_uplink_received()
Returns whether uplink information is received after restart.

Not protecting this with a mutex, since this gets called inside the loop from dfa_agent and having a mutex is a overkill. Worst case, during multiple restarts on server and when the corner case is hit, this may return an incorrect value of False when _dfa_uplink_restart is at the middle of execution. Returning an incorrect value of False, may trigger an RPC to the server to retrieve the uplink one extra time. _dfa_uplink_restart will not get executed twice, since that is anyway protected with a mutex.

process_bulk_vm_event(*msg*, *phy_uplink*)
Process the VM bulk event usually after a restart.

process_err_queue()

process_queue()

process_uplink_event(*msg*, *phy_uplink*)

process_vm_event(*msg*, *phy_uplink*)

read_static_uplink()
Read the static uplink from file, if given.

save_topo_disc_params(*intf*, *topo_disc_obj*)

save_uplink(*uplink=''*, *veth_intf=''*, *fail_reason=''*)

start()

static_uplink_detect(*veth*)
Return the static uplink based on argument passed.

The very first time, this function is called, it returns the uplink port read from a file. After restart, when this function is called the first time, it returns 'normal' assuming a veth is passed to this function which will be the case if uplink processing is successfully done. If user modified the uplink configuration and restarted, a 'down' will be returned to clear the old uplink.

topo_disc_cb(*intf*, *topo_disc_obj*)

update_vm_result(*port_uuid*, *result*, *lvid=None*, *vdp_vlan=None*, *fail_reason=None*)

uplink_bond_intf_process()
Process the case when uplink interface becomes part of a bond.

This is called to check if the phy interface became a part of the bond. If the below condition is True, this means, a physical interface that was not a part of a bond was earlier discovered as uplink and now that interface became part of the bond. Usually, this doesn't happen as LLDP and in turn this function will first detect a 'down' followed by an 'up'. When regular interface becomes part of bond, it's rare for it to hit this 'normal' case. But, still providing the functionality if it happens. The following is done : a. Bring down the physical interface by sending a 'down' event b. Add the bond interface by sending an 'up' event Consquently, when bond is added that will be assigned to self.phy_uplink. Then, the below condition will be False. i.e.. 'get_bond_intf' will return False, when the argument is 'bond0'.

vdp_uplink_proc()
Periodic handler to detect the uplink interface to the switch.

-> restart_uplink_called: should be called by agent initially to set the stored uplink and veth from DB -> process_uplink_ongoing: Will be set when uplink message is enqueue and reset when dequeued and processed completely -> uplink_det_compl: Will be set to True when a valid uplink is detected and object created. Will be reset when uplink is down -> phy_uplink: Is the uplink interface -> veth_intf : Signifies the veth interface.

vdp_uplink_proc_top()

vdp_vlan_change_cb(*port_uuid*, *lvid*, *vdp_vlan*, *fail_reason*)
Callback function for updating the VDP VLAN in DB.

vdp_vm_event(*vm_dict_list*)

class networking_cisco.apps.saf.agent.vdp.dfa_vdp_mgr.**VdpMsgPriQue**
Bases: object

VDP Message Queue.

dequeue()

dequeue_nonblock()

enqueue(*priority*, *msg*)

is_not_empty()

class networking_cisco.apps.saf.agent.vdp.dfa_vdp_mgr.**VdpQueMsg**(*msg_type*, *port_uuid=None*, *vm_mac=None*, *oui=None*, *net_uuid=None*, *segmentation_id=None*, *status=None*, *vm_bulk_list=None*, *phy_uplink=None*, *br_int=None*, *br_ex=None*, *root_helper=None*)

Bases: object

Construct VDP Message.

construct_uplink_msg(*status*, *phy_uplink*, *br_int*, *br_ex*, *root_helper*)

construct_vm_bulk_sync_msg(*vm_bulk_list*, *phy_uplink*)

construct_vm_msg(*port_uuid*, *vm_mac*, *net_uuid*, *segmentation_id*, *status*, *oui*, *phy_uplink*)

get_ext_br()

get_integ_br()

get_mac()

get_net_uuid()

get_oui()

get_port_uuid()

get_root_helper()

get_segmentation_id()

get_status()

get_uplink()

set_uplink(*uplink*)

networking_cisco.apps.saf.agent.vdp.lldpad module

This file contains the implementation of OpenStack component of VDP. VDP is a part of LLDP Agent Daemon (lldpad). For more information on VDP, pls visit http://www.ieee802.org/1/pages/802.1bg.html

class networking_cisco.apps.saf.agent.vdp.lldpad.**LldpadDriver**(*port_name*, *phy_uplink*, *root_helper*, *is_ncb=True*, *is_nb=False*)

Bases: `object`

LLDPad driver class.

check_filter_validity(*reply*, *filter_str*)
Check for the validify of the filter.

check_hints(*reply*)
Parse the hints to check for errors.

clear_oui(*port_uuid*)
Clears the OUI specific info.

Parameters **uuid** – vNIC UUID

Currently only one OUI per VSI fixme(padkrish)

clear_uplink()

clear_vdp_vsi(*port_uuid*)
Stores the vNIC specific info for VDP Refresh.

Parameters **uuid** – vNIC UUID

construct_vdp_dict(*mode*, *mgrid*, *typeid*, *typeid_ver*, *vsiid_frmt*, *vsiid*, *filter_frmt*, *gid*, *mac*, *vlan*, *oui_id*, *oui_data*)
Constructs the VDP Message.

Please refer http://www.ieee802.org/1/pages/802.1bg.html VDP Section for more detailed information :param mode: Associate or De-associate :param mgrid: MGR ID :param typeid: Type ID :param typeid_ver: Version of the Type ID :param vsiid_frmt: Format of the following VSI argument :param vsiid: VSI value :param filter_frmt: Filter Format :param gid: Group ID the vNIC belongs to :param mac: MAC Address of the vNIC :param vlan: VLAN of the vNIC :param oui_id: OUI Type :param oui_data: OUI Data :return vdp_keyword_str: Dictionary of VDP arguments and values

crosscheck_query_vsiid_mac(*reply*, *vsiid*, *mac*)
Cross Check the reply against the input vsiid,mac for get query.

crosscheck_reply_vsiid_mac(*reply*, *vsiid*, *mac*)
Cross Check the reply against the input vsiid,mac for associate.

enable_evb()
Function to enable EVB on the interface.

enable_gpid()
Function to enable Group ID on the interface.

This is needed to use the MAC, GID, VID Filter.

enable_lldp()
Function to enable LLDP on the interface.

gen_cisco_vdp_oui(*oui_id*, *oui_data*)
Cisco specific handler for constructing OUI arguments.

gen_oui_str(*oui_list*)
Generate the OUI string for vdptool.

get_vdp_failure_reason(*reply*)
Parse the failure reason from VDP.

get_vlan_from_associate_reply(*reply*, *vsiid*, *mac*)
Parse the associate reply from VDP daemon to get the VLAN value.

get_vlan_from_query_reply(*reply*, *vsiid*, *mac*)
Parse the query reply from VDP daemon to get the VLAN value.

read_vdp_cfg()

run_lldptool(*args*)
Function for invoking the lldptool utility.

run_vdptool(*args*, *oui_args=None*)
Function that runs the vdptool utility.

send_vdp_assoc(*vsiid=None*, *mgrid=None*, *typeid=None*, *typeid_ver=None*, *vsiid_frmt=5*, *filter_frmt=4*, *gid=0*, *mac=''*, *vlan=0*, *oui_id=''*, *oui_data=''*, *sw_resp=False*)
Sends the VDP Associate Message.

Please refer http://www.ieee802.org/1/pages/802.1bg.html VDP Section for more detailed information :param vsiid: VSI value, Only UUID supported for now :param mgrid: MGR ID :param typeid: Type ID :param typeid_ver: Version of the Type ID :param vsiid_frmt: Format of the following VSI argument :param filter_frmt: Filter Format. Only <GID,MAC,VID> supported for now :param gid: Group ID the vNIC belongs to :param mac: MAC Address of the vNIC :param vlan: VLAN of the vNIC :param oui_id: OUI Type :param oui_data: OUI Data :param sw_resp: Flag indicating if response is required from the daemon :return vlan: VLAN value returned by vdptool which in turn is given : by Switch

send_vdp_deassoc(*vsiid=None*, *mgrid=None*, *typeid=None*, *typeid_ver=None*, *vsiid_frmt=5*, *filter_frmt=4*, *gid=0*, *mac=''*, *vlan=0*, *oui_id=''*, *oui_data=''*, *sw_resp=False*)
Sends the VDP Dis-Associate Message.

Please refer http://www.ieee802.org/1/pages/802.1bg.html VDP Section for more detailed information :param vsiid: VSI value, Only UUID supported for now :param mgrid: MGR ID :param typeid: Type ID :param typeid_ver: Version of the Type ID :param vsiid_frmt: Format of the following VSI argument :param filter_frmt: Filter Format. Only <GID,MAC,VID> supported for now :param gid: Group ID the vNIC belongs to :param mac: MAC Address of the vNIC :param vlan: VLAN of the vNIC :param oui_id: OUI Type :param oui_data: OUI Data :param sw_resp: Flag indicating if response is required from the daemon

send_vdp_msg(*mode*, *mgrid*, *typeid*, *typeid_ver*, *vsiid_frmt*, *vsiid*, *filter_frmt*, *gid*, *mac*, *vlan*, *oui_id*, *oui_data*, *sw_resp*)
Constructs and Sends the VDP Message.

Please refer http://www.ieee802.org/1/pages/802.1bg.html VDP Section for more detailed information :param mode: Associate or De-associate :param mgrid: MGR ID :param typeid: Type ID :param typeid_ver: Version of the Type ID :param vsiid_frmt: Format of the following VSI argument :param vsiid: VSI value :param filter_frmt: Filter Format :param gid: Group ID the vNIC belongs to :param mac: MAC Address of the vNIC :param vlan: VLAN of the vNIC :param oui_id: OUI Type :param oui_data: OUI Data :param sw_resp: Flag indicating if response is required from the daemon :return reply: Reply from vdptool

send_vdp_query_msg(*mode*, *mgrid*, *typeid*, *typeid_ver*, *vsiid_frmt*, *vsiid*, *filter_frmt*, *gid*, *mac*, *vlan*, *oui_id*, *oui_data*)
Constructs and Sends the VDP Query Message.

Please refer http://www.ieee802.org/1/pages/802.1bg.html VDP Section for more detailed information :param mode: Associate or De-associate :param mgrid: MGR ID :param typeid: Type ID :param typeid_ver: Version of the Type ID :param vsiid_frmt: Format of the following VSI argument :param vsiid: VSI value :param filter_frmt: Filter Format :param gid: Group ID the vNIC belongs to :param mac: MAC Address of the vNIC :param vlan: VLAN of the vNIC :param oui_id: OUI Type :param oui_data: OUI Data :param sw_resp: Flag indicating if response is required from the daemon :return reply: Reply from vdptool

send_vdp_vnic_down(*port_uuid=None*, *vsiid=None*, *mgrid=None*, *typeid=None*, *typeid_ver=None*, *vsiid_frmt=5*, *filter_frmt=4*, *gid=0*, *mac=”*, *vlan=0*, *oui=”*)
Interface function to apps, called for a vNIC DOWN.

This currently sends an VDP dis-associate message. Please refer http://www.ieee802.org/1/pages/802.1bg.html VDP Section for more detailed information :param uuid: uuid of the vNIC :param vsiid: VSI value, Only UUID supported for now :param mgrid: MGR ID :param typeid: Type ID :param typeid_ver: Version of the Type ID :param vsiid_frmt: Format of the following VSI argument :param filter_frmt: Filter Format. Only <GID,MAC,VID> supported for now :param gid: Group ID the vNIC belongs to :param mac: MAC Address of the vNIC :param vlan: VLAN of the vNIC :param oui_id: OUI Type :param oui_data: OUI Data :param sw_resp: Flag indicating if response is required from the daemon

send_vdp_vnic_up(*port_uuid=None*, *vsiid=None*, *mgrid=None*, *typeid=None*, *typeid_ver=None*, *vsiid_frmt=5*, *filter_frmt=4*, *gid=0*, *mac=”*, *vlan=0*, *oui=None*, *new_network=False*, *vsw_cb_fn=None*, *vsw_cb_data=None*)
Interface function to apps, called for a vNIC UP.

This currently sends an VDP associate message. Please refer http://www.ieee802.org/1/pages/802.1bg.html VDP Section for more detailed information :param uuid: uuid of the vNIC :param vsiid: VSI value, Only UUID supported for now :param mgrid: MGR ID :param typeid: Type ID :param typeid_ver: Version of the Type ID :param vsiid_frmt: Format of the following VSI argument :param filter_frmt: Filter Format. Only <GID,MAC,VID> supported for now :param gid: Group ID the vNIC belongs to :param mac: MAC Address of the vNIC :param vlan: VLAN of the vNIC :param oui_id: OUI Type :param oui_data: OUI Data :param sw_resp: Flag indicating if response is required from the daemon :return reply: VLAN reply from vdptool

store_oui(*port_uuid*, *oui_type*, *oui_data*)
Function for storing the OUI.

param uuid: UUID of the vNIC param oui_type: OUI ID param oui_data: OUI Opaque Data

store_vdp_vsi(*port_uuid*, *mgrid*, *typeid*, *typeid_ver*, *vsiid_frmt*, *vsiid*, *filter_frmt*, *gid*, *mac*, *vlan*, *new_network*, *reply*, *oui_id*, *oui_data*, *vsw_cb_fn*, *vsw_cb_data*, *reason*)
Stores the vNIC specific info for VDP Refresh.

Parameters

- **uuid** – vNIC UUID
- **mgrid** – MGR ID
- **typeid** – Type ID
- **typeid_ver** – Version of the Type ID
- **vsiid_frmt** – Format of the following VSI argument
- **vsiid** – VSI value
- **filter_frmt** – Filter Format
- **gid** – Group ID the vNIC belongs to
- **mac** – MAC Address of the vNIC
- **vlan** – VLAN of the vNIC
- **new_network** – Is this the first vNIC of this network
- **reply** – Response from the switch
- **oui_id** – OUI Type
- **oui_data** – OUI Data

- **vsw_cb_fn** – Callback function from the app.
- **vsw_cb_data** – Callback data for the app.
- **reason** – Failure Reason

networking_cisco.apps.saf.agent.vdp.lldpad.**enable_lldp**(*self*, *port_name*, *is_ncb=True*, *is_nb=False*)

Function to enable LLDP on the interface.

networking_cisco.apps.saf.agent.vdp.lldpad_constants module

Service VDP 2.2 constants.

networking_cisco.apps.saf.agent.vdp.ovs_vdp module

This file contains the mixin class implementation of OVS extensions for VDP. VDP is a part of LLDP Agent Daemon (lldpad). For more information on VDP, pls visit http://www.ieee802.org/1/pages/802.1bg.html

class networking_cisco.apps.saf.agent.vdp.ovs_vdp.**LocalVlan**(*vlan*, *segmentation_id*)

Bases: object

any_consistent_vlan()

any_valid_vlan()

decr_reset_vlan(*port_uuid*, *new_vlan*)

get_portid_fail_reason(*port_id*)

get_portid_vlan(*port_id*)

reset_port_vlan(*vdp_vlan*)

set_fail_reason(*port_uuid*, *fail_reason*)

set_port_uuid(*port_uuid*, *vdp_vlan*, *fail_reason*)

set_port_vlan(*vdp_vlan*)

set_portid_fail_reason(*port_id*, *fail_reason*)

set_portid_vlan(*port_id*, *new_vlan*)

class networking_cisco.apps.saf.agent.vdp.ovs_vdp.**OVSNeutronVdp**(*uplink*, *integ_br*, *ext_br*, *root_helper*, *vdp_vlan_cb*, *vdp_mode=10*)

Bases: object

Implements the VDP specific changes in OVS.

Creating the veth pairs, programming the flows for VDP, deleting the VDP specific flows, communicating with VDP (lldpad) daemon using lldpad class are some of the functionality provided by this class.

clear_obj_params()

delete_vdp_flows()

find_interconnect_ports()

Find the internal veth or patch ports.

gen_veth_str(*const_str*, *intf_str*)
Generate a veth string.

Concatenates the constant string with remaining available length of interface string from trailing position.

get_lldp_local_bridge_port()

get_lldp_ovs_bridge_port()

get_lvid_vdp_vlan(*net_uuid*, *port_uuid*)
Retrieve the Local Vlan ID and VDP Vlan.

get_uplink_fail_reason()

is_lldpad_setup_done()

pop_local_cache(*port_uuid*, *mac*, *net_uuid*, *lvid*, *vdp_vlan*, *segmentation_id*)
Populate the local cache after restart.

port_down_segment_mode(*lldpad_port*, *port_uuid*, *mac*, *net_uuid*, *segmentation_id*, *oui*)

port_up_segment_mode(*lldpad_port*, *port_name*, *port_uuid*, *mac*, *net_uuid*, *segmentation_id*, *oui*)

program_vdp_flows(*lldp_ovs_portnum*, *phy_port_num*)

program_vm_ovs_flows(*lvid*, *old_vlan*, *new_vlan*)

provision_vdp_overlay_networks(*port_uuid*, *mac*, *net_uuid*, *segmentation_id*, *lvid*, *oui*)
Provisions a overlay type network configured using VDP.

Parameters

- **port_uuid** – the uuid of the VM port.
- **mac** – the MAC address of the VM.
- **net_uuid** – the uuid of the network associated with this vlan.
- **segmentation_id** – the VID for 'vlan' or tunnel ID for 'tunnel'

Lvid Local VLAN ID

Oui OUI Parameters

send_vdp_port_event(*port_uuid*, *mac*, *net_uuid*, *segmentation_id*, *status*, *oui*)
Send vNIC UP/Down event to VDP.

Parameters

- **port** – a ovslib.VifPort object.
- **net_uuid** – the net_uuid this port is to be associated with.
- **segmentation_id** – the VID for 'vlan' or tunnel ID for 'tunnel'
- **status** – Type of port event. 'up' or 'down'

send_vdp_port_event_internal(*port_uuid*, *mac*, *net_uuid*, *segmentation_id*, *status*, *oui*)
Send vNIC UP/Down event to VDP.

Parameters

- **port_uuid** – a ovslib.VifPort object.
- **net_uuid** – the net_uuid this port is to be associated with.
- **segmentation_id** – the VID for 'vlan' or tunnel ID for 'tunnel'
- **status** – Type of port event. 'up' or 'down'

Mac MAC address of the VNIC

Oui OUI Parameters

setup_lldpad_ports()
Setup the flows for passing LLDP/VDP frames in OVS.

unprovision_vdp_overlay_networks(*net_uuid*, *lvid*, *vdp_vlan*, *oui*)
Unprovisions a overlay type network configured using VDP.

Parameters **net_uuid** – the uuid of the network associated with this vlan.

Lvid Local VLAN ID

Vdp_vlan VDP VLAN ID

Oui OUI Parameters

vdp_vlan_change(*vsw_cb_data*, *vdp_vlan*, *fail_reason*)
Callback Function from VDP when provider VLAN changes.

This will be called only during error cases when switch reloads or when compute reloads.

vdp_vlan_change_internal(*vsw_cb_data*, *vdp_vlan*, *fail_reason*)
Callback Function from VDP when provider VLAN changes.

This will be called only during error cases when switch reloads or when compute reloads.

networking_cisco.apps.saf.agent.vdp.ovs_vdp.**delete_uplink_and_flows**(*root_helper*, *br_ex*, *port_name*)

networking_cisco.apps.saf.agent.vdp.ovs_vdp.**glob_delete_vdp_flows**(*br_ex*, *root_helper*)

networking_cisco.apps.saf.agent.vdp.ovs_vdp.**is_bridge_present**(*br*, *root_helper*)

networking_cisco.apps.saf.agent.vdp.ovs_vdp.**is_uplink_already_added**(*root_helper*, *br_ex*, *port_name*)

networking_cisco.apps.saf.agent.vdp.vdp_constants module

LLDP/VDP Constants.

Module contents

Submodules

networking_cisco.apps.saf.agent.detect_uplink module

networking_cisco.apps.saf.agent.detect_uplink.**detect_uplink**(*input_string=None*)

networking_cisco.apps.saf.agent.detect_uplink.**detect_uplink_auto**(*input_string*)

networking_cisco.apps.saf.agent.detect_uplink.**detect_uplink_non_auto**(*input_string*)

networking_cisco.apps.saf.agent.detect_uplink.**find_uplink**()

networking_cisco.apps.saf.agent.detect_uplink.**read_file**(*file_name*)

networking_cisco.apps.saf.agent.detect_uplink.**run_cmd_line**(*cmd_str*, *stderr=None*, *shell=True*, *echo_cmd=False*, *check_result=False*)

networking_cisco.apps.saf.agent.dfa_agent module

class networking_cisco.apps.saf.agent.dfa_agent.**DfaAgent**(*host*, *rpc_qn*)
Bases: object

DFA agent.

is_uplink_received()
Finds if uplink information is received and processed.

request_uplink_info()

send_heartbeat()

setup_client_rpc()
Setup RPC client for dfa agent.

setup_rpc()
Setup RPC server for dfa agent.

start_iptables_task()

start_rpc()

start_rpc_task()

start_tasks()

stop_rpc()

class networking_cisco.apps.saf.agent.dfa_agent.**RpcCallBacks**(*vdpd*, *ipt_drvr*)
Bases: object

RPC call back methods.

send_msg_to_agent(*context*, *msg*)

send_vm_info(*context*, *msg*)

update_ip_rule(*context*, *msg*)

networking_cisco.apps.saf.agent.dfa_agent.**main**()

networking_cisco.apps.saf.agent.dfa_agent.**save_my_pid**(*cfg*)

networking_cisco.apps.saf.agent.iptables_driver module

class networking_cisco.apps.saf.agent.iptables_driver.**IpMacPort**(*ip*, *mac*, *port*)
Bases: object

This class keeps host rule information.

class networking_cisco.apps.saf.agent.iptables_driver.**IptablesDriver**(*cfg*)
Bases: object

This class provides API to update iptables rule.

add_rule_entry(*rule_info*)
Add host data object to the rule_info list.

create_thread()
Create a task to process event for updating iptables.

enqueue_event(*event*)
Enqueue the given event.

The event contains host data (ip, mac, port) which will be used to update the spoofing rule for the host in the iptables.

process_rule_info()
Task responsible for processing event queue.

remove_rule_entry(*rule_info*)
Remove host data object from rule_info list.

update_ip_rule(*ip*, *mac*)
Update a rule associated with given ip and mac.

update_iptables()
Update iptables based on information in the rule_info.

update_rule_entry(*rule_info*)
Update the rule_info list.

Module contents

networking_cisco.apps.saf.common package

Submodules

networking_cisco.apps.saf.common.config module

class networking_cisco.apps.saf.common.config.**CiscoDFAConfig**
Bases: object

Cisco DFA Mechanism Driver Configuration class.

cfg

networking_cisco.apps.saf.common.constants module

networking_cisco.apps.saf.common.dfa_exceptions module

Exceptions used by DFA enabler

exception networking_cisco.apps.saf.common.dfa_exceptions.**ConfigProfileFwdModeNotFound**(**kw
Bases: neutron_lib.exceptions.NotFound

Config Profile forwarding mode cannot be found.

message = 'Forwarding Mode for network %(network_id)s could not be found.'

exception networking_cisco.apps.saf.common.dfa_exceptions.**ConfigProfileIdNotFound**(**kwargs)
Bases: neutron_lib.exceptions.NotFound

Config Profile ID cannot be found.

message = 'Config Profile %(profile_id)s could not be found.'

exception networking_cisco.apps.saf.common.dfa_exceptions.**ConfigProfileNameNotFound**(***kwargs*)
Bases: neutron_lib.exceptions.NotFound

Config Profile name cannot be found.

message = 'Config Profile %(name)s could not be found.'

exception networking_cisco.apps.saf.common.dfa_exceptions.**ConfigProfileNotFound**(***kwargs*)
Bases: neutron_lib.exceptions.NotFound

Config Profile cannot be found.

message = 'Config profile for network %(network_id)s could not be found.'

exception networking_cisco.apps.saf.common.dfa_exceptions.**DfaAgentFailed**(***kwargs*)
Bases: neutron_lib.exceptions.ServiceUnavailable

Failure in running DfaAgent.

message = 'OpenStack is not running: %(reason)s.'

exception networking_cisco.apps.saf.common.dfa_exceptions.**DfaClientRequestFailed**(***kwargs*)
Bases: neutron_lib.exceptions.ServiceUnavailable

Request to DCNM failed.

message = 'Request to DCNM failed: %(reason)s.'

exception networking_cisco.apps.saf.common.dfa_exceptions.**InvalidInput**(***kwargs*)
Bases: neutron_lib.exceptions.InvalidInput

Invalid Input specified.

message = 'Invalid input for operation: %(error_message)s.'

exception networking_cisco.apps.saf.common.dfa_exceptions.**NetworkNotFound**(***kwargs*)
Bases: neutron_lib.exceptions.NotFound

Network cannot be found.

message = 'Network %(network_id)s could not be found.'

exception networking_cisco.apps.saf.common.dfa_exceptions.**ProjectIdNotFound**(***kwargs*)
Bases: neutron_lib.exceptions.NotFound

Project ID cannot be found.

message = 'Project ID %(project_id)s could not be found.'

networking_cisco.apps.saf.common.dfa_logger module

DFA logging helper module.

networking_cisco.apps.saf.common.dfa_logger.**getLogger**(*name*)

networking_cisco.apps.saf.common.dfa_logger.**setup_logger**(*project*, *cfg*)

networking_cisco.apps.saf.common.dfa_sys_lib module

class networking_cisco.apps.saf.common.dfa_sys_lib.**BaseOVS**(*root_helper*)

Bases: object

add_bridge(*bridge_name*)

bridge_exists(*bridge_name*)

delete_bridge(*bridge_name*)

get_bridge_name_for_port_name(*port_name*)

port_exists(*port_name*)

run_vsctl(*args*, *check_error=False*)

class networking_cisco.apps.saf.common.dfa_sys_lib.**OVSBridge**(*br_name*, *root_helper*)

Bases: *networking_cisco.apps.saf.common.dfa_sys_lib.BaseOVS*

add_flow(***kwargs*)

add_port(*port_name*)

clear_db_attribute(*table_name*, *record*, *column*)

create()

db_get_val(*table*, *record*, *column*, *check_error=False*)

delete_flows(***kwargs*)

delete_port(*port_name*)

destroy()

do_action_flows(*action*, *kwargs_list*)

dump_flows_for(***kwargs*)

get_ofport_name(*iface_uuid*)

get_port_name_list()

get_port_ofport(*port_name*)

get_port_vlan_tag(*port_name*)

remove_all_flows()

run_ofctl(*cmd*, *args*, *process_input=None*)

set_db_attribute(*table_name*, *record*, *column*, *value*)

set_secure_mode()

networking_cisco.apps.saf.common.dfa_sys_lib.**create_process**(*cmd*, *root_helper=None*, *addl_env=None*, *log_output=True*)

Create a process object for the given command.

The return value will be a tuple of the process object and the list of command arguments used to create it.

networking_cisco.apps.saf.common.dfa_sys_lib.**delete_port_glob**(*root_helper*, *br_ex*, *port_name*)

networking_cisco.apps.saf.common.dfa_sys_lib.**execute**(*cmd*, *root_helper=None*, *process_input=None*, *addl_env=None*, *check_exit_code=True*, *return_stderr=False*, *log_fail_as_error=True*, *log_output=True*)

networking_cisco.apps.saf.common.dfa_sys_lib.**get_all_run_phy_intf**()
Retrieve all physical interfaces that are operationally up.

networking_cisco.apps.saf.common.dfa_sys_lib.**get_bond_intf**(*intf*)

networking_cisco.apps.saf.common.dfa_sys_lib.**get_bridge_name_for_port_name_glob**(*root_helper*, *port_name*)

networking_cisco.apps.saf.common.dfa_sys_lib.**get_bridges**(*root_helper*)

networking_cisco.apps.saf.common.dfa_sys_lib.**get_member_ports**(*intf*)

networking_cisco.apps.saf.common.dfa_sys_lib.**get_peer**(*root_helper*, *port*)

networking_cisco.apps.saf.common.dfa_sys_lib.**is_intf_bond**(*intf*)

networking_cisco.apps.saf.common.dfa_sys_lib.**is_intf_up**(*intf*)
Function to check if a interface is up.

networking_cisco.apps.saf.common.dfa_sys_lib.**is_patch**(*root_helper*, *port*)

networking_cisco.apps.saf.common.dfa_sys_lib.**is_valid_vlan_tag**(*vlan*)

networking_cisco.apps.saf.common.dfa_sys_lib.**port_exists_glob**(*root_helper*, *port_name*)

networking_cisco.apps.saf.common.dfa_sys_lib.**subprocess_popen**(*args*, *stdin=None*, *stdout=None*, *stderr=None*, *shell=False*, *env=None*)

networking_cisco.apps.saf.common.rpc module

class networking_cisco.apps.saf.common.rpc.**DfaNotifcationListener**(*topic*, *url*, *endpoints*, *exchange=None*, *fanout=False*)

Bases: object

RPC Client class for DFA enabler.

start()

stop()

wait()

class networking_cisco.apps.saf.common.rpc.**DfaNotificationEndpoints**(*endp*)
Bases: object

Notification endpoints.

info(*ctxt*, *publisher_id*, *event_type*, *payload*, *metadata*)

class networking_cisco.apps.saf.common.rpc.**DfaRpcClient**(*transport_url*, *topic*, *exchange=None*, *fanout=False*)

Bases: object

RPC Client class for DFA enabler.

call(*msg*)

cast(*msg*)

make_msg(*method*, *context*, ***kwargs*)

class networking_cisco.apps.saf.common.rpc.**DfaRpcServer**(*topic*, *server*, *url*, *endpoints*, *exchange=None*, *fanout=False*, *executor='eventlet'*)

Bases: object

RPC server class for DFA enabler.

start()

stop()

wait()

networking_cisco.apps.saf.common.utils module

class networking_cisco.apps.saf.common.utils.**Dict2Obj**(*d*)

Bases: object

Convert a dictionary to an object.

class networking_cisco.apps.saf.common.utils.**EventProcessingThread**(*name*, *obj*, *task*, *excq=None*)

Bases: threading.Thread

Event processing thread.

am_i_active

name

run()

class networking_cisco.apps.saf.common.utils.**PeriodicTask**(*interval*, *func*, ***kwargs*)

Bases: object

Periodic task

run()

stop()

networking_cisco.apps.saf.common.utils.**find_agent_host_id**(*this_host*)

Returns the neutron agent host id for RHEL-OSP6 HA setup.

networking_cisco.apps.saf.common.utils.**get_uuid**()

networking_cisco.apps.saf.common.utils.**is_valid_ipv4**(*addr*)

networking_cisco.apps.saf.common.utils.**is_valid_mac**(*addr*)
Check the syntax of a given mac address.

The acceptable format is xx:xx:xx:xx:xx:xx

networking_cisco.apps.saf.common.utils.**lock**()

networking_cisco.apps.saf.common.utils.**make_cidr**(*gw*, *mask*)
Create network address in CIDR format.

Return network address for a given gateway address and netmask.

networking_cisco.apps.saf.common.utils.**utc_time**(*ct*)

networking_cisco.apps.saf.common.utils.**utc_time_lapse**(*lapse*)

Module contents

networking_cisco.apps.saf.db package

Submodules

networking_cisco.apps.saf.db.dfa_db_api module

networking_cisco.apps.saf.db.dfa_db_api.**configure_db**(*cfg*)

networking_cisco.apps.saf.db.dfa_db_api.**get_session**()

networking_cisco.apps.saf.db.dfa_db_models module

class networking_cisco.apps.saf.db.dfa_db_models.**DfaAgentsDb**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents DFA agent.

configurations

created

heartbeat

host

class networking_cisco.apps.saf.db.dfa_db_models.**DfaDBMixin**(*cfg*)
Bases: object

Database API.

add_fw_db(*fw_id*, *fw_data*, *result=None*)

add_network_db(*net_id*, *net_data*, *source*, *result*)

add_project_db(*pid*, *name*, *dci_id*, *result*)

add_vms_db(*vm_data*, *result*)

append_state_final_result(*fw_id*, *cur_res*, *state*)

clear_fw_entry_by_netid(*net_id*)

conv_db_dict(*alloc*)

del_project_db(*pid*)

delete_fw(*fw_id*)

delete_network_db(*net_id*)

delete_vm_db(*port_id*)

get_agent_configurations(*host*)

get_all_fw_db()

get_all_networks()

get_all_projects()

get_fialed_projects_entries(*fail_res*)

get_fw(*fw_id*)

get_fw_by_netid(*netid*)

get_fw_by_rtr_netid(*netid*)

get_fw_by_rtrid(*rtrid*)

get_fw_by_tenant_id(*tenant_id*)

get_fw_rule_by_id(*fw_id*)

get_network(*net_id*)

get_network_by_name(*name*)

get_network_by_segid(*segid*)

get_project_id(*name*)

get_project_name(*pid*)

get_str_dict(*fw_data*)

get_vm(*port_id*)

get_vms()

get_vms_for_this_req(***req*)

update_agent_configurations(*host*, *configs*)

update_agent_db(*agent_info*)

update_fw_db(*fw_id*, *fw_data*)

update_fw_db_dev_status(*fw_id*, *status*)

update_fw_db_final_result(*fw_id*, *result*)

update_fw_db_mgmt_ip(*fw_id*, *mgmt_ip*)

update_fw_db_result(*fw_id*, *fw_data*)

update_network(*net_id*, ***params*)

update_network_db(*net_id*, *result*)

update_project_entry(*pid*, *dci_id*, *result*)

update_vm_db(*vm_port_id*, ***params*)

class networking_cisco.apps.saf.db.dfa_db_models.**DfaFwInfo**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents Firewall info.

dcnm_provision_status

device_provision_status

fw_id

fw_mgmt_ip

fw_type

in_network_id

in_service_node_ip

name

openstack_provision_status

out_network_id

out_service_node_ip

result

router_id

router_net_id

router_subnet_id

rules

tenant_id

class networking_cisco.apps.saf.db.dfa_db_models.**DfaInServiceSubnet**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents DFA Service Subnet.

allocated

network_id

subnet_address

subnet_id

class networking_cisco.apps.saf.db.dfa_db_models.**DfaInSubnet**
Bases: *networking_cisco.apps.saf.db.dfa_db_models.DfaResource*

dfa_in_subnet_init = 0

get_model()

classmethod init_done()

is_init_done()

class networking_cisco.apps.saf.db.dfa_db_models.**DfaNetwork**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents DFA network.

config_profile

fwd_mod

name

network_id

result

segmentation_id

source

tenant_id

vlan

class networking_cisco.apps.saf.db.dfa_db_models.**DfaOutServiceSubnet**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents DFA Service Subnet.

allocated

network_id

subnet_address

subnet_id

class networking_cisco.apps.saf.db.dfa_db_models.**DfaOutSubnet**
Bases: *networking_cisco.apps.saf.db.dfa_db_models.DfaResource*

dfa_out_subnet_init = 0

get_model()

classmethod init_done()

is_init_done()

class networking_cisco.apps.saf.db.dfa_db_models.**DfaResource**
Bases: object

is_res_init_done(*num_init*)

class networking_cisco.apps.saf.db.dfa_db_models.**DfaSegment**
Bases: *networking_cisco.apps.saf.db.dfa_db_models.DfaResource*

dfa_segment_init = 0

get_model()

classmethod init_done()

is_init_done()

class networking_cisco.apps.saf.db.dfa_db_models.**DfaSegmentTypeDriver**(*segid_min*, *segid_max*, *res_name*, *cfg*, *reuse_timeout=0*)
Bases: object

allocate_segmentation_id(*net_id*, *seg_id=None*, *source=None*)

get_all_seg_netid()

get_seg_netid_src(*source*)

get_segid_allocation(*session*, *seg_id*)

release_segmentation_id(*seg_id*)

class networking_cisco.apps.saf.db.dfa_db_models.**DfaSegmentationId**(**kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents DFA segmentation ID.

allocated

delete_time

network_id

segmentation_id

source

class networking_cisco.apps.saf.db.dfa_db_models.**DfaTenants**(**kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents DFA tenants.

dci_id

id

name

result

class networking_cisco.apps.saf.db.dfa_db_models.**DfaTopologyDb**(**kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents DFA Topology Discovery.

configurations

created

heartbeat

host

phy_interface

protocol_interface

remote_chassis_id_mac

remote_evb_cfgd

remote_evb_mode

remote_mgmt_addr

remote_port

remote_port_id_mac

remote_system_desc

remote_system_name

class networking_cisco.apps.saf.db.dfa_db_models.**DfaVlan**
Bases: *networking_cisco.apps.saf.db.dfa_db_models.DfaResource*

dfa_vlan_init = 0

get_model()

classmethod init_done()

is_init_done()

class networking_cisco.apps.saf.db.dfa_db_models.**DfaVlanId**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents DFA VLAN ID.

allocated

delete_time

network_id

segmentation_id

source

class networking_cisco.apps.saf.db.dfa_db_models.**DfaVmInfo**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents VM info.

fwd_mod

gw_mac

host

instance_id

ip

mac

name

network_id

port_id

result

segmentation_id

status

class networking_cisco.apps.saf.db.dfa_db_models.**DfasubnetDriver**(*subnet_min_str*, *subnet_max_str*, *res_name*)
Bases: object

allocate_subnet(*subnet_lst*, *net_id=None*)
Allocate subnet from pool.

Return allocated db object or None.

get_subnet(*sub*)

get_subnet_by_netid(*netid*)

release_subnet(*subnet_address*)

release_subnet_by_netid(*netid*)

release_subnet_no_netid()

update_subnet(*subnet*, *net_id*, *subnet_id*)

class networking_cisco.apps.saf.db.dfa_db_models.**TopologyDiscoveryDb**(*cfg*)
Bases: object

Topology Discovery Database API.

add_update_topology_db(***params*)
Add or update an entry to the topology DB.

delete_topology_entry(***req*)
Delete the entries from the topology DB.

query_topology_db(*dict_convert=False*, ***req*)
Query an entry to the topology DB.

Module contents

networking_cisco.apps.saf.server package

Subpackages

networking_cisco.apps.saf.server.services package

Subpackages

networking_cisco.apps.saf.server.services.firewall package

Subpackages

networking_cisco.apps.saf.server.services.firewall.native package

Subpackages

networking_cisco.apps.saf.server.services.firewall.native.drivers package

Submodules

networking_cisco.apps.saf.server.services.firewall.native.drivers.asa_rest module

class networking_cisco.apps.saf.server.services.firewall.native.drivers.asa_rest.**Asa5585**(*m*
us
na
pa
wa

Bases: object

ASA 5585 Driver.

apply_policy(*policy*)
Apply a firewall policy.

build_acl(*tenant_name*, *rule*)
Build the ACL.

build_acl_ip(*network_obj*)
Build the acl for IP address.

build_acl_port(*port*, *enabled=True*)
Build the acl for L4 Ports.

cleanup(***kwargs*)
cleanup ASA context for an edge tenant pair.

get_ip_address(*ip_address*)
Decode the IP address.

get_quota()

rest_send_cli(*data*)

setup(***kwargs*)
setup ASA context for an edge tenant pair.

networking_cisco.apps.saf.server.services.firewall.native.drivers.base module

class networking_cisco.apps.saf.server.services.firewall.native.drivers.base.**BaseDriver**
Bases: object

Base Driver class for FW driver classes.

create_fw(*tenant_id*, *data*)
Create the Firewall.

delete_fw(*tenant_id*, *data*)
Delete the Firewall.

get_max_quota()
Retrieves the maximum number of FW that could be created.

get_name()
Return the name of the driver service.

initialize()
Initialize method.

is_device_virtual()
Return False if device is physical, True otherwise.

modify_fw(*tenant_id*, *data*)
Modify the Firewall.

network_create_notif(*tenant_id*, *tenant_name*, *cidr*)
Network Create Notification.

network_delete_notif(*tenant_id*, *tenant_name*, *net_id*)
Network Delete Notification.

populate_dcnm_obj()
Populate DCNM Obj.

populate_event_que()
Populate Event Queue.

networking_cisco.apps.saf.server.services.firewall.native.drivers.dev_mgr module

class networking_cisco.apps.saf.server.services.firewall.native.drivers.dev_mgr.**DeviceMgr**(*c*
Bases: object

Device Manager.

create_fw_device(*tenant_id*, *fw_id*, *data*)
Creates the Firewall.

delete_fw_device(*tenant_id*, *fw_id*, *data*)
Deletes the Firewall.

drvr_initialize(*cfg*)
Initialize the driver routines.

is_device_virtual()
Returns if the device is physical or virtual.

modify_fw_device(*tenant_id*, *fw_id*, *data*)
Modifies the firewall cfg.

network_create_notif(*tenant_id*, *tenant_name*, *cidr*)
Notification for Network create.

Since FW ID not present, it's not possible to know which FW instance to call. So, calling everyone, each instance will figure out if it applies to them.

network_delete_notif(*tenant_id*, *tenant_name*, *net_id*)
Notification for Network delete.

Since FW ID not present, it's not possible to know which FW instance to call. So, calling everyone, each instance will figure out if it applies to them.

populate_dcnm_obj(*dcnm_obj*)
Populates the DCNM object.

populate_event_que(*que_obj*)
Populates the event queue object.

This is for sending router events to event handler.

populate_local_sch_cache(*fw_dict*)
Populate the local cache from FW DB after restart.

class networking_cisco.apps.saf.server.services.firewall.native.drivers.dev_mgr.**MaxSched**(*ob*
Bases: object

Max Sched class.

This scheduler will return the first firewall until it reaches its quota.

allocate_fw_dev(*fw_id*)
Allocate firewall device.

Allocate the first Firewall device which has resources available.

deallocate_fw_dev(*fw_id*)
Release the firewall resource.

get_fw_dev_map(*fw_id*)
Return the object dict and mgmt ip for a firewall.

populate_fw_dev(*fw_id*, *mgmt_ip*, *new*)
Populate the class after a restart.

networking_cisco.apps.saf.server.services.firewall.native.drivers.dev_mgr_plug module

class networking_cisco.apps.saf.server.services.firewall.native.drivers.dev_mgr_plug.**Device**

Bases: stevedore.named.NamedExtensionManager

Device Manager that loads the firewall drivers

get_drvr_obj()
Return the dynamically loaded object

networking_cisco.apps.saf.server.services.firewall.native.drivers.native module

class networking_cisco.apps.saf.server.services.firewall.native.drivers.native.**NativeFirewa**

Bases: *networking_cisco.apps.saf.server.services.firewall.native.drivers.base.BaseDriver*, *networking_cisco.apps.saf.server.services.firewall.native.fabric_setup_base.FabricApi*

Native Firewall Driver.

attach_intf_router(*tenant_id*, *tenant_name*, *router_id*)
Routine to attach the interface to the router.

create_fw(*tenant_id*, *data*)
Top level routine called when a FW is created.

create_tenant_dict(*tenant_id*, *router_id=None*)
Tenant dict creation.

delete_fw(*tenant_id*, *data*)
Top level routine called when a FW is deleted.

delete_intf_router(*tenant_id*, *tenant_name*, *router_id*)
Routine to delete the router.

get_max_quota()
Returns the number of Firewall instances.

Returns the maximum number of Firewall instance that a single Firewall can support.

get_name()
Returns the name of the FW appliance.

get_router_id(*tenant_id*, *tenant_name*)
Retrieve the router ID.

initialize(*cfg_dict*)
Initialization routine.

is_device_virtual()
Returns if device is virtual.

modify_fw(*tenant_id*, *data*)
Modify Firewall attributes.

Routine called when Firewall attributes gets modified. Nothing to be done for native FW.

network_create_notif(*tenant_id*, *tenant_name*, *cidr*)
Tenant Network create Notification.

Restart is not supported currently for this. fixme(padkrish).

network_delete_notif(*tenant_id*, *tenant_name*, *network_id*)
Tenant Network delete Notification.

Restart is not supported currently for this. fixme(padkrish).

populate_dcnm_obj(*dcnm_obj*)
Populate the DCNM object.

populate_event_que(*que_obj*)
Populate the event queue object.

prepare_router_vm_msg(*tenant_id*, *tenant_name*, *router_id*, *net_id*, *subnet_id*, *seg*, *status*)
Prepare the message to be sent to Event queue for VDP trigger.

This is actually called for a subnet add to a router. This function prepares a VM's VNIC create/delete message.

program_default_gw(*tenant_id*, *arg_dict*)
Program the default gateway to the 'out' interface.

program_next_hop(*tenant_id*, *arg_dict*)
Program the next hop for all host subnets to the 'in' gateway.

send_in_router_port_msg(*tenant_id*, *arg_dict*, *status*)
Call routine to send vNic create notification for 'in' interface.

send_out_router_port_msg(*tenant_id*, *arg_dict*, *status*)
Call routine to send vNic create notification for 'out' interface.

send_router_port_msg(*tenant_id*, *tenant_name*, *router_id*, *net_id*, *subnet_id*, *seg*, *status*)
Sends the router port message to the queue.

update_dcnm_partition_static_route(*tenant_id*, *arg_dict*)
Add static route in DCNM's partition.

This gets pushed to the relevant leaf switches.

networking_cisco.apps.saf.server.services.firewall.native.drivers.phy_asa module

class networking_cisco.apps.saf.server.services.firewall.native.drivers.phy_asa.**PhyAsa**
Bases: *networking_cisco.apps.saf.server.services.firewall.native.drivers.base.BaseDriver*, *networking_cisco.apps.saf.server.services.firewall.native.fabric_setup_base.FabricApi*

Physical ASA Driver.

create_fw(*tenant_id*, *data*)

delete_fw(*tenant_id*, *data*)

get_max_quota()

get_name()

initialize(*cfg_dict*)

is_device_virtual()

modify_fw(*tenant_id*, *data*)

network_create_notif(*tenant_id*, *tenant_name*, *cidr*)
Network Create Notification.

network_delete_notif(*tenant_id*, *tenant_name*, *network_id*)
Network Delete Notification.

populate_dcnm_obj(*dcnm_obj*)

populate_event_que(*que_obj*)

Module contents

Submodules

networking_cisco.apps.saf.server.services.firewall.native.fabric_setup_base module

class networking_cisco.apps.saf.server.services.firewall.native.fabric_setup_base.**FabricAp**
Bases: object

Class for retrieving FW attributes, available to external modules.

classmethod del_obj(*tenant_id*, *obj*)
Delete the tenant obj.

classmethod get_dummy_router_net(*tenant_id*)
Retrieves the dummy router network info.

classmethod get_in_ip_addr(*tenant_id*)
Retrieves the 'in' service subnet attributes.

classmethod get_in_net_id(*tenant_id*)
Retrieve the network ID of IN network.

classmethod get_in_seg_vlan(*tenant_id*)
Retrieves the IN Seg, VLAN, mob domain.

classmethod get_in_srvc_node_ip_addr(*tenant_id*)
Retrieves the IN service node IP address.

classmethod get_in_subnet_id(*tenant_id*)
Retrieve the subnet ID of IN network.

classmethod get_out_ip_addr(*tenant_id*)
Retrieves the 'out' service subnet attributes.

classmethod get_out_net_id(*tenant_id*)
Retrieve the network ID of OUT network.

classmethod get_out_seg_vlan(*tenant_id*)
Retrieves the OUT Seg, VLAN, mob domain.

classmethod get_out_srvc_node_ip_addr(*tenant_id*)
Retrieves the OUT service node IP address.

classmethod get_out_subnet_id(*tenant_id*)
Retrieve the subnet ID of OUT network.

ip_db_obj = {}

classmethod is_network_source_fw(*nwk*, *nwk_name*)
Check if SOURCE is FIREWALL, if yes return TRUE.

If source is None or entry not in NWK DB, check from Name. Name should have constant AND length should match.

is_subnet_source_fw(*tenant_id*, *subnet*)
Check if the subnet is created as a result of any FW operation.

serv_obj_dict = {}

classmethod store_db_obj(*in_obj*, *out_obj*)
Store the IP DB object.

classmethod store_tenant_obj(*tenant_id*, *obj*)
Store the tenant obj.

class networking_cisco.apps.saf.server.services.firewall.native.fabric_setup_base.**FabricBas**
Bases: *networking_cisco.apps.saf.db.dfa_db_models.DfaDBMixin*, *networking_cisco.apps.saf.server.services.firewall.native.fabric_setup_base.FabricApi*

Class to implement Fabric configuration for Physical FW.

alloc_retrieve_subnet_info(*tenant_id*, *direc*)
Allocate and store Subnet.

This function initially checks if subnet is allocated for a tenant for the in/out direction. If not, it calls routine to allocate a subnet and stores it on tenant object.

alloc_seg(*net_id*)
Allocates the segmentation ID.

alloc_vlan(*net_id*)
Allocates the vlan ID.

allocate_seg_vlan(*net_id*, *is_fw_virt*, *direc*, *tenant_id*)
allocate segmentation ID and VLAN ID.

Allocate vlan, seg thereby storing NetID atomically. This saves an extra step to update DB with NetID after allocation. Also may save an extra step after restart, if process crashed after allocation but before updating DB with NetID. Now, since both steps are combined, Vlan/Seg won't be allocated w/o NetID.

check_allocate_ip(*obj*, *direc*)
This function allocates a subnet from the pool.

It first checks to see if Openstack is already using the subnet. If yes, it retries until it finds a free subnet not used by Openstack.

clear_dcnm_in_part(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Clear the DCNM in partition service information.

Clear the In partition service node IP address in DCNM and update the result.

clear_dcnm_out_part(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Clear DCNM out partition information.

Clear the DCNM OUT partition service node IP address and update the result

correct_db_restart()
Ensure DB is consistent after unexpected restarts.

create_dcnm_in_nwk(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Create the DCNM In Network and store the result in DB.

create_dcnm_out_nwk(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Create the DCNM OUT Network and update the result.

create_dcnm_out_part(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Create the DCNM OUT partition and update the result.

create_openstack_network(*subnet*, *network*, *tenant_id*, *tenant_name*, *direction*)
Helper function to create openstack network.

The net_id and subnet_id is returned. Upon failure, the subnet is deallocated.

create_os_dummy_rtr(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Create the dummy interface and attach it to router.

Attach the dummy interface to the Openstack router and store the info in DB.

create_os_in_nwk(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Create the Openstack IN network and stores the values in DB.

create_os_out_nwk(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Create the Openstack OUT network and stores the values in DB.

create_serv_obj(*tenant_id*)
Creates and stores the service object associated with a tenant.

delete_dcnm_in_nwk(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Delete the DCNM In Network and store the result in DB.

delete_dcnm_out_nwk(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Delete the DCNM OUT network and update the result.

delete_dcnm_out_part(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Delete the DCNM OUT partition and update the result.

delete_fabric_fw(*tenant_id*, *fw_dict*, *is_fw_virt*, *result*)
Top level routine to unconfigure the fabric.

delete_fabric_fw_internal(*tenant_id*, *fw_dict*, *is_fw_virt*, *result*)
Internal routine to delete the fabric configuration.

This runs the SM and deletes the entries from DB and local cache.

delete_os_dummy_rtr(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Delete the Openstack Dummy router and store the info in DB.

delete_os_dummy_rtr_nwk(*rtr_id*, *net_id*, *subnet_id*)
Delete the dummy interface to the router.

delete_os_in_nwk(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Deletes the Openstack In network and update the DB.

delete_os_nwk_db(*net_id*, *seg*, *vlan*)
Delete the Openstack Network from the database.

Release the segmentation ID, VLAN associated with the net. Delete the network given the partial name. Delete the entry from Network DB, given the net ID. Delete the entry from Firewall DB, given the net ID. Release the IN/OUT sug=bnets associated with the net.

delete_os_out_nwk(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Deletes the Openstack Out network and update the DB.

delete_serv_obj(*tenant_id*)
Creates and stores the service object associated with a tenant.

fill_dcnm_net_info(*tenant_id*, *direc*, *vlan_id=0*, *segmentation_id=0*)
Fill DCNM network parameters.

Function that fills the network parameters for a tenant required by DCNM.

fill_dcnm_subnet_info(*tenant_id*, *subnet*, *start*, *end*, *gateway*, *sec_gateway*, *direc*)
Fills the DCNM subnet parameters.

Function that fills the subnet parameters for a tenant required by DCNM.

get_dummy_router_net(*tenant_id*)
Retrieves the dummy router information from service object.

get_end_ip(*subnet*)
Returns the end IP associated with a subnet.

It's value is the second last address of the CIDR. i.e.. end IP address is -2 (from the end of the subnet)

get_gateway(*subnet*)
Retrieve the Gateway associated with the subnet.

Returns the Gateway associated with a subnet. This is also the Gateway address configured in the leaf switch or ToR. This value is subnet + 1. The Gateway created in Openstack is + 2, which is also the FW's IN interface address and DCNM's service node IP address.

get_key_state(*status*, *state_dict*)
Returns the key associated with the dict.

get_next_create_state(*state*, *ret*)
Return the next create state from previous state.

get_next_del_state(*state*, *ret*)
Return the next delete state from previous state.

get_next_ip(*tenant_id*, *direc*)
Retrieve the next available subnet.

Given a tenant, it returns the service subnet values assigned to it based on direction.

get_next_state(*state*, *ret*, *oper*)
Returns the next state for a create or delete operation.

get_secondary_gateway(*subnet*)
Returns the Secondary Gateway associated with a subnet.

This is the end IP address. The secondary GW IP address reuses the end IP address.

get_service_obj(*tenant_id*)
Retrieves the service object associated with a tenant.

get_start_ip(*subnet*)
Returns the starting IP associated with a subnet.

This value is start IP address is + 3.

init_state(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Dummy function called at the init stage.

initialize_create_state_map()
This is a mapping of create result message string to state.

initialize_delete_state_map()
This is a mapping of delete result message string to state.

initialize_fsm()
Initializing the Finite State Machine.

This is a mapping of state to a dict of appropriate create and delete functions.

pop_fw_local(*tenant_id*, *net_id*, *direc*, *node_ip*)
Populate the local cache.

Read the Network DB and populate the local cache. Read the subnet from the Subnet DB, given the net_id and populate the cache.

pop_fw_state(*compl_result*, *os_status*, *dcnm_status*)
Populate the state information in the cache.

Check if state information is embedded in result If not: a. It's still in Init state and no SM is called yet b. The SM has completely run c. Delete has started and before any SM is run, it restarted.

populate_local_cache()
Populate the local cache from DB.

Read the entries from FW DB and Calls routines to populate the cache.

populate_local_cache_tenant(*fw_id*, *fw_data*)
Populate the cache for a given tenant.

Calls routines to Populate the in and out information. Update the result information. Populate the state information. Populate the router information.

prepare_fabric_done(*tenant_id*, *tenant_name*, *is_fw_virt=False*)
Dummy function called at the final stage.

prepare_fabric_fw(*tenant_id*, *fw_dict*, *is_fw_virt*, *result*)
Top level routine to prepare the fabric.

release_subnet(*cidr*, *direc*)
Routine to release a subnet from the DB.

retrieve_dcnm_net_info(*tenant_id*, *direc*)
Retrieves the DCNM network info for a tenant.

retrieve_dcnm_subnet_info(*tenant_id*, *direc*)
Retrieves the DCNM subnet info for a tenant.

retrieve_network_info(*tenant_id*, *direc*)
Retrieve the DCNM Network information.

Retrieves DCNM net dict if already filled, else, it calls routines to fill the net info and store it in tenant obj.

retry_failure(*tenant_id*, *tenant_name*, *fw_data*, *is_fw_virt*, *result*)
Top level retry failure routine.

retry_failure_internal(*tenant_id*, *tenant_name*, *fw_data*, *is_fw_virt*, *result*)
Internal routine to retry the failed cases.

run_create_sm(*tenant_id*, *fw_dict*, *is_fw_virt*)
Runs the create State Machine.

Goes through every state function until the end or when one state returns failure.

run_delete_sm(*tenant_id*, *fw_dict*, *is_fw_virt*)
Runs the delete State Machine.

Goes through every state function until the end or when one state returns failure.

store_dcnm(*dcnm_obj*)
Stores the DCNM object.

store_fw_db(*tenant_id*, *net*, *subnet_dict*, *direc*)
Calls the service object routine to commit the FW entry to DB.

store_fw_db_router(*tenant_id*, *net_id*, *subnet_id*, *router_id*, *os_status*)
Store the result of FW router operation in DB.

Calls the service object routine to commit the result of router operation in to DB, after updating the local cache.

store_net_db(*tenant_id*, *net*, *net_dict*, *result*)
Store service network in DB.

store_net_fw_db(*tenant_id*, *net*, *net_dict*, *subnet_dict*, *direc*, *result*, *os_status=None*, *dcnm_status=None*, *dev_status=None*)
Save the entries in Network and Firewall DB.

Stores the entries into Network DB and Firewall DB as well as update the result of operation into FWDB. Generally called by OS operations that wants to modify both the Net DB and FW DB.

update_dcnm_in_part(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Update DCNM's in partition information.

Update the In partition service node IP address in DCNM and update the result

update_dcnm_net_info(*tenant_id*, *direc*, *vlan_id*, *segmentation_id*)
Update the DCNM net info with allocated values of seg/vlan.

update_dcnm_out_part(*tenant_id*, *fw_dict*, *is_fw_virt=False*)
Update DCNM OUT partition service node IP address and result.

update_fw_db_result(*tenant_id*, *os_status=None*, *dcnm_status=None*, *dev_status=None*)
Update the FW DB Result and commit it in DB.

Calls the service object routine to commit the result of a FW operation in to DB

update_net_info(*tenant_id*, *direc*, *vlan_id*, *segmentation_id*)
Update the DCNM netinfo with vlan and segmentation ID.

update_subnet_db_info(*tenant_id*, *direc*, *net_id*, *subnet_id*)
Update the subnet DB with Net and Subnet ID, given the subnet.

class networking_cisco.apps.saf.server.services.firewall.native.fabric_setup_base.**ServiceIp**
Bases: *networking_cisco.apps.saf.db.dfa_db_models.DfaDBMixin*

Tenant Specific Service Attributes.

Class for storing/retrieving the tenant specific service attributes locally as well as from the FW DB

commit_fw_db()
Calls routine to update the FW DB.

commit_fw_db_result()
Calls routine to update the FW create/delete result in DB.

create_fw_db(*fw_id*, *fw_name*, *tenant_id*)
Create FW dict.

destroy_local_fw_db()
Delete the FW dict and its attributes.

fixup_state(*from_str*, *state*)
Fixup state after retart.

Fixup the state, if Delete is called when create SM is half-way through.

get_dcnm_net_dict(*direc*)
Retrieve the DCNM net dict.

get_dcnm_subnet_dict(*direc*)
Retrieve the DCNM subnet dict.

get_dummy_router_net()
Retrieve the dummy router attributes.

get_fw_dict()
retrieving the fw dict.

get_in_ip_addr()
Retrieve 'in' service subnet attributes.

get_in_seg_vlan()
Retrieve the seg, vlan, mod domain for IN network.

get_local_final_result()
Retrieve the final reult for FW create/delete.

get_out_ip_addr()
Retrieve 'out' service subnet attributes.

get_out_seg_vlan()
Retrieve the seg, vlan, mod domain for OUT network.

get_state()
Return the current state.

get_store_local_final_result()
Store/Retrieve the final result.

Retrieve the final result for FW create/delete from DB and store it locally.

is_fabric_create()
Retrieve the fabric create status.

set_fabric_create(*status*)
Store the fabric create status.

store_dcnm_net_dict(*net_dict*, *direc*)
Storing the DCNM net dict.

store_dcnm_subnet_dict(*subnet_dict*, *direc*)
Store the subnet attributes and dict.

store_dummy_router_net(*net_id*, *subnet_id*, *rtr_id*)
Storing the router attributes.

store_local_final_result(*final_res*)
Store the final reult for FW create/delete.

store_state(*state*, *popl_db=True*)
Store the state of FW create/del operation.

update_fw_dict(*fw_dict*)
updating the fw dict.

update_fw_local_cache(*net*, *direc*, *start*)
Update the fw dict with Net ID and service IP.

update_fw_local_result(*os_result=None*, *dcnm_result=None*, *dev_result=None*)
Retrieve and update the FW result in the dict.

update_fw_local_result_str(*os_result=None*, *dcnm_result=None*, *dev_result=None*)
Update the FW result in the dict.

update_fw_local_router(*net_id*, *subnet_id*, *router_id*, *os_result*)
Update the FW with router attributes.

networking_cisco.apps.saf.server.services.firewall.native.fw_constants module

networking_cisco.apps.saf.server.services.firewall.native.fw_mgr module

class networking_cisco.apps.saf.server.services.firewall.native.fw_mgr.**FwMapAttr**(*tenant_id*)
Bases: object

Firewall Attributes. This class is instantiated per tenant.

create_fw(*proj_name*, *pol_id*, *fw_id*, *fw_name*, *fw_type*, *rtr_id*)
Fills up the local attributes when FW is created.

delete_fw(*fw_id*)
Deletes the FW local attributes.

delete_policy(*pol_id*)
Deletes the policy from the local dictionary.

delete_rule(*rule_id*)
Delete the specific Rule from dictionary indexed by rule id.

fw_drvr_created(*status*)
This stores the status of the driver init.

This API assumes only one FW driver.

get_fw_dict()
This API creates a FW dictionary from the local attributes.

is_fw_complete()
This API returns the completion status of FW.

This returns True if a FW is created with a active policy that has more than one rule associated with it and if a driver init is done successfully.

is_fw_drvr_create_needed()
This API returns True if a driver init needs to be performed.

This returns True if a FW is created with a active policy that has more than one rule associated with it and if a driver init is NOT done.

is_fw_drvr_created()
This returns the status of the driver creation.

This API assumes only one FW driver.

is_fw_present(*fw_id*)
Returns if firewall index by ID is present in dictionary.

is_policy_present(*pol_id*)
Returns if policy specified by ID is present in the dictionary.

is_rule_present(*rule_id*)
Returns if rule specified by rule id is present in dictionary.

one_rule_present(*pol_id*)
Returns if atleast one rule is present in the policy.

rule_update(*rule_id*, *rule*)
Update the rule.

store_policy(*pol_id*, *policy*)
Store the policy.

Policy is maintained as a dictionary of pol ID.

store_rule(*rule_id*, *rule*)
Store the rules.

Policy is maintained as a dictionary of Rule ID.

update_fw_params(*rtr_id=-1*, *fw_type=-1*)
Updates the FW parameters.

class networking_cisco.apps.saf.server.services.firewall.native.fw_mgr.**FwMgr**(*cfg*)
Bases: *networking_cisco.apps.saf.server.services.firewall.native.drivers.dev_mgr.DeviceMgr*

Firewall Native Manager

convert_fwdb(*tenant_id*, *name*, *policy_id*, *fw_id*)
Convert the Firewall DB to a query response.

From FWDB inputs, this will create a FW message that resembles the actual data from Openstack, when a query for FW is done.

convert_fwdb_event_msg(*rule*, *tenant_id*, *rule_id*, *policy_id*)
Convert the Firewall DB to a event message format.

From inputs from DB, this will create a FW rule dictionary that resembles the actual data from Openstack when a rule is created. This is usually called after restart, in order to populate local cache.

fill_fw_dict_from_db(*fw_data*)
This routine is called to create a local fw_dict with data from DB.

fw_create(*data*, *fw_name=None*, *cache=False*)
Top level FW create function.

fw_delete(*data*, *fw_name=None*)
Top level FW delete function.

fw_handler_decorator(*fw_func*)

fw_policy_create(*data*, *fw_name=None*, *cache=False*)
Top level policy create routine.

fw_policy_delete(*data*, *fw_name=None*)
Top level policy delete routine.

fw_retry_failures()
Top level retry routine called.

fw_retry_failures_create()
This module is called for retrying the create cases.

fw_retry_failures_delete()
This routine is called for retrying the delete cases.

fw_rule_create(*data*, *fw_name=None*, *cache=False*)
Top level rule creation routine.

fw_rule_delete(*data*, *fw_name=None*)
Top level rule delete function.

fw_rule_update(*data*, *fw_name=None*)
Top level rule update routine.

fw_update(*data*, *fw_name=None*)
Top level FW update function.

network_del_notif(*tenant_id*, *tenant_name*, *net_id*)
Network delete notification.

network_sub_create_notif(*tenant_id*, *tenant_name*, *cidr*)
Network create notification.

populate_cfg_dcnm(*cfg*, *dcnm_obj*)
This routine stores the DCNM object.

populate_event_queue(*cfg*, *que_obj*)
This routine is for storing the Event Queue obj.

populate_local_cache()
This populates the local cache after reading the Database.

It calls the appropriate rule create, fw create routines. It doesn't actually call the routine to prepare the fabric or cfg the device since it will be handled by retry module.

project_create_notif(*tenant_id*, *tenant_name*)
Tenant Create notification.

project_delete_notif(*tenant_id*, *tenant_name*)
Tenant Delete notification.

retry_failure_fab_dev_create(*tenant_id*, *fw_data*, *fw_dict*)
This module calls routine in fabric to retry the failure cases.

If device is not successfully cfg/uncfg, it calls the device manager routine to cfg/uncfg the device.

retry_failure_fab_dev_delete(*tenant_id*, *fw_data*, *fw_dict*)
Retry the failure cases for delete.

This module calls routine in fabric to retry the failure cases for delete. If device is not successfully cfg/uncfg, it calls the device manager routine to cfg/uncfg the device.

class networking_cisco.apps.saf.server.services.firewall.native.fw_mgr.**FwTenant**
Bases: object

Class to hold Tenant specific attributes.

This class maintains a mapping of rule, policies, FW IDs with its associated tenant ID. It's assumed that a Rule, policy or FW signified by their unique ID can only be associated with one Tenant.

del_fw_tenant(*fw_id*)
Deletes the FW Tenant mapping.

del_policy_tenant(*pol_id*)
Deletes the Tenant policy mapping.

del_rule_tenant(*rule_id*)
Deletes the Tenant policy mapping.

get_fw_tenant(*fw_id*)
Retrieves the tenant ID corresponding to the firewall.

get_policy_tenant(*policy_id*)
Retrieves the tenant ID corresponding to the policy.

get_rule_tenant(*rule_id*)
Retrieves the tenant ID corresponding to the rule.

store_fw_tenant(*fw_id*, *tenant_id*)
Stores the tenant ID corresponding to the firewall.

store_policy_tenant(*policy_id*, *tenant_id*)
Stores the tenant ID corresponding to the policy.

store_rule_tenant(*rule_id*, *tenant_id*)
Stores the tenant ID corresponding to the rule.

Module contents

Module contents

Submodules

networking_cisco.apps.saf.server.services.constants module

Module contents

Submodules

networking_cisco.apps.saf.server.cisco_dfa_rest module

This module provides APIs for communicating with DCNM.

class networking_cisco.apps.saf.server.cisco_dfa_rest.**DFARESTClient**(*cfg*)
Bases: object

DFA client class that provides APIs to interact with DCNM.

config_profile_fwding_mode_get(*profile_name*)
Return forwarding mode of given config profile.

config_profile_list()
Return config profile list from DCNM.

create_network(*tenant_name*, *network*, *subnet*, *dhcp_range=True*)
Create network on the DCNM.

Parameters

- **tenant_name** – name of tenant the network belongs to
- **network** – network parameters
- **subnet** – subnet parameters of the network

create_partition(*org_name*, *part_name*, *dci_id*, *vrf_prof*, *service_node_ip=None*, *desc=None*)
Create partition on the DCNM.

Parameters

- **org_name** – name of organization to be created
- **part_name** – name of partition to be created

- **dci_id** – DCI ID
- **service_node_ip** – Specifies the Default route IP address.
- **desc** – string that describes organization

Vrf_prof VRF profile for the partition

create_project(*orch_id*, *org_name*, *part_name*, *dci_id*, *desc=None*)
Create project on the DCNM.

Parameters

- **orch_id** – orchestrator ID
- **org_name** – name of organization.
- **part_name** – name of partition.
- **dci_id** – Data Center interconnect id.
- **desc** – description of project.

create_service_network(*tenant_name*, *network*, *subnet*, *dhcp_range=True*)
Create network on the DCNM.

Parameters

- **tenant_name** – name of tenant the network belongs to
- **network** – network parameters
- **subnet** – subnet parameters of the network

delete_network(*tenant_name*, *network*)
Delete network on the DCNM.

Parameters

- **tenant_name** – name of tenant the network belongs to
- **network** – object that contains network parameters

delete_partition(*org_name*, *partition_name*)
Send partition delete request to DCNM.

Parameters **partition_name** – name of partition to be deleted

delete_project(*tenant_name*, *part_name*)
Delete project on the DCNM.

Parameters

- **tenant_name** – name of project.
- **part_name** – name of partition.

delete_service_network(*tenant_name*, *network*)
Delete service network on the DCNM.

Parameters

- **tenant_name** – name of tenant the network belongs to
- **network** – object that contains network parameters

fill_urls()
This assigns the URL's based on the protocol.

get_config_profile_for_network(*net_name*)
Get the list of profiles.

get_dcnm_protocol()
Routine to find out if DCNM is using http or https.

DCNM 10 (Fuji-4) and above does not support http. Only https is supported and enabled by default. Prior DCNM versions supported both http and https. But, only http was enabled by default. So, enabler needs to find out if DCNM is supporting http or https to be friendly with the existing installed setups.

get_network(*org*, *segid*)
Return given network from DCNM.

Parameters

- **org** – name of organization.
- **segid** – segmentation id of the network.

get_partition_dciId(*org_name*, *part_name*, *part_info=None*)
get DCI ID for the partition.

Parameters

- **org_name** – name of organization
- **part_name** – name of partition

get_partition_segmentId(*org_name*, *part_name*, *part_info=None*)
get partition Segment ID from the DCNM.

Parameters

- **org_name** – name of organization
- **part_name** – name of partition

get_partition_serviceNodeIp(*org_name*, *part_name*, *part_info=None*)
get Service Node IP address from the DCNM.

Parameters

- **org_name** – name of organization
- **part_name** – name of partition

get_partition_vrfProf(*org_name*, *part_name=None*, *part_info=None*)
get VRF Profile for the partition from the DCNM.

Parameters

- **org_name** – name of organization
- **part_name** – name of partition

get_segmentid_range(*orchestrator_id*)
Get segment id range from DCNM.

get_version()
Get the DCNM version.

list_networks(*org*, *part*)
Return list of networks from DCNM.

Parameters

- **org** – name of organization.

- **part** – name of partition.

list_organizations()
Return list of organizations from DCNM.

set_segmentid_range(*orchestrator_id*, *segid_min*, *segid_max*)
set segment id range in DCNM.

update_partition_static_route(*org_name*, *part_name*, *static_ip_list*, *vrf_prof=None*, *service_node_ip=None*)
Send static route update requests to DCNM.

Parameters

- **org_name** – name of organization
- **part_name** – name of partition

Static_ip_list List of static IP addresses

Vrf_prof VRF Profile

Service_node_ip Service Node IP address

update_project(*org_name*, *part_name*, *dci_id=-1*, *service_node_ip='0.0.0.0'*, *vrf_prof=None*, *desc=None*)
Update project on the DCNM.

Parameters

- **org_name** – name of organization.
- **part_name** – name of partition.
- **dci_id** – Data Center interconnect id.
- **desc** – description of project.

update_segmentid_range(*orchestrator_id*, *segid_min*, *segid_max*)
update segment id range in DCNM.

networking_cisco.apps.saf.server.dfa_events_handler module

class networking_cisco.apps.saf.server.dfa_events_handler.**EventsHandler**(*ser_name*, *pqueue*, *c_pri*, *d_pri*)
Bases: object

This class defines methods to listen and process events.

callback(*timestamp*, *event_type*, *payload*)
Callback method for processing events in notification queue.

Parameters

- **timestamp** – time the message is received.
- **event_type** – event type in the notification queue such as identity.project.created, identity.project.deleted.
- **payload** – Contains information of an event

create_rpc_client(*thishost*)

event_handler()
Wait on queue for listening to the events.

nclient

send_msg_to_agent(*thishost*, *msg_type*, *msg*)

send_vm_info(*thishost*, *msg*)

start()

update_ip_rule(*thishost*, *msg*)

wait()

networking_cisco.apps.saf.server.dfa_fail_recovery module

class networking_cisco.apps.saf.server.dfa_fail_recovery.**DfaFailureRecovery**(*cfg*)
Bases: object

Failure recovery class.

add_events(***kwargs*)
Add failure event into the queue.

cfg

failure_recovery(*fail_info*)
Failure recovery task.

In case of failure in projects, network and VM create/delete, this task goes through all failure cases and try the request.

networking_cisco.apps.saf.server.dfa_instance_api module

This file provides a wrapper to novaclient API, for getting the instacne's information such as display_name.

class networking_cisco.apps.saf.server.dfa_instance_api.**DFAInstanceAPI**
Bases: object

This class provides API to get information for a given instance.

get_instance_for_uuid(*uuid*, *project_id*)
Return instance name for given uuid of an instance and project.

Uuid Instance's UUID

Project_id UUID of project (tenant)

networking_cisco.apps.saf.server.dfa_listen_dcnm module

class networking_cisco.apps.saf.server.dfa_listen_dcnm.**DCNMListener**(*name*, *ip*, *user*, *password*, *pqueue=None*, *c_pri=100*, *d_pri=100*)
Bases: object

This AMQP client class listens to DCNM's AMQP notification and interacts with openstack for further tenant and network information. It also communicates with CPNR to populate DHCP data.

process_amqp_msgs()
Process AMQP queue messages.

It connects to AMQP server and calls callbacks to process DCNM events, i.e. routing key containing '.cisco.dcnm.', once they arrive in the queue.

networking_cisco.apps.saf.server.dfa_openstack_helper module

class networking_cisco.apps.saf.server.dfa_openstack_helper.**DfaNeutronHelper**
Bases: object

Helper Routines for Neutron.

add_intf_router(*rout_id*, *tenant_id*, *subnet_lst*)
Add the interfaces to a router.

create_network(*name*, *tenant_id*, *subnet*, *gw=None*)
Create the openstack network, including the subnet.

create_router(*name*, *tenant_id*, *subnet_lst*)
Create a openstack router and add the interfaces.

delete_intf_router(*name*, *tenant_id*, *rout_id*, *subnet_lst*)
Delete the openstack router and remove the interfaces attached.

delete_network(*name*, *tenant_id*, *subnet_id*, *net_id*)
Delete the openstack subnet and network.

delete_network_all_subnets(*net_id*)
Delete the openstack network including all its subnets.

delete_network_subname(*sub_name*)
Delete the network by part of its name, use with caution.

delete_router(*name*, *tenant_id*, *rout_id*, *subnet_lst*)
Delete the openstack router.

Delete the router and remove the interfaces attached to it.

delete_router_by_name(*rtr_name*, *tenant_id*)
Delete the openstack router and its interfaces given its name.

The interfaces should be already removed prior to calling this function.

find_rtr_namespace(*rout_id*)
Find the namespace associated with the router.

get_all_subnets_cidr(*no_mask=False*)
Returns all the subnets.

get_fw(*fw_id*)
Return the Firewall given its ID.

get_fw_policy(*policy_id*)
Return the firewall policy, given its ID.

get_fw_rule(*rule_id*)
Return the firewall rule, given its ID.

get_network_by_name(*nwk_name*)
Search for a openstack network by name.

get_network_by_tenant(*tenant_id*)
Returns the network of a given tenant.

get_router_intf(*router_id*)
Retrieve the router interfaces. Incomplete, TODO(padkrish).

get_router_port_subnet(*subnet_id*)

get_rtr_by_name(*rtr_name*)
Search a router by its name.

get_rtr_name(*router_id*)
Retrieve the router name. Incomplete.

get_subnet_cidr(*subnet_id*)
retrieve the CIDR associated with a subnet, given its ID.

get_subnet_nwk_excl(*tenant_id*, *excl_list*, *excl_part=False*)
Retrieve the subnets of a network.

Get the subnets inside a network after applying the exclusion list.

get_subnets_for_net(*net*)
Returns the subnets in a network.

is_subnet_present(*subnet_addr*)
Returns if a subnet is present.

neutronclient
Returns client object.

program_rtr(*args*, *rout_id*, *namespace=None*)
Execute the command against the namespace.

program_rtr_all_nwk_next_hop(*tenant_id*, *rout_id*, *next_hop*, *excl_list*)
Program the next hop for all networks of a tenant.

program_rtr_default_gw(*tenant_id*, *rout_id*, *gw*)
Program the default gateway of a router.

program_rtr_nwk_next_hop(*rout_id*, *next_hop*, *cidr*)
Program the next hop for all networks of a tenant.

program_rtr_return(*args*, *rout_id*, *namespace=None*)
Execute the command against the namespace and return the result.

remove_rtr_nwk_next_hop(*rout_id*, *next_hop*, *subnet_lst*, *excl_list*)
Remove the next hop for all networks of a tenant.

networking_cisco.apps.saf.server.dfa_server module

This is the DFA enabler server module which is respnsible for processing neutron, keystone and DCNM events. Also interacting with DFA enabler agent module for port events.

class networking_cisco.apps.saf.server.dfa_server.**DfaServer**(*cfg*)
Bases: *networking_cisco.apps.saf.server.dfa_fail_recovery.DfaFailureRecovery*, *networking_cisco.apps.saf.db.dfa_db_models.DfaDBMixin*, *networking_cisco.apps.saf.server.services.firewall.native.fw_mgr.FwMgr*

Process keystone and neutron events.

The supported events project and network create/delete/update. For each events, data will be provided and sent to DCNM.

add_dhcp_port(*p*)

add_lbaas_port(*port_id*, *lb_id*)
Give port id, get port info and send vm info to agent.

Parameters

- **port_id** – port id of vip port
- **lb_id** – vip id for v1 and lbaas_id for v2

cfg

correct_dhcp_ports(*net_id*)

create_subnet(*snet*)
Create subnet.

create_threads()
Create threads on server.

dcnm_network_create_event(*network_info*)
Process network create event from DCNM.

dcnm_network_delete_event(*network_info*)
Process network delete event from DCNM.

decrement_dhcp_check()

delete_lbaas_port(*lb_id*)
send vm down event and delete db.

Parameters **lb_id** – vip id for v1 and lbaas_id for v2

delete_vm_function(*port_id*, *vm=None*)

dhcp_agent_network_add(*dhcp_net_info*)
Process dhcp agent net add event.

dhcp_agent_network_remove(*dhcp_net_info*)
Process dhcp agent net remove event.

get_dci_id(*tenant_id*)

get_project_name(*tenant_id*)

listener_create_event(*listener_info*)
Process listener create event.

This is lbaas v2 vif will be plugged into ovs when first listener is created and unpluged from ovs when last listener is deleted

listener_delete_event(*listener_info*)
Process listener delete event.

This is lbaas v2 vif will be plugged into ovs when first listener is created and unpluged from ovs when last listener is deleted. as the data only contains listener id, we will scan all loadbalancers from db and delete the vdp if their admin state is down in that loadbalancer

loadbalancer_delete_event(*lb_info*)
Process loadbalancer delete event.

This is lbaas v2

need_dhcp_check()

network_create_event(*network_info*)
Process network create event.

Save the network information in the database.

network_create_func(*net*)
Create network in database and dcnm :param net: network dictionary

network_delete_event(*network_info*)
Process network delete event.

neutronclient

pool_create_event(*pool_info*)
Process pool create event.

Extract pool info and get listener info and call next listen_create_event

port_create_event(*port_info*)

port_delete_event(*port_info*)

port_update_event(*port_info*)

process_data(*data*)

process_queue()

project_create_event(*proj_info*)
Create project.

project_create_func(*proj_id*, *proj=None*)
Create project given project uuid

project_delete_event(*proj_info*)
Process project delete event.

project_update_event(*proj_info*)
Process project update event.

There could be change in project name. DCNM doesn't allow change in project (a.k.a tenant). This event may be received for the DCI update. If the change is for DCI, update the DCI portion of the project name and send the update event to the DCNM.

register_segment_dcnm(*cfg*, *seg_id_min*, *seg_id_max*)
Register segmentation id pool with DCNM.

request_uplink_info(*payload*)
Get the uplink from the database and send the info to the agent.

request_vms_info(*payload*)
Get the VMs from the database and send the info to the agent.

send_vm_info(*vm_info*)
Send vm info to the compute host. it will return True/False

service_vnic_create(*vnic_info_arg*)

service_vnic_delete(*vnic_info_arg*)

set_static_ip_address(*payload*)
Set static ip address for a VM.

start_rpc()

stop_rpc()

subnet_create_event(*subnet_info*)
Process subnet create event.

sync_networks()
sync networks.

It will retrieve networks from neutron and populate them in dfa database and dcnm

sync_projects()
Sync projects.

This function will retrieve project from keystone and populate them dfa database and dcnm

turn_on_dhcp_check()

update_agent_status(*agent*, *ts*)

update_port_ip_address()
Find the ip address that assinged to a port via DHCP

The port database will be updated with the ip address.

update_project_info_cache(*pid*, *dci_id=None*, *name=None*, *opcode='add'*, *result='SUCCESS'*)

vip_create_event(*vip_info*)
Process vip create event.

vip_delete_event(*vip_info*)
Process vip delete event.

vm_result_update(*payload*)
Update the result field in VM database.

This request comes from an agent that needs to update the result in VM database to success or failure to reflect the operation's result in the agent.

class networking_cisco.apps.saf.server.dfa_server.**RpcCallBacks**(*obj*)
Bases: object

RPC call back methods.

heartbeat(*context*, *msg*)
Process heartbeat message from agents on compute nodes.

is_mand_arg_present(*intf_dict*)
Check if mndatory parameters is present for LLDPAD.

Just checking for 2 parameters.

request_uplink_info(*context*, *agent*)
Process uplink message from an agent.

request_vms_info(*context*, *agent*)
Process request for VM information from an agent.

save_topo_disc_params(*context*, *msg*)

save_uplink(*context*, *msg*)

set_static_ip_address(*context*, *msg*)
Process request for setting rules in iptables.

In cases that static ip address is assigned for a VM, it is needed to update the iptables rule for that address.

update_vm_result(*context*, *msg*)
Update VM's result field in the DB.

The result reflects the success of failure of operation when an agent processes the vm info.

networking_cisco.apps.saf.server.dfa_server.**dfa_server**()

networking_cisco.apps.saf.server.dfa_server.**save_my_pid**(*cfg*)

Module contents

Submodules

networking_cisco.apps.saf.dfa_cli module

CLI module for fabric enabler.

class networking_cisco.apps.saf.dfa_cli.**DfaCli**
Bases: cmd.Cmd

Represents fabric enabler command line interface.

do_EOF(*line*)
Use Ctrl-D to exit the program.

do_get_config_profile(*line*)

do_get_dcnm_version(*line*)
Get current version of DCNM.

do_get_enabler_version(*line*)
Get current fabric enabler's package version.

do_get_network(*line*)

do_list_networks(*line*)

do_list_organizations(*line*)
Get list of organization on DCNM.

do_prompt(*line*)
Set prompt for the command line.

do_q(*line*)
exit the program.

do_quit(*line*)
exit the program.

do_set_static_ip(*line*)

emptyline()

help_get_config_profile()

help_get_network()

help_list_networks()

help_set_static_ip()

intro = 'Fabric Enabler Command Line Interface'

prompt = '(enabler) '

set_static_ip_address(*ipaddr*, *macaddr*)

setup_client_rpc()

networking_cisco.apps.saf.dfa_cli.**dfa_cli**()

networking_cisco.apps.saf.dfa_enabler_agent module

networking_cisco.apps.saf.dfa_enabler_agent.**dfa_agent**()

networking_cisco.apps.saf.dfa_enabler_server module

networking_cisco.apps.saf.dfa_enabler_server.**dfa_server**()

Module contents

Module contents

networking_cisco.backwards_compatibility package

Submodules

networking_cisco.backwards_compatibility.attributes module

networking_cisco.backwards_compatibility.constants module

networking_cisco.backwards_compatibility.extensions module

networking_cisco.backwards_compatibility.neutron_version module

networking_cisco.backwards_compatibility.rpc module

networking_cisco.backwards_compatibility.worker module

Module contents

networking_cisco.backwards_compatibility.**auto_schedule_routers**(*self*, *hosts*, *r_ids*)

networking_cisco.backwards_compatibility.**get_agent_db_obj**(*agent*)

networking_cisco.backwards_compatibility.**get_context**()

networking_cisco.backwards_compatibility.**get_db_ref**(*context*)

networking_cisco.backwards_compatibility.**get_novaclient_images**(*nclient*)

networking_cisco.backwards_compatibility.**get_reader_session**()

networking_cisco.backwards_compatibility.**get_tunnel_session**(*context*)

networking_cisco.backwards_compatibility.**get_writer_session**()

networking_cisco.config package

Submodules

networking_cisco.config.base module

class networking_cisco.config.base.**RemainderOpt**(*name*, *item_type=None*, ***kwargs*)
Bases: oslo_config.cfg.DictOpt

A greedy dictionary option

This option will greedily parse any unparsed options in the config section this option is included in storing them in a dictionary structure. For example:

A config file like this:

```
[section_a]
host1=value1
host2=value2
```

with a RemainderOpt option defined like:

```
opts = [
    RemainderOpt('hosts')
]
cfg.CONF.register_opts(opts, "section_a")
```

Results in a dictionary like:

```
hosts: {
   host1: value1,
   host2: value2
}
```

which is accessed using the oslo config object being used, for example:

```
cfg.CONF['hosts']['host1']
```

class networking_cisco.config.base.**SubsectionOpt**(*name*, *subopts=None*, ***kwargs*)
Bases: oslo_config.cfg.DictOpt

An option for parsing multiple sections with sub-IDs

This option allows to parse multiple defintions of the same config section which have unique IDs. For example:

```
[switch:switch1]
address=1.1.1.1
password=p1

[switch:switch2]
address=2.2.2.2
password=p2
```

Both sections are type "switch" and will provide the same configuration options, but one is switch1 and the other is for switch2.

Using this option this can be represented like:

```
SubsectionOpt("switch",
              dest="switches",
              subopts=[StrOpt('address')
                       StrOpt('password')])
```

When parsed the above config file example will result in a dictionary in the form:

```
switches: {
    switch1: {
        address: 1.1.1.1,
        password: p1
    },
    switch2: {
        address: 2.2.2.2,
        password: p2
    }
}
```

networking_cisco.config.opts module

networking_cisco.config.opts.**list_asr_conf_opts**()

networking_cisco.config.opts.**list_nexus_conf_opts**()

networking_cisco.config.opts.**list_nexus_vxlan_type_driver_conf_opts**()

networking_cisco.config.opts.**list_ucsm_conf_opts**()

Module contents

networking_cisco.ml2_drivers package

Subpackages

networking_cisco.ml2_drivers.nexus package

Subpackages

networking_cisco.ml2_drivers.nexus.extensions package

Submodules

networking_cisco.ml2_drivers.nexus.extensions.cisco_providernet module

class networking_cisco.ml2_drivers.nexus.extensions.cisco_providernet.**CiscoProviderNetDrive**

Bases: neutron_lib.plugins.ml2.api.ExtensionDriver, neutron.db.common_db_mixin.CommonDbMixin

extension_alias

initialize()

process_create_network(*context*, *data*, *result*)

Module contents

Submodules

networking_cisco.ml2_drivers.nexus.config module

networking_cisco.ml2_drivers.nexus.constants module

networking_cisco.ml2_drivers.nexus.exceptions module

Exceptions used by Cisco Nexus ML2 mechanism driver.

exception networking_cisco.ml2_drivers.nexus.exceptions.**CredentialAlreadyExists**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Credential name already exists.

message = 'Credential %(credential_name)s already exists for tenant %(tenant_id)s.'

exception networking_cisco.ml2_drivers.nexus.exceptions.**CredentialNameNotFound**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Credential Name could not be found.

message = 'Credential %(credential_name)s could not be found.'

exception networking_cisco.ml2_drivers.nexus.exceptions.**CredentialNotFound**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Credential with this ID cannot be found.

message = 'Credential %(credential_id)s could not be found.'

exception networking_cisco.ml2_drivers.nexus.exceptions.**NexusConfigFailed**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Failed to configure Nexus switch.

message = 'Failed to configure Nexus switch: %(nexus_host)s Config: %(config)s. Reas

exception networking_cisco.ml2_drivers.nexus.exceptions.**NexusConnectFailed**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Failed to connect to Nexus switch.

message = 'Unable to connect to Nexus %(nexus_host)s. Reason: %(exc)s.'

exception networking_cisco.ml2_drivers.nexus.exceptions.**NexusCredentialNotFound**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Credential for this switch_ip cannot be found.

message = 'Credential for switch %(switch_ip)s could not be found.'

exception networking_cisco.ml2_drivers.nexus.exceptions.**NexusHostMappingNotFound**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

NexusHost Mapping is not present.

message = 'Nexus Host Mapping (%(filters)s) is not present'

exception networking_cisco.ml2_drivers.nexus.exceptions.**NexusMissingRequiredFields**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Missing required fields to configure nexus switch.

message = 'Missing required field(s) to configure nexus switch: %(fields)s'

exception networking_cisco.ml2_drivers.nexus.exceptions.**NexusPortBindingNotFound**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

NexusPort Binding is not present.

message = 'Nexus Port Binding (%(filters)s) is not present'

exception networking_cisco.ml2_drivers.nexus.exceptions.**NexusVPCAllocFailure**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Nexus VPC alloc Failure.

message = 'Unable to allocate vpcid for all switches (%s).'

exception networking_cisco.ml2_drivers.nexus.exceptions.**NexusVPCAllocIncorrectArgCount**(***kw*
Bases: neutron_lib.exceptions.NeutronException

Nexus VPC alloc args count incorrect.

message = 'Nexus VPC Alloc init failed. Expected 2 args for start,end received %(count

exception networking_cisco.ml2_drivers.nexus.exceptions.**NexusVPCAllocNotFound**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Nexus VPC alloc is not present.

message = 'Nexus VPC Alloc (%(filters)s) is not present.'

exception networking_cisco.ml2_drivers.nexus.exceptions.**NexusVPCExpectedNoChgrp**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Allocated Channel group not consistent on interface set in switches.

message = 'Channel group state in baremetal interface set not consistent: first inter

exception networking_cisco.ml2_drivers.nexus.exceptions.**NexusVPCLearnedNotConsistent**(***kwarg*
Bases: neutron_lib.exceptions.NeutronException

Learned Channel group not consistent on interface set in switches.

message = 'Learned Nexus channel group not consistent on this interface set: first in

exception networking_cisco.ml2_drivers.nexus.exceptions.**NoDynamicSegmentAllocated**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

VLAN dynamic segment not allocated.

message = 'VLAN dynamic segment not created for Nexus VXLAN overlay static segment. Ne

exception networking_cisco.ml2_drivers.nexus.exceptions.**NoNexusSviSwitch**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

No usable nexus switch found.

message = 'No usable Nexus switch found to create SVI interface.'

exception networking_cisco.ml2_drivers.nexus.exceptions.**PhysnetNotConfigured**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Variable 'physnet' is not configured.

message = "Configuration variable 'physnet' is not configured for host_id %(host_id)s.

exception networking_cisco.ml2_drivers.nexus.exceptions.**PortIdForNexusSvi**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Port Id specified for Nexus SVI.

message = 'Nexus hardware router gateway only uses Subnet Ids.'

exception networking_cisco.ml2_drivers.nexus.exceptions.**SubnetInterfacePresent**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Subnet SVI interface already exists.

message = 'Subnet %(subnet_id)s has an interface on %(router_id)s.'

exception networking_cisco.ml2_drivers.nexus.exceptions.**SubnetNotSpecified**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

Subnet id not specified.

message = 'No subnet_id specified for router gateway.'

networking_cisco.ml2_drivers.nexus.mech_cisco_nexus module

ML2 Mechanism Driver for Cisco Nexus platforms.

class networking_cisco.ml2_drivers.nexus.mech_cisco_nexus.**CiscoNexusCfgMonitor**(*driver*, *mdriver*)
Bases: neutron_lib.worker.BaseWorker

Replay config on communication failure between OpenStack to Nexus.

check_connections()
Check connection between OpenStack to Nexus device.

replay_config(*switch_ip*)
Sends pending config data in OpenStack to Nexus.

reset()

start()

stop()

wait()

class networking_cisco.ml2_drivers.nexus.mech_cisco_nexus.**CiscoNexusMechanismDriver**
Bases: neutron_lib.plugins.ml2.api.MechanismDriver

Cisco Nexus ML2 Mechanism Driver.

bind_port(*context*)

configure_next_batch_of_vlans(*switch_ip*)
Get next batch of vlans and send them to Nexus.

configure_switch_entries(*switch_ip*, *port_bindings*)
Create a nexus switch entry in Nexus.

The port_bindings is sorted by vlan_id, vni, port_id. When there is a change in vlan_id or vni, then vlan data is configured in Nexus device. Otherwise we check if there is a change in port_id where we configure the port with vlan trunk config.

Called during switch replay event.

create_network_precommit(*context*)

create_port_postcommit(*context*)
Create port non-database commit event.

delete_network_postcommit(*context*)

delete_port_postcommit(*context*)
Delete port non-database commit event.

delete_port_precommit(*context*)
Delete port pre-database commit event.

get_all_switch_ips()
Using reserved switch binding get all switch ips.

get_nve_loopback(*switch_ip*)

get_switch_ip_and_active_state(*switch_ip*)

get_switch_ips()

get_switch_nexus_type(*switch_ip*)

get_switch_replay_failure(*fail_key*, *switch_ip*)

get_workers()

incr_switch_replay_failure(*fail_key*, *switch_ip*)

initialize()

is_replay_enabled()

is_switch_active(*switch_ip*)

register_switch_as_inactive(*switch_ip*, *func_name*)

reset_switch_replay_failure(*fail_key*, *switch_ip*)

set_switch_ip_and_active_state(*switch_ip*, *state*)

set_switch_nexus_type(*switch_ip*, *type*)

update_port_postcommit(*context*)
Update port non-database commit event.

update_port_precommit(*context*)
Update port pre-database transaction commit event.

networking_cisco.ml2_drivers.nexus.nexus_db_v2 module

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**add_host_mapping**(*host_id*, *nexus_ip*, *interface*, *ch_grp*, *is_static*)

Add Host to interface mapping entry into mapping data base.

Parameters

- **host_id** – is the name of the host to add
- **interface** – is the interface for this host
- **nexus_ip** – is the ip addr of the nexus switch for this interface
- **ch_grp** – is the port channel this interface belos
- **is_static** – whether this is from conf file or learned from baremetal.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**add_nexusnve_binding**(*vni*, *switch_ip*, *device_id*, *mcast_group*)

Adds a nexus nve binding.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**add_nexusport_binding**(*port_id*, *vlan_id*, *vni*, *switch_ip*, *instance_id*, *is_native=False*, *ch_grp=0*)

Adds a nexusport binding.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**add_provider_network**(*network_id*, *vlan_id*)

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**add_reserved_switch_binding**(*switch_ip*, *state*)

Add a reserved switch binding.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**alloc_vpcid**(*nexus_ips*)

Allocate a vpc id for the given list of switch_ips.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**delete_provider_network**(*network_id*)

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**delete_vpcid_for_switch**(*vpc_id*, *switch_ip*)

Removes unused vpcid for a switch.

Parameters

- **vpc_id** – vpc id to remove
- **switch_ip** – ip address of the switch

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**free_vpcid_for_switch**(*vpc_id*, *nexus_ip*)

Free a vpc id for the given switch_ip.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**free_vpcid_for_switch_list**(*vpc_id*, *nexus_ips*)

Free a vpc id for the given list of switch_ips.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_active_switch_vpc_allocs**(*switch_ip*)

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_all_host_mappings**()

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_all_switch_vpc_allocs**(*switch_ip*)

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_free_switch_vpc_allocs**(*switch_ip*)

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_host_mappings**(*host_id*)

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_nexus_switchport_binding**(*port_id*, *switch_ip*)

Lists all bindings for this switch & port.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_nexusport_binding**(*port_id*, *vlan_id*, *switch_ip*, *instance_id*)

Lists a nexusport binding.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_nexusport_switch_bindings**(*switch_ip*)

Lists all Nexus port switch bindings.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_nexussvi_bindings**()

Lists nexus svi bindings.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_nexusvlan_binding**(*vlan_id*, *switch_ip*)

Lists a vlan and switch binding.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_nexusvm_bindings**(*vlan_id*, *instance_id*)

Lists nexusvm bindings.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_nve_switch_bindings**(*switch_ip*)

Return all the nexus nve bindings for one switch.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_nve_vni_deviceid_bindings**(*vni*, *device_id*)

Return all the nexus nve bindings for one vni/one device_id.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_nve_vni_member_bindings**(*vni*, *switch_ip*, *device_id*)

Return the nexus nve binding per switch and device_id.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_nve_vni_switch_bindings**(*vni*, *switch_ip*)

Return the nexus nve binding(s) per switch.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_port_switch_bindings**(*port_id*, *switch_ip*)

List all vm/vlan bindings on a Nexus switch port.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_port_vlan_switch_binding**(*port_id*, *vlan_id*, *switch_ip*)

Lists nexusvm bindings.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_reserved_bindings**(*vlan_id*, *instance_id*, *switch_ip=None*, *port_id=None*)

Lists reserved bindings.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_reserved_switch_binding**(*switch_ip=None*)

Get a reserved switch binding.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_switch_and_host_mappings**(*host_id*, *switch_ip*)

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_switch_host_mappings**(*switch_ip*)

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_switch_if_host_mappings**(*switch_ip*, *if_id*)

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_switch_vpc_alloc**(*switch_ip*, *vpc_id*)

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**get_switch_vpc_count_min_max**(*switch_ip*)

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**init_vpc_entries**(*nexus_ip*, *vpc_list*)

Initialize switch/vpc entries in vpc alloc data base.

param: nexus_ip ip addr of the nexus switch for this interface param: vpc_list list of vpc integers to create

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**is_provider_network**(*network_id*)

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**is_provider_vlan**(*vlan_id*)

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**is_reserved_binding**(*binding*)

Identifies switch & port operational bindings.

There are two types of reserved bindings.

1. The Switch binding purpose is to keep track of the switch state for when replay is enabled. Keeping it in the db, allows for all processes to determine known state of each switch.
2. The reserved port binding is used with baremetal transactions which don't rely on host to interface mapping in the ini file. It is learned from the transaction and kept in the data base for further reference.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**remove_all_nexusnve_bindings**()

Removes all nexusnve bindings.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**remove_all_nexusport_bindings**()

Removes all nexusport bindings.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**remove_all_static_host_mappings**()

Remove all entries defined in config file from mapping data base.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**remove_host_mapping**(*interface*, *nexus_ip*)

Remove host to interface mapping entry from mapping data base.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**remove_nexusnve_binding**(*vni*, *switch_ip*, *device_id*)

Remove the nexus nve binding.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**remove_nexusport_binding**(*port_id*, *vlan_id*, *vni*, *switch_ip*, *instance_id*)

Removes a nexusport binding.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**remove_reserved_binding**(*vlan_id*, *switch_ip*, *instance_id*, *port_id*)

Removes reserved binding.

This overloads port bindings to support reserved Switch binding used to maintain the state of a switch so it can be viewed by all other neutron processes. There's also the case of a reserved port binding to keep switch information on a given interface. The values of these arguments is as follows: :param vlan_id: 0 :param switch_ip: ip address of the switch :param instance_id: fixed string RESERVED_NEXUS_SWITCH_DEVICE_ID_R1 :param port_id: switch-state of ACTIVE, RESTORE_S1, RESTORE_S2, INACTIVE : port-expected port_id

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**update_host_mapping**(*host_id*, *interface*, *nexus_ip*, *new_ch_grp*)

Change channel_group in host/interface mapping data base.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**update_nexusport_binding**(*port_id*, *new_vlan_id*)

Updates nexusport binding.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**update_reserved_binding**(*vlan_id*, *switch_ip*, *instance_id*, *port_id*, *is_switch_binding=True*, *is_native=False*, *ch_grp=0*)

Updates reserved binding.

This overloads port bindings to support reserved Switch binding used to maintain the state of a switch so it can be viewed by all other neutron processes. There's also the case of a reserved port binding to keep switch information on a given interface.

The values of these arguments is as follows: :param vlan_id: 0 :param switch_ip: ip address of the switch :param instance_id: fixed string RESERVED_NEXUS_SWITCH_DEVICE_ID_R1 :param port_id: switch-state of ACTIVE, RESTORE_S1, RESTORE_S2, INACTIVE : port-expected port_id :param ch_grp: 0 if no port-channel else non-zero integer

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**update_reserved_switch_binding**(*switch_ip*, *state*)

Update a reserved switch binding.

networking_cisco.ml2_drivers.nexus.nexus_db_v2.**update_vpc_entry**(*nexus_ips*, *vpc_id*, *learned*, *active*)

Change active state in vpc_allocate data base.

networking_cisco.ml2_drivers.nexus.nexus_helpers module

ML2 Nexus Driver - Helper Methods

networking_cisco.ml2_drivers.nexus.nexus_helpers.**format_interface_name**(*intf_type*, *port*, *ch_grp=0*)

Method to format interface name given type, port.

Given interface type, port, and channel-group, this method formats an interface name. If channel-group is non-zero, then port-channel is configured.

Parameters

- **intf_type** – Such as 'ethernet' or 'port-channel'
- **port** – unique identification – 1/32 or 1

Ch_grp If non-zero, ignore other params and format port-channel<ch_grp>

Returns the full formatted interface name. ex: ethernet:1/32, port-channel:1

networking_cisco.ml2_drivers.nexus.nexus_helpers.**is_baremetal**(*port*)

Identifies ironic baremetal transactions.

There are two types of transactions.

1. A host transaction which is dependent on host to interface mapping config stored in the ml2_conf.ini file. The VNIC type for this is 'normal' which is the assumed condition.
2. A baremetal transaction which comes from the ironic project where the interfaces are provided in the port transaction. In this case the VNIC_TYPE is 'baremetal'.

networking_cisco.ml2_drivers.nexus.nexus_helpers.**split_interface_name**(*interface*, *ch_grp=0*)

Method to split interface type, id from name.

Takes an interface name or just interface suffix and returns interface type and number separately.

Parameters

- **interface** – interface name or just suffix
- **ch_grp** – if non-zero, ignore interface name and return 'port-channel' grp

Returns interface type like 'ethernet'

Returns returns suffix to interface name

networking_cisco.ml2_drivers.nexus.nexus_models_v2 module

class networking_cisco.ml2_drivers.nexus.nexus_models_v2.**NexusHostMapping**(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base

Nexus Host to interface Mapping.

ch_grp

host_id

if_id

is_static

switch_ip

class networking_cisco.ml2_drivers.nexus.nexus_models_v2.**NexusMcastGroup**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base, neutron_lib.db.model_base.HasId

associated_vni

id

mcast_group

class networking_cisco.ml2_drivers.nexus.nexus_models_v2.**NexusNVEBinding**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents Network Virtualization Endpoint configuration.

device_id

mcast_group

switch_ip

vni

class networking_cisco.ml2_drivers.nexus.nexus_models_v2.**NexusPortBinding**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents a binding of VM's to nexus ports.

binding_id

channel_group

instance_id

is_native

port_id

switch_ip

vlan_id

vni

class networking_cisco.ml2_drivers.nexus.nexus_models_v2.**NexusProviderNetwork**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

network_id

vlan_id

class networking_cisco.ml2_drivers.nexus.nexus_models_v2.**NexusVPCAlloc**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Nexus Port Channel Allocation.

active

learned

switch_ip

vpc_id

class networking_cisco.ml2_drivers.nexus.nexus_models_v2.**NexusVxlanAllocation**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

allocated

vxlan_vni

networking_cisco.ml2_drivers.nexus.nexus_restapi_client module

Implements REST API Client For Nexus

class networking_cisco.ml2_drivers.nexus.nexus_restapi_client.**CiscoNexusRestapiClient**(*creden ac- cepted 201, 204], schem time- out=3 max_r re- quest_*

Bases: object

rest_delete(*action*, *ipaddr=None*, *body=None*, *headers=None*)

rest_get(*action*, *ipaddr*, *body=None*, *headers=None*)

rest_post(*action*, *ipaddr=None*, *body=None*, *headers=None*)

send_request(*method*, *action*, *body=None*, *headers=None*, *ipaddr=None*)
Perform the HTTP request.

The response is in either JSON format or plain text. A GET method will invoke a JSON response while a PUT/POST/DELETE returns message from the the server in plain text format. Exception is raised when server replies with an INTERNAL SERVER ERROR status code (500) i.e. an error has occurred on the server or SERVICE UNAVAILABLE (404) i.e. server is not reachable.

Parameters

- **method** – type of the HTTP request. POST, GET, PUT or DELETE
- **action** – path to which the client makes request
- **body** – dict of arguments which are sent as part of the request
- **headers** – header for the HTTP request
- **server_ip** – server_ip for the HTTP request.

Returns JSON or plain text in HTTP response

networking_cisco.ml2_drivers.nexus.nexus_restapi_network_driver module

Implements a Nexus-OS NETCONF over SSHv2 API Client

class networking_cisco.ml2_drivers.nexus.nexus_restapi_network_driver.**CiscoNexusRestapiDriv**
Bases: object

Nexus Driver Restapi Class.

add_ch_grp_to_interface(*nexus_host*, *if_type*, *port*, *ch_grp*)
Applies channel-group n to ethernet interface.

capture_and_print_timeshot(*start_time*, *which*, *other=99*, *switch='x.x.x.x'*)
Determine delta, keep track, and print results.

create_and_trunk_vlan(*nexus_host*, *vlan_id*, *intf_type*, *nexus_port*, *vni*, *is_native*)
Create VLAN and trunk it on the specified ports.

create_nve_member(*nexus_host*, *nve_int_num*, *vni*, *mcast_group*)
Add a member configuration to the NVE interface.

create_port_channel(*nexus_host*, *vpc_nbr*)
Creates port channel n on Nexus switch.

create_vlan(*nexus_host*, *vlanid*, *vni*)
Given switch, vlanid, vni, Create a VLAN on Switch.

delete_ch_grp_to_interface(*nexus_host*, *if_type*, *port*, *ch_grp*)
Removes channel-group n from ethernet interface.

delete_nve_member(*nexus_host*, *nve_int_num*, *vni*)
Delete a member configuration on the NVE interface.

delete_port_channel(*nexus_host*, *vpc_nbr*)
Deletes delete port channel on Nexus switch.

delete_vlan(*nexus_host*, *vlanid*)
Delete a VLAN on Nexus Switch given the VLAN ID.

disable_vlan_on_trunk_int(*nexus_host*, *vlanid*, *intf_type*, *interface*, *is_native*)
Disable a VLAN on a trunk interface.

disable_vxlan_feature(*nexus_host*)
Disable VXLAN on the switch.

enable_vxlan_feature(*nexus_host*, *nve_int_num*, *src_intf*)
Enable VXLAN on the switch.

end_create_vlan(*conf_str*)
Returns current config + end of config.

get_create_vlan(*nexus_host*, *vlanid*, *vni*, *conf_str*)
Returns an XML snippet for create VLAN on a Nexus Switch.

get_interface_switch(*nexus_host*, *intf_type*, *interface*)
Get the interface data from host.

Parameters

- **nexus_host** – IP address of Nexus switch
- **intf_type** – String which specifies interface type. example: ethernet
- **interface** – String indicating which interface. example: 1/19

Returns response Returns interface data

get_nexus_type(*nexus_host*)
Given the nexus host, get the type of Nexus switch.

Parameters **nexus_host** – IP address of Nexus switch

Returns Nexus type

initialize_all_switch_interfaces(*interfaces*, *switch_ip=None*, *replay=True*)
Configure Nexus interface and get port channel number.

Called during switch replay or just init if no replay is configured. For latter case, only configured interfaces are affected by this method.

During switch replay, the change group from the host mapping data base is used. There is no attempt to relearn port-channel from the Nexus switch. What we last knew it to be will persist.

Parameters

- **interfaces** – List of interfaces for a given switch. ch_grp can be altered as last arg to each interface. If no ch_grp, this arg will be zero.
- **switch_ip** – IP address of Nexus switch
- **replay** – Whether in replay path

initialize_baremetal_switch_interfaces(*interfaces*)
Initialize Nexus interfaces and for initial baremetal event.

This get/create port channel number, applies channel-group to ethernet interface, and initializes trunking on interface.

Parameters **interfaces** – Receive a list of interfaces containing: nexus_host: IP address of Nexus switch intf_type: String which specifies interface type. example: ethernet interface: String indicating which interface. example: 1/19 is_native: Whether native vlan must be configured. ch_grp: May replace port channel to each entry. channel number is 0 if none

send_edit_string(*nexus_host*, *path_snip*, *body_snip*, *check_to_close_session=True*)
Sends rest Post request to Nexus switch.

send_enable_vlan_on_trunk_int(*nexus_host*, *vlanid*, *intf_type*, *interface*, *is_native*, *add_mode=False*)
Gathers and sends an interface trunk XML snippet.

set_all_vlan_states(*nexus_host*, *vlanid_range*)
Set the VLAN states to active.

start_create_vlan()
Returns REST API path and config start.

networking_cisco.ml2_drivers.nexus.nexus_restapi_snippets module

networking_cisco.ml2_drivers.nexus.trunk module

class networking_cisco.ml2_drivers.nexus.trunk.**NexusMDTrunkHandler**
Bases: object

Cisco Nexus Mechanism Driver Trunk Handler.

This class contains methods called by the cisco_nexus MD for processing trunk subports.

is_trunk_parentport(*port*)

is_trunk_subport(*port*)

is_trunk_subport_baremetal(*port*)

update_subports(*port*)
Set port attributes for trunk subports.

For baremetal deployments only, set the neutron port attributes during the bind_port event.

networking_cisco.ml2_drivers.nexus.type_nexus_vxlan module

class networking_cisco.ml2_drivers.nexus.type_nexus_vxlan.**NexusVxlanTypeDriver**
Bases: neutron.plugins.ml2.drivers.type_tunnel.ML2TunnelTypeDriver

add_endpoint(*ip*, *udp_port*)

allocate_tenant_segment(*context*)

delete_endpoint(*ip*)

delete_endpoint_by_host_or_ip(*host*, *ip*)

get_endpoint_by_host(*host*)

get_endpoint_by_ip(*ip*)

get_endpoints()

get_type()

initialize()

release_segment(*context*, *segment*)

reserve_provider_segment(*context*, *segment*)

sync_allocations()
Synchronize vxlan_allocations table with configured tunnel ranges.

Module contents

networking_cisco.ml2_drivers.ucsm package

Submodules

networking_cisco.ml2_drivers.ucsm.config module

class networking_cisco.ml2_drivers.ucsm.config.**EthPortType**(*choices=None*, *quotes=False*, *regex=None*, *ignore_case=False*, *max_length=None*, *type_name='string value'*)
Bases: oslo_config.types.String

networking_cisco.ml2_drivers.ucsm.config.**LOG = <oslo_log.log.KeywordArgumentAdapter object>**
Cisco UCS Manager ML2 Mechanism driver specific configuration.

Following are user configurable options for UCS Manager ML2 Mechanism driver. The ucsm_username, ucsm_password, and ucsm_ip are required options in single UCS Manager mode. A repetitive block starting with ml2_cisco_ucsm_ip signals multi-UCSM configuration. When both are present, the multi-UCSM config will only take effect.

class networking_cisco.ml2_drivers.ucsm.config.**SPTemplateListType**(*type_name='SPTemplateList'*)
Bases: oslo_config.types.ConfigType

class networking_cisco.ml2_drivers.ucsm.config.**UCSTemplate**(*path*, *name*)
Bases: object

class networking_cisco.ml2_drivers.ucsm.config.**VNICTemplateListType**(*type_name='VNICTemplateList'*)
Bases: oslo_config.types.ConfigType

class networking_cisco.ml2_drivers.ucsm.config.**VlanListType**(*item_type=None*, *bounds=False*, *type_name='list value'*)
Bases: oslo_config.types.List

networking_cisco.ml2_drivers.ucsm.config.**add_sp_template_config_for_host**(*host*, *ucsm_ip*, *sp_template_path*, *sp_template*)

networking_cisco.ml2_drivers.ucsm.config.**load_single_ucsm_config**()

networking_cisco.ml2_drivers.ucsm.config.**update_sp_template_config**(*host_id*, *ucsm_ip*, *sp_template_with_path*)

networking_cisco.ml2_drivers.ucsm.constants module

networking_cisco.ml2_drivers.ucsm.exceptions module

Exceptions used by Cisco UCSM ML2 mechanism driver.

exception networking_cisco.ml2_drivers.ucsm.exceptions.**UcsmConfigDeleteFailed**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

message = 'Failed to delete %(config)s on UCS Manager %(ucsm_ip)s. Reason: %(exc)s.'

exception networking_cisco.ml2_drivers.ucsm.exceptions.**UcsmConfigFailed**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

message = 'Failed to configure %(config)s on UCS Manager %(ucsm_ip)s. Reason: %(exc)s

exception networking_cisco.ml2_drivers.ucsm.exceptions.**UcsmConfigReadFailed**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

message = 'Unable to read config from UCS Manager %(ucsm_ip)s. Reason: %(exc)s.'

exception networking_cisco.ml2_drivers.ucsm.exceptions.**UcsmConnectFailed**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

message = 'Unable to connect to UCS Manager %(ucsm_ip)s. Reason: %(exc)s.'

exception networking_cisco.ml2_drivers.ucsm.exceptions.**UcsmDisconnectFailed**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

message = 'Disconnect to UCS Manager %(ucsm_ip)s failed. Reason: %(exc)s.'

networking_cisco.ml2_drivers.ucsm.mech_cisco_ucsm module

class networking_cisco.ml2_drivers.ucsm.mech_cisco_ucsm.**CiscoUcsmMechanismDriver**
Bases: neutron_lib.plugins.ml2.api.MechanismDriver

ML2 Mechanism Driver for Cisco UCS Manager.

bind_port(*context*)
Binds port to current network segment.

Binds port only if the vnic_type is direct or macvtap and the port is from a supported vendor. While binding port set it in ACTIVE state and provide the Port Profile or Vlan Id as part vif_details.

static check_segment(*segment*)

delete_network_postcommit(*context*)
Delete all configuration added to UCS Manager for the vlan_id.

delete_network_precommit(*context*)
Delete entry corresponding to Network's VLAN in the DB.

initialize()

static make_profile_name(*vlan_id*)

update_port_postcommit(*context*)
Creates a port profile on UCS Manager.

Creates a Port Profile for this VLAN if it does not already exist.

update_port_precommit(*context*)
Adds port profile and vlan information to the DB.

Assign a port profile to this port. To do that: 1. Get the vlan_id associated with the bound segment 2. Check if a port profile already exists for this vlan_id 3. If yes, associate that port profile with this port. 4. If no, create a new port profile with this vlan_id and associate with this port

networking_cisco.ml2_drivers.ucsm.ucs_ssl module

class networking_cisco.ml2_drivers.ucsm.ucs_ssl.**SSLContext**(**args*, ***kwargs*)
Bases: eventlet.green.ssl.GreenSSLContext

networking_cisco.ml2_drivers.ucsm.ucs_ssl.**wrap_socket**(*sock*, *keyfile=None*, *certfile=None*, *server_side=False*, *cert_reqs=0*, *ssl_version=<_SSLMethod.PROTOCOL_SSLv23: 2>*, *ca_certs=None*, *do_handshake_on_connect=True*, *suppress_ragged_eofs=True*, *ciphers=None*)

networking_cisco.ml2_drivers.ucsm.ucsm_db module

class networking_cisco.ml2_drivers.ucsm.ucsm_db.**UcsmDbModel**
Bases: object

add_port_profile(*profile_name*, *vlan_id*, *device_id*)
Adds a port profile and its vlan_id to the table.

add_port_profile_to_delete_table(*profile_name*, *device_id*)
Adds a port profile to the delete table.

add_service_profile_template(*vlan_id*, *sp_template*, *ucsm_ip*)
Adds an entry for a vlan_id on a SP template to the table.

add_vnic_template(*vlan_id*, *ucsm_ip*, *vnic_template*, *physnet*)
Adds an entry for a vlan_id on a SP template to the table.

delete_sp_template_for_vlan(*vlan_id*)
Deletes SP Template for a vlan_id if it exists.

delete_vlan_entry(*vlan_id*)
Deletes entry for a vlan_id if it exists.

delete_vnic_template_for_vlan(*vlan_id*)
Deletes VNIC Template for a vlan_id and physnet if it exists.

get_all_port_profiles_to_delete()

get_port_profile_for_vlan(*vlan_id*, *device_id*)
Returns Vlan id associated with the port profile.

get_sp_template_vlan_entry(*vlan_id*, *sp_template*, *ucsm_ip*)

get_vnic_template_vlan_entry(*vlan_id*, *vnic_template*, *ucsm_ip*, *physnet*)

has_port_profile_to_delete(*profile_name*, *device_id*)
Returns True if port profile delete table containes PP.

is_port_profile_created(*vlan_id*, *device_id*)
Indicates if port profile has been created on UCS Manager.

remove_port_profile_to_delete(*profile_name*, *device_id*)
Removes port profile to be deleted from table.

set_port_profile_created(*vlan_id*, *profile_name*, *device_id*)
Sets created_on_ucs flag to True.

set_sp_template_updated(*vlan_id*, *sp_template*, *device_id*)
Sets update_on_ucs flag to True.

set_vnic_template_updated(*vlan_id*, *ucsm_ip*, *vnic_template*, *physnet*)
Sets update_on_ucs flag to True for a Vnic Template entry.

networking_cisco.ml2_drivers.ucsm.ucsm_model module

class networking_cisco.ml2_drivers.ucsm.ucsm_model.**PortProfile**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Port profiles created on the UCS Manager.

created_on_ucs

device_id

profile_id

vlan_id

class networking_cisco.ml2_drivers.ucsm.ucsm_model.**PortProfileDelete**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Port profiles to be deleted on the UCS Manager.

device_id

profile_id

class networking_cisco.ml2_drivers.ucsm.ucsm_model.**ServiceProfileTemplate**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Service Profile Templates modified on the UCS Manager.

device_id

sp_template

updated_on_ucs

vlan_id

class networking_cisco.ml2_drivers.ucsm.ucsm_model.**VnicTemplate**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Vnic Templates modified on the UCS Manager.

device_id

physnet

updated_on_ucs

vlan_id

vnic_template

networking_cisco.ml2_drivers.ucsm.ucsm_network_driver module

class networking_cisco.ml2_drivers.ucsm.ucsm_network_driver.**CiscoUcsmDriver**
Bases: object

UCS Manager Driver Main Class.

check_vnic_type_and_vendor_info(*vnic_type*, *profile*)
Checks if this vnic_type and vendor device info are supported.

Returns True if: 1. the port vnic_type is direct or macvtap and 2. the vendor_id and product_id of the port is supported by this MD Useful in determining if this MD should bind the current port.

create_portprofile(*profile_name*, *vlan_id*, *vnic_type*, *host_id*, *trunk_vlans*)
Top level method to create Port Profiles on the UCS Manager.

Calls all the methods responsible for the individual tasks that ultimately result in the creation of the Port Profile on the UCS Manager.

delete_all_config_for_vlan(*vlan_id*, *port_profile*, *trunk_vlans*)
Top level method to delete all config for vlan_id.

get_ucsm_ip_for_host(*host_id*)

is_vmfex_port(*profile*)
Checks if the port is a VMFEX port.

Returns True only for port that support VM-FEX. It is important to distinguish between the two since Port Profiles on the UCS Manager are created only for the VM-FEX ports.

static make_vlan_name(*vlan_id*)

ucs_manager_connect(*ucsm_ip*)
Connects to a UCS Manager.

ucs_manager_disconnect(*handle*, *ucsm_ip*)
Disconnects from the UCS Manager.

After the disconnect, the handle associated with this connection is no longer valid.

ucsm_connect_disconnect(*ucsm_ip*)

update_service_profile_template(*vlan_id*, *host_id*, *ucsm_ip*)

update_serviceprofile(*host_id*, *vlan_id*)
Top level method to update Service Profiles on UCS Manager.

Calls all the methods responsible for the individual tasks that ultimately result in a vlan_id getting programed on a server's ethernet ports and the Fabric Interconnect's network ports.

update_vnic_template(*host_id*, *vlan_id*, *physnet*, *vnic_template_path*, *vnic_template*)
Updates VNIC Template with the vlan_id.

Module contents

Module contents

networking_cisco.neutronclient package

Submodules

networking_cisco.neutronclient.hostingdevice module

class networking_cisco.neutronclient.hostingdevice.**HostingDevice**(*app*, *app_args*, *cmd_name=None*)
Bases: neutronclient.common.extension.NeutronClientExtension

allow_names = True

log = <logging.Logger object>

object_path = '/dev_mgr/hosting_devices'

resource = 'hosting_device'

resource_path = '/dev_mgr/hosting_devices/%s'

resource_plural = 'hosting_devices'

versions = ['2.0']

class networking_cisco.neutronclient.hostingdevice.**HostingDeviceCreate**(*app*, *app_args*, *cmd_name=None*)
Bases: neutronclient.common.extension.ClientExtensionCreate, *networking_cisco.neutronclient.hostingdevice.HostingDevice*

Create a hosting device for a given tenant.

add_known_arguments(*parser*)

args2body(*parsed_args*)

log = <logging.Logger object>

shell_command = 'cisco-hosting-device-create'

class networking_cisco.neutronclient.hostingdevice.**HostingDeviceDelete**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionDelete, *networking_cisco.neutronclient.hostingdevice.HostingDevice*

Delete a given hosting device.

log = <logging.Logger object>

shell_command = 'cisco-hosting-device-delete'

class networking_cisco.neutronclient.hostingdevice.**HostingDeviceGetConfig**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionShow, *networking_cisco.neutronclient.hostingdevice.HostingDevice*

Fetch running of a given hosting device.

execute(*parsed_args*)

get_hosting_device_config(*client*, *hosting_device_id*)
Get config of hosting_device.

log = <logging.Logger object>

run(*parsed_args*)

shell_command = 'cisco-hosting-device-get-config'

class networking_cisco.neutronclient.hostingdevice.**HostingDeviceList**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionList, *networking_cisco.neutronclient.hostingdevice.HostingDevice*

List hosting devices that belong to a given tenant.

list_columns = ['id', 'name', 'template_id', 'admin_state_up', 'status']

log = <logging.Logger object>

pagination_support = True

shell_command = 'cisco-hosting-device-list'

sorting_support = True

class networking_cisco.neutronclient.hostingdevice.**HostingDeviceShow**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionShow, *networking_cisco.neutronclient.hostingdevice.HostingDevice*

Show information of a given hosting device.

log = <logging.Logger object>

shell_command = 'cisco-hosting-device-show'

class networking_cisco.neutronclient.hostingdevice.**HostingDeviceUpdate**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionUpdate, *networking_cisco.neutronclient.hostingdevice.HostingDevice*

Update hosting device's information.

add_known_arguments (*parser*)

args2body (*parsed_args*)

log = <logging.Logger object>

shell_command = 'cisco-hosting-device-update'

networking_cisco.neutronclient.hostingdevicescheduler module

class networking_cisco.neutronclient.hostingdevicescheduler.**ConfigAgentHandlingHostingDevic**

Bases: neutronclient.common.extension.NeutronClientExtension

allow_names = True

log = <logging.Logger object>

object_path = '/agents'

resource = 'agent'

resource_path = '/agents/%s'

resource_plural = 'agents'

versions = ['2.0']

class networking_cisco.neutronclient.hostingdevicescheduler.**ConfigAgentHandlingHostingDevic**

Bases: neutronclient.common.extension.ClientExtensionList, *networking_cisco.neutronclient.hostingdevicescheduler.ConfigAgentHandlingHostingDevice*

call_server (*neutron_client*, *search_opts*, *parsed_args*)

extend_list (*data*, *parsed_args*)

get_parser (*prog_name*)

list_columns = ['id', 'alive', 'agent_type', 'admin_state_up', 'host']

list_config_agents_handling_hosting_device (*client*, *hosting_device_id*, ***_params*)
Fetches a list of config agents handling a hosting device.

log = <logging.Logger object>

shell_command = 'cisco-hosting-device-list-config-agents'

class networking_cisco.neutronclient.hostingdevicescheduler.**HostingDeviceAssociateWithConfi**

Bases: neutronclient.common.extension.ClientExtensionCreate, *networking_cisco.neutronclient.hostingdevicescheduler.HostingDeviceHandledByConfigAgent*

associate_hosting_device_with_config_agent (*client*, *config_agent_id*, *body*)
Associates a hosting_device with a config agent.

execute (*parsed_args*)

get_parser (*prog_name*)

log = <logging.Logger object>

shell_command = 'cisco-config-agent-associate-hosting-device'

class networking_cisco.neutronclient.hostingdevicescheduler.**HostingDeviceDisassociateFromC**

Bases: neutronclient.common.extension.ClientExtensionCreate, *networking_cisco.neutronclient.hostingdevicescheduler.HostingDeviceHandledByConfigAgent*

disassociate_hosting_device_with_config_agent(*client*, *config_agent_id*, *hosting_device_id*)

Disassociates a hosting_device with a config agent.

execute(*parsed_args*)

get_parser(*prog_name*)

log = <logging.Logger object>

shell_command = 'cisco-config-agent-disassociate-hosting-device'

class networking_cisco.neutronclient.hostingdevicescheduler.**HostingDeviceHandledByConfigAge**

Bases: neutronclient.common.extension.NeutronClientExtension

allow_names = True

log = <logging.Logger object>

object_path = '/dev_mgr/hosting_devices'

resource = 'hosting_device'

resource_path = '/dev_mgr/hosting_devices/%s'

resource_plural = 'hosting_devices'

versions = ['2.0']

class networking_cisco.neutronclient.hostingdevicescheduler.**HostingDeviceHandledByConfigAge**

Bases: neutronclient.common.extension.ClientExtensionList, *networking_cisco.neutronclient.hostingdevicescheduler.HostingDeviceHandledByConfigAgent*

call_server(*neutron_client*, *search_opts*, *parsed_args*)

get_parser(*prog_name*)

list_columns = ['id', 'name', 'admin_state_up', 'template_id']

list_hosting_device_handled_by_config_agent(*client*, *cfg_agent_id*, ***_params*)

Fetches a list of hosting devices handled by a config agent.

log = <logging.Logger object>

shell_command = 'cisco-config-agent-list-hosting-devices'

networking_cisco.neutronclient.hostingdevicetemplate module

class networking_cisco.neutronclient.hostingdevicetemplate.**HostingDeviceTemplate**(*app*, *app_args*, *cmd_name=N*

Bases: neutronclient.common.extension.NeutronClientExtension

allow_names = True

log = <logging.Logger object>

object_path = '/dev_mgr/hosting_device_templates'

resource = 'hosting_device_template'

resource_path = '/dev_mgr/hosting_device_templates/%s'

resource_plural = 'hosting_device_templates'

versions = ['2.0']

class networking_cisco.neutronclient.hostingdevicetemplate.**HostingDeviceTemplateCreate**(*app*, *app_ cmd_*

Bases: neutronclient.common.extension.ClientExtensionCreate, *networking_cisco.neutronclient.hostingdevicetemplate.HostingDeviceTemplate*

Create a hosting device template for a given tenant.

add_known_arguments(*parser*)

args2body(*parsed_args*)

log = <logging.Logger object>

shell_command = 'cisco-hosting-device-template-create'

class networking_cisco.neutronclient.hostingdevicetemplate.**HostingDeviceTemplateDelete**(*app*, *app_ cmd_*

Bases: neutronclient.common.extension.ClientExtensionDelete, *networking_cisco.neutronclient.hostingdevicetemplate.HostingDeviceTemplate*

Delete a given hosting device template.

log = <logging.Logger object>

shell_command = 'cisco-hosting-device-template-delete'

class networking_cisco.neutronclient.hostingdevicetemplate.**HostingDeviceTemplateList**(*app*, *app_arg cmd_na*

Bases: neutronclient.common.extension.ClientExtensionList, *networking_cisco.neutronclient.hostingdevicetemplate.HostingDeviceTemplate*

List hosting device templates that belong to a given tenant.

list_columns = ['id', 'name', 'host_category', 'service_types', 'enabled']

log = <logging.Logger object>

pagination_support = True

shell_command = 'cisco-hosting-device-template-list'

sorting_support = True

class networking_cisco.neutronclient.hostingdevicetemplate.**HostingDeviceTemplateShow**(*app*, *app_arg*
cmd_na

Bases: neutronclient.common.extension.ClientExtensionShow, *networking_cisco.neutronclient.hostingdevicetemplate.HostingDeviceTemplate*

Show information of a given hosting device template.

log = <logging.Logger object>

shell_command = 'cisco-hosting-device-template-show'

class networking_cisco.neutronclient.hostingdevicetemplate.**HostingDeviceTemplateUpdate**(*app*, *app_*
*cmd_*

Bases: neutronclient.common.extension.ClientExtensionUpdate, *networking_cisco.neutronclient.hostingdevicetemplate.HostingDeviceTemplate*

Update hosting device template's information.

add_known_arguments(*parser*)

args2body(*parsed_args*)

log = <logging.Logger object>

shell_command = 'cisco-hosting-device-template-update'

networking_cisco.neutronclient.networkprofile module

class networking_cisco.neutronclient.networkprofile.**NetworkProfile**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.NeutronClientExtension

allow_names = True

log = <logging.Logger object>

object_path = '/network_profiles'

resource = 'network_profile'

resource_path = '/network_profiles/%s'

resource_plural = 'network_profiles'

segment_sub_types = ['native', 'enhanced']

segment_types = ['vlan', 'overlay']

versions = ['2.0']

class networking_cisco.neutronclient.networkprofile.**NetworkProfileCreate**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionCreate, *networking_cisco.neutronclient.networkprofile.NetworkProfile*

Create a network profile.

add_known_arguments(*parser*)

args2body(*parsed_args*)

log = <logging.Logger object>

shell_command = 'cisco-network-profile-create'

class networking_cisco.neutronclient.networkprofile.**NetworkProfileDelete**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionDelete, *networking_cisco.neutronclient.networkprofile.NetworkProfile*

Delete a given network profile.

log = <logging.Logger object>

shell_command = 'cisco-network-profile-delete'

class networking_cisco.neutronclient.networkprofile.**NetworkProfileList**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionList, *networking_cisco.neutronclient.networkprofile.NetworkProfile*

List network profiles that belong to a given tenant.

list_columns = ['id', 'name', 'segment_type', 'sub_type', 'segment_range', 'physical_n

log = <logging.Logger object>

pagination_support = True

shell_command = 'cisco-network-profile-list'

sorting_support = True

class networking_cisco.neutronclient.networkprofile.**NetworkProfileShow**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionShow, *networking_cisco.neutronclient.networkprofile.NetworkProfile*

Show information of a given network profile.

log = <logging.Logger object>

shell_command = 'cisco-network-profile-show'

networking_cisco.neutronclient.policyprofile module

class networking_cisco.neutronclient.policyprofile.**PolicyProfile**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.NeutronClientExtension

allow_names = True

log = <logging.Logger object>

object_path = '/policy_profiles'

resource = 'policy_profile'

resource_path = '/policy_profiles/%s'

resource_plural = 'policy_profiles'

versions = ['2.0']

class networking_cisco.neutronclient.policyprofile.**PolicyProfileList**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionList, *networking_cisco.neutronclient.policyprofile.PolicyProfile*

List policy profiles that belong to a given tenant.

list_columns = ['id', 'name']

log = <logging.Logger object>

pagination_support = True

shell_command = 'cisco-policy-profile-list'

sorting_support = True

class networking_cisco.neutronclient.policyprofile.**PolicyProfileShow**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionShow, *networking_cisco.neutronclient.policyprofile.PolicyProfile*

Show information of a given policy profile.

log = <logging.Logger object>

shell_command = 'cisco-policy-profile-show'

class networking_cisco.neutronclient.policyprofile.**UpdatePolicyProfile**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionUpdate, *networking_cisco.neutronclient.policyprofile.PolicyProfile*

Update policy profile's information.

args2body(*parsed_args*)

get_parser(*prog_name*)

log = <logging.Logger object>

shell_command = 'cisco-policy-profile-update'

networking_cisco.neutronclient.routerscheduler module

class networking_cisco.neutronclient.routerscheduler.**AddRouterToHostingDevice**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionCreate, *networking_cisco.neutronclient.routerscheduler.RoutersOnHostingDevice*

Add a router to hosting device.

add_router_to_hosting_device(*client*, *hosting_device_id*, *body*)

Adds a router to hosting device.

execute(*parsed_args*)

get_parser(*prog_name*)

log = <logging.Logger object>

shell_command = 'cisco-hosting-device-router-add'

class networking_cisco.neutronclient.routerscheduler.**HostingDeviceHostingRouter**(*app*, *app_args*, *cmd_name=No*

Bases: neutronclient.common.extension.NeutronClientExtension

allow_names = True

log = <logging.Logger object>

object_path = '/hosting_devices'

resource = 'hosting_device'

resource_path = '/hosting_devices/%s'

resource_plural = 'hosting_devices'

versions = ['2.0']

class networking_cisco.neutronclient.routerscheduler.**HostingDeviceHostingRouterList**(*app*, *app_args* *cmd_nam*

Bases: neutronclient.common.extension.ClientExtensionList, *networking_cisco.neutronclient.routerscheduler.HostingDeviceHostingRouter*

call_server(*neutron_client*, *search_opts*, *parsed_args*)

get_parser(*prog_name*)

list_columns = ['id', 'name', 'status', 'admin_state_up', 'template_id']

list_hosting_devices_hosting_routers(*client*, *router_id*, ***_params*)
Fetches a list of hosting devices hosting a router.

log = <logging.Logger object>

shell_command = 'cisco-router-list-hosting-devices'

class networking_cisco.neutronclient.routerscheduler.**RemoveRouterFromHostingDevice**(*app*, *app_args*, *cmd_name*

Bases: neutronclient.common.extension.ClientExtensionCreate, *networking_cisco.neutronclient.routerscheduler.RoutersOnHostingDevice*

Remove a router from Hosting Device.

execute(*parsed_args*)

get_parser(*prog_name*)

log = <logging.Logger object>

remove_router_from_hosting_device(*client*, *hosting_device_id*, *router_id*)
Remove a router from hosting_device.

shell_command = 'cisco-hosting-device-router-remove'

class networking_cisco.neutronclient.routerscheduler.**RoutersOnHostingDevice**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.NeutronClientExtension

allow_names = True

log = <logging.Logger object>

object_path = '/routers'

resource = 'router'

resource_path = '/routers/%s'

resource_plural = 'routers'

versions = ['2.0']

class networking_cisco.neutronclient.routerscheduler.**RoutersOnHostingDeviceList**(*app*, *app_args*, *cmd_name=No*

Bases: neutronclient.common.extension.ClientExtensionList, *networking_cisco.neutronclient.routerscheduler.RoutersOnHostingDevice*

call_server(*neutron_client*, *search_opts*, *parsed_args*)

get_parser(*prog_name*)

list_columns = ['id', 'name', 'external_gateway_info']

list_routers_on_hosting_device(*client*, *hosting_device_id*, ***_params*)
Fetches a list of routers hosted on a hosting device.

log = <logging.Logger object>

shell_command = 'cisco-hosting-device-list-hosted-routers'

networking_cisco.neutronclient.routertype module

class networking_cisco.neutronclient.routertype.**RouterType**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.NeutronClientExtension

allow_names = True

log = <logging.Logger object>

object_path = '/routertypes'

resource = 'routertype'

resource_path = '/routertypes/%s'

resource_plural = 'routertypes'

versions = ['2.0']

class networking_cisco.neutronclient.routertype.**RouterTypeCreate**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionCreate, *networking_cisco.neutronclient.routertype.RouterType*

Create a router type for a given tenant.

add_known_arguments(*parser*)

args2body(*parsed_args*)

log = <logging.Logger object>

shell_command = 'cisco-router-type-create'

class networking_cisco.neutronclient.routertype.**RouterTypeDelete**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionDelete, *networking_cisco.neutronclient.routertype.RouterType*

Delete a given router type.

log = <logging.Logger object>

shell_command = 'cisco-router-type-delete'

class networking_cisco.neutronclient.routertype.**RouterTypeList**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionList, *networking_cisco.neutronclient.routertype.RouterType*

List router types that belong to a given tenant.

list_columns = ['id', 'name', 'description', 'template_id']

log = <logging.Logger object>

pagination_support = True

shell_command = 'cisco-router-type-list'

sorting_support = True

class networking_cisco.neutronclient.routertype.**RouterTypeShow**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionShow, *networking_cisco.neutronclient.routertype.RouterType*

Show information of a given router type.

log = <logging.Logger object>

shell_command = 'cisco-router-type-show'

class networking_cisco.neutronclient.routertype.**RouterTypeUpdate**(*app*, *app_args*, *cmd_name=None*)

Bases: neutronclient.common.extension.ClientExtensionUpdate, *networking_cisco.neutronclient.routertype.RouterType*

Update router type's information.

add_known_arguments(*parser*)

args2body(*parsed_args*)

log = <logging.Logger object>

shell_command = 'cisco-router-type-update'

Module contents

networking_cisco.plugins package

Subpackages

networking_cisco.plugins.cisco package

Subpackages

networking_cisco.plugins.cisco.cfg_agent package

Subpackages

networking_cisco.plugins.cisco.cfg_agent.device_drivers package

Subpackages

networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k package

Submodules

networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_auto_config_check module

class networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_auto_config_check

Bases: object

Agent side of the device manager RPC API.

get_all_hosting_devices(*context*)
Get a list of all hosting devices.

class networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_auto_config_check

Bases: object

RoutingServiceHelper(Agent) side of the routing RPC API.

get_all_hosted_routers(*context*)

Make a remote process call to retrieve the sync data for routers that have been scheduled to a hosting device.

Parameters **context** – session context

get_hardware_router_type_id(*context*)
Get the ID for the ASR1k hardware router type.

networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_auto_config_check.**get_r**

networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_auto_config_check.**main**

networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_cfg_syncer module

class networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_cfg_syncer.**Confi**

Bases: object

check_acl_permit_rules_valid(*segment_id*, *acl*, *intf_segment_dict*)

clean_acls(*conn*, *intf_segment_dict*, *segment_nat_dict*, *parsed_cfg*)

clean_interfaces(*conn*, *intf_segment_dict*, *segment_nat_dict*, *parsed_cfg*)

clean_interfaces_ipv4_check(*intf*, *intf_db_dict*)

clean_interfaces_ipv4_hsrp_check(*intf*, *intf_db_dict*)

clean_interfaces_ipv6_check(*intf*, *intf_segment_dict*)

clean_interfaces_nat_check(*intf*, *segment_nat_dict*)

clean_nat_pool(*conn*, *router_id_dict*, *intf_segment_dict*, *segment_nat_dict*, *parsed_cfg*)

clean_nat_pool_overload(*conn*, *router_id_dict*, *intf_segment_dict*, *segment_nat_dict*, *parsed_cfg*)

clean_routes(*conn*, *router_id_dict*, *intf_segment_dict*, *segment_nat_dict*, *parsed_cfg*, *route_regex*)

clean_snat(*conn*, *router_id_dict*, *intf_segment_dict*, *segment_nat_dict*, *parsed_cfg*)

clean_vrfs(*conn*, *router_id_dict*, *parsed_cfg*)

delete_invalid_cfg(*conn=None*)

get_ostk_router_ids(*router_id_dict*)

get_running_config(*conn*)
Get the ASR1k's current running config. :return: Current IOS running config as multiline string

get_running_config_router_ids(*parsed_cfg*)

get_single_cfg(*cfg_line*)

gw_port_hsrp_ip_check(*gw_port*, *ip_addr*)

process_routers_data(*routers*)

subintf_hsrp_ip_check(*intf_list*, *is_external*, *ip_addr*)

subintf_real_ip_check(*intf_list*, *ip_addr*, *netmask*)

subintf_real_ip_check_gw_port(*gw_port*, *ip_addr*, *netmask*)
checks running-cfg derived ip_addr and netmask against neutron-db gw_port

subintf_real_ipv6_check(*intf_list*, *is_external*, *ipv6_addr*, *prefixlen*)

networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_cfg_syncer.**is_port_v6**(*

networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_cfg_validator module

class networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_cfg_validator.**Co**

Bases: object

check_acls(*router*, *running_config*)

check_default_route(*router*, *running_config*)

check_fips(*router*, *running_config*)

check_global_router(*router*, *running_config*, *segment_nat_dict*)

check_interfaces(*router*, *running_config*, *segment_nat_dict*, *is_external*)

check_nat_pool(*router*, *running_config*)

check_router(*router*, *running_config*, *segment_nat_dict*)

check_running_config()

check_tenant_router(*router*, *running_config*, *segment_nat_dict*)

check_vrf(*router*, *running_config*)

get_interface_name_from_hosting_port(*port*)
generates the underlying subinterface name for a port e.g. Port-channel10.200

get_running_config(*conn*)
Get the ASR1k's current running config. :return: Current IOS running config as multiline string

get_vrf_name(*router*)

populate_segment_nat_dict(*segment_nat_dict*, *routers*)

process_routers_data(*routers*)

set_ip_cidr(*intf*)

networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_cfg_validator.**ROUTER_RO**
Compares ASR running-config and neutron DB state, informs caller if any configuration was missing from running-config.

networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_routing_driver module

class networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_routing_driver.**AS**
Bases: *networking_cisco.plugins.cisco.cfg_agent.device_drivers.iosxe.iosxe_routing_driver.IosXeRoutingDriver*

cleanup_invalid_cfg(*hd*, *routers*)

disable_internal_network_NAT(*ri*, *port*, *ext_gw_port*, *itfc_deleted=False*)

disable_router_interface(*ri*, *port=None*)

enable_router_interface(*ri*, *port*)

external_gateway_added(*ri*, *ext_gw_port*)

external_gateway_removed(*ri*, *ext_gw_port*)

floating_ip_added(*ri*, *ext_gw_port*, *floating_ip*, *fixed_ip*)

floating_ip_removed(*ri*, *ext_gw_port*, *floating_ip*, *fixed_ip*)

get_configuration()

internal_network_added(*ri*, *port*)

internal_network_removed(*ri*, *port*)

send_empty_cfg()

networking_cisco.plugins.cisco.cfg_agent.device_drivers.asr1k.asr1k_snippets module

Module contents

networking_cisco.plugins.cisco.cfg_agent.device_drivers.iosxe package

Submodules

networking_cisco.plugins.cisco.cfg_agent.device_drivers.iosxe.cisco_iosxe_snippets module

IOS-XE XML-based configuration snippets

networking_cisco.plugins.cisco.cfg_agent.device_drivers.iosxe.iosxe_routing_driver module

class networking_cisco.plugins.cisco.cfg_agent.device_drivers.iosxe.iosxe_routing_driver.**Io**

Bases: *networking_cisco.plugins.cisco.cfg_agent.device_drivers.devicedriver_api.RoutingDriverBase*

Generic IOS XE Routing Driver.

This driver encapsulates the configuration logic via NETCONF protocol to configure a generic (IOS-XE based) device for implementing Neutron L3 services. These services include routing, NAT and floating IPs (as per Neutron terminology).

DEV_NAME_LEN = 14

caller_name(*skip=2*)

Get a name of a caller in the format module.class.method

skip specifies how many levels of stack to skip while getting caller name. skip=1 means "who calls me", skip=2 "who calls my caller" etc.

An empty string is returned if skipped levels exceed stack height

cleanup_invalid_cfg(*hd*, *routers*)

clear_connection()

disable_internal_network_NAT(*ri*, *port*, *ext_gw_port*)

enable_internal_network_NAT(*ri*, *port*, *ext_gw_port*)

external_gateway_added(*ri*, *ext_gw_port*)

external_gateway_removed(*ri*, *ext_gw_port*)

floating_ip_added(*ri*, *ext_gw_port*, *floating_ip*, *fixed_ip*)

floating_ip_removed(*ri*, *ext_gw_port*, *floating_ip*, *fixed_ip*)

get_configuration()

internal_network_added(*ri*, *port*)

internal_network_removed(*ri*, *port*)

router_added(*ri*)

router_removed(*ri*)

routes_updated(*ri*, *action*, *route*)

networking_cisco.plugins.cisco.cfg_agent.device_drivers.iosxe.run_asr1kv module

Module contents

Submodules

networking_cisco.plugins.cisco.cfg_agent.device_drivers.devicedriver_api module

class networking_cisco.plugins.cisco.cfg_agent.device_drivers.devicedriver_api.**RoutingDrive**
Bases: object

Base class that defines an abstract interface for the Routing Driver.

This class defines the abstract interface/API for the Routing and NAT related operations. Driver class corresponding to a hosting device should inherit this base driver and implement its methods. RouterInfo object (networking_cisco.plugins.cisco.cfg_agent.router_info.RouterInfo) is a wrapper around the router dictionary, with attributes for easy access to parameters.

cleanup_invalid_cfg(*hosting_device*, *routers*)
Cleanup any invalid configuration in backend.

Parameters

- **hosting_device** – hosting_device dictionary for backend
- **routers** – list of router dictionaries for routers on backend

Returns None

disable_internal_network_NAT(*router_info*, *port*, *ext_gw_port*)
Disable NAT on an internal network.

Parameters

- **router_info** – RouterInfo object for this router
- **port** – port dictionary for the port where the internal network is connected
- **ext_gw_port** – port dictionary for the port where the external gateway network is connected

Returns None

enable_internal_network_NAT(*router_info*, *port*, *ext_gw_port*)
Enable NAT on an internal network.

Parameters

- **router_info** – RouterInfo object for this router
- **port** – port dictionary for the port where the internal network is connected

- **ext_gw_port** – port dictionary for the port where the external gateway network is connected

Returns None

external_gateway_added(*router_info*, *ext_gw_port*)
An external network was added to a router.

Parameters

- **router_info** – RouterInfo object of the router
- **ext_gw_port** – port dictionary for the port where the external gateway network is connected

Returns None

external_gateway_removed(*router_info*, *ext_gw_port*)
An external network was removed from the router.

Parameters

- **router_info** – RouterInfo object of the router
- **ext_gw_port** – port dictionary for the port where the external gateway network was connected

Returns None

floating_ip_added(*router_info*, *ext_gw_port*, *floating_ip*, *fixed_ip*)
A floating IP was added.

Parameters

- **router_info** – RouterInfo object for this router
- **ext_gw_port** – port dictionary for the port where the external gateway network is connected
- **floating_ip** – Floating IP as a string
- **fixed_ip** – Fixed IP of internal internal interface as a string

Returns None

floating_ip_removed(*router_info*, *ext_gw_port*, *floating_ip*, *fixed_ip*)
A floating IP was removed.

Parameters

- **router_info** – RouterInfo object for this router
- **ext_gw_port** – port dictionary for the port where the external gateway network is connected
- **floating_ip** – Floating IP as a string
- **fixed_ip** – Fixed IP of internal internal interface as a string

Returns None

get_configuration()
Return configuration of hosting_device for driver instance

Returns configuration as a text string

internal_network_added(*router_info*, *port*)
An internal network was connected to a router.

Parameters

- **router_info** – RouterInfo object for this router
- **port** – port dictionary for the port where the internal network is connected

Returns None

internal_network_removed(*router_info*, *port*)
An internal network was removed from a router.

Parameters

- **router_info** – RouterInfo object for this router
- **port** – port dictionary for the port where the internal network was connected

Returns None

router_added(*router_info*)
A logical router was assigned to the hosting device.

Parameters **router_info** – RouterInfo object for this router

Returns None

router_removed(*router_info*)
A logical router was de-assigned from the hosting device.

Parameters **router_info** – RouterInfo object for this router

Returns None

routes_updated(*router_info*, *action*, *route*)
Routes were updated for router.

Parameters

- **router_info** – RouterInfo object for this router
- **action** – Action on the route , either 'replace' or 'delete'
- **route** – route dictionary with keys 'destination' & 'next_hop'

Returns None

networking_cisco.plugins.cisco.cfg_agent.device_drivers.driver_mgr module

class networking_cisco.plugins.cisco.cfg_agent.device_drivers.driver_mgr.**DeviceDriverManage**
Bases: object

This class acts as a manager for device drivers.

The device driver manager maintains the relationship between the different neutron logical resource (eg: routers, firewalls, vpns etc.) and where they are hosted. For configuring a logical resource (router) in a hosting device, a corresponding device driver object is used. Device drivers encapsulate the necessary configuration information to configure a logical resource (eg: routers, firewalls, vpns etc.) on a hosting device (eg: ASR1k).

The device driver class loads one driver object per hosting device. The loaded drivers are cached in memory, so when a request is made to get driver object for the same hosting device and resource (like router), the existing driver object is reused.

This class is used by the service helper classes.

get_driver(*resource_id*)

get_driver_for_hosting_device(*hd_id*)

remove_driver(*resource_id*)
Remove driver associated to a particular resource.

remove_driver_for_hosting_device(*hd_id*)
Remove driver associated to a particular hosting device.

set_driver(*resource*)
Set the driver for a neutron resource.

Parameters **resource** – Neutron resource in dict format. Expected keys:

```
{
    'id': <value>,
    'hosting_device': { 'id': <value>, },
    'router_type': {'cfg_agent_driver': <value>,  }
}
```

Returns driver object

networking_cisco.plugins.cisco.cfg_agent.device_drivers.dummy_driver module

class networking_cisco.plugins.cisco.cfg_agent.device_drivers.dummy_driver.**DummyRoutingDriv**
Bases: *networking_cisco.plugins.cisco.cfg_agent.device_drivers.devicedriver_api.RoutingDriverBase*

Dummy Routing Driver.

This class emulates a routing driver without a real backing device.

clear_connection()

disable_internal_network_NAT(*ri*, *port*, *ex_gw_port*)

enable_internal_network_NAT(*ri*, *port*, *ex_gw_port*)

external_gateway_added(*ri*, *ex_gw_port*)

external_gateway_removed(*ri*, *ex_gw_port*)

floating_ip_added(*ri*, *ex_gw_port*, *floating_ip*, *fixed_ip*)

floating_ip_removed(*ri*, *ex_gw_port*, *floating_ip*, *fixed_ip*)

internal_network_added(*ri*, *port*)

internal_network_removed(*ri*, *port*)

router_added(*ri*)

router_removed(*ri*)

routes_updated(*ri*, *action*, *route*)

Module contents

networking_cisco.plugins.cisco.cfg_agent.service_helpers package

Submodules

networking_cisco.plugins.cisco.cfg_agent.service_helpers.routing_svc_helper module

class networking_cisco.plugins.cisco.cfg_agent.service_helpers.routing_svc_helper.**CiscoRout**

Bases: object

RoutingServiceHelper(Agent) side of the routing RPC API.

get_hardware_router_type_id(*context*)
Get the ID for the ASR1k hardware router type.

get_router_ids(*context*, *router_ids=None*, *hd_ids=None*)
Make a remote process call to retrieve scheduled routers ids.

get_routers(*context*, *router_ids=None*, *hd_ids=None*)
Make a remote process call to retrieve the sync data for routers.

Parameters

- **context** – session context
- **router_ids** – list of routers to fetch
- **hd_ids** – hosting device ids, only routers assigned to these hosting devices will be returned.

send_update_port_statuses(*context*, *port_ids*, *status*)
Call the pluging to update the port status which updates the DB.

Parameters

- **context** – contains user information
- **port_ids** – list of ids of the ports associated with the status
- **status** – value of the status for the given port list (port_ids)

update_floatingip_statuses(*context*, *router_id*, *fip_statuses*)
Make a remote process call to update operational status for one or several floating IPs.

@param context: contains user information @param router_id: id of router associated with the floatingips
@param fip_statuses: dict with floatingip_id as key and status as value

exception networking_cisco.plugins.cisco.cfg_agent.service_helpers.routing_svc_helper.**IPAdd**
Bases: neutron_lib.exceptions.NeutronException

message = 'Router port %(port_id)s has no IP address on subnet %(subnet_id)s.'

exception networking_cisco.plugins.cisco.cfg_agent.service_helpers.routing_svc_helper.**Multi**
Bases: neutron_lib.exceptions.NeutronException

message = 'There should not be multiple IPv4 subnets %(subnets)s on router port %(port_

class networking_cisco.plugins.cisco.cfg_agent.service_helpers.routing_svc_helper.**RouterIn**

Bases: object

Wrapper class around the (neutron) router dictionary.

Information about the neutron router is exchanged as a python dictionary between plugin and config agent. RouterInfo is a wrapper around that dict, with attributes for common parameters. These attributes keep the state of the current router configuration, and are used for detecting router state changes when an updated router dict is received.

This is a modified version of the RouterInfo class defined in the (reference) l3-agent implementation, for use with cisco config agent.

ha_enabled

id

router

router_name()

snat_enabled

class networking_cisco.plugins.cisco.cfg_agent.service_helpers.routing_svc_helper.**RoutingSe**

Bases: object

collect_state(*configurations*)
Collect state from this helper.

A set of attributes which summarizes the state of the routers and configurations managed by this config agent. :param configurations: dict of configuration values :return dict of updated configuration values

driver_manager

process_service(*device_ids=None*, *removed_devices_info=None*)

router_added_to_hosting_device(*context*, *routers*)

router_deleted(*context*, *routers*)
Deal with router deletion RPC message.

router_removed_from_hosting_device(*context*, *routers*)

routers_removed_from_hosting_device(*context*, *router_ids*)

routers_updated(*context*, *routers*)
Deal with routers modification and creation RPC message.

target = <Target version=1.1>

networking_cisco.plugins.cisco.cfg_agent.service_helpers.routing_svc_helper_aci module

class networking_cisco.plugins.cisco.cfg_agent.service_helpers.routing_svc_helper_aci.**Rout**

Bases: *networking_cisco.plugins.cisco.cfg_agent.service_helpers.routing_svc_helper.RoutingServiceHelper*

networking_cisco.plugins.cisco.cfg_agent.service_helpers.service_helper module

class networking_cisco.plugins.cisco.cfg_agent.service_helpers.service_helper.**QueueMixin**
Bases: object

dequeue(*qname*)

enqueue(*qname*, *data*)

qsize(*qname*)
Return the approximate size of the queue.

class networking_cisco.plugins.cisco.cfg_agent.service_helpers.service_helper.**ServiceHelper**
Bases: object

notify(*resource*, ***kwargs*)
Calls all observers attached to the given subject.

process_service(**args*, ***kwargs*)

register(*observer*)

unregister(*observer*)

update(*resource*, ***kwargs*)
For future support.

Module contents

Submodules

networking_cisco.plugins.cisco.cfg_agent.cfg_agent module

class networking_cisco.plugins.cisco.cfg_agent.cfg_agent.**CiscoCfgAgent**(*host*, *conf=None*)
Bases: neutron.manager.Manager

Cisco Cfg Agent.

This class defines a generic configuration agent for cisco devices which implement network services in the cloud backend. It is based on the (reference) l3-agent, but has been enhanced to support multiple services in addition to routing.

The agent acts like as a container for services and does not do any service specific processing or configuration itself. All service specific processing is delegated to service helpers which the agent loads. Thus routing specific updates are processed by the routing service helper, firewall by firewall helper etc. A further layer of abstraction is implemented by using device drivers for encapsulating all configuration operations of a service on a device. Device drivers are specific to a particular device/service VM eg: ASR1k.

The main entry points in this class are the *process_services()* and *_backlog_task()* .

after_start()

agent_updated(*context*, *payload*)
Deal with agent updated RPC message.

get_assigned_hosting_devices(*max_retry_attempts=3*)

get_hosting_device_configuration(*context*, *payload*)

get_hosting_device_password(*credentials_id*)

get_routing_service_helper()

hosting_devices_assigned_to_cfg_agent(*context*, *payload*)
Deal with hosting devices assigned to this config agent.

hosting_devices_removed(*context*, *payload*)
Deal with hosting device removed RPC message.

hosting_devices_unassigned_from_cfg_agent(*context*, *payload*)
Deal with hosting devices unassigned from this config agent.

process_services(*device_ids=None*, *removed_devices_info=None*)
Process services managed by this config agent.

This method is invoked by any of three scenarios.

1. Invoked by a periodic task running every *RPC_LOOP_INTERVAL* seconds. This is the most common scenario. In this mode, the method is called without any arguments.

2. Called by the *_process_backlogged_hosting_devices()* as part of the backlog processing task. In this mode, a list of device_ids are passed as arguments. These are the list of backlogged hosting devices that are now reachable and we want to sync services on them.

3. Called by the *hosting_devices_removed()* method. This is when the config agent has received a notification from the plugin that some hosting devices are going to be removed. The payload contains the details of the hosting devices and the associated neutron resources on them which should be processed and removed.

To avoid race conditions with these scenarios, this function is protected by a lock.

This method goes on to invoke *process_service()* on the different service helpers.

Parameters

- **device_ids** – List of devices that are now available and needs to be processed
- **removed_devices_info** – Info about the hosting devices which are going to be removed and details of the resources hosted on them. Expected Format:

```
{
 'hosting_data': {'hd_id1': {'routers': [id1, id2, ...]},
                  'hd_id2': {'routers': [id3, id4, ...]}, ...},
 'deconfigure': True/False
}
```

Returns None

target = <Target version=1.1>

class networking_cisco.plugins.cisco.cfg_agent.cfg_agent.**CiscoCfgAgentWithStateReport**(*host*, *conf=*

Bases: *networking_cisco.plugins.cisco.cfg_agent.cfg_agent.CiscoCfgAgent*

send_agent_report(*report*, *context*)
Send the agent report via RPC.

class networking_cisco.plugins.cisco.cfg_agent.cfg_agent.**CiscoDeviceManagementApi**(*topic*, *host*)

Bases: object

Agent side of the device manager RPC API.

get_hosting_devices_for_agent(*context*)
Get a list of hosting devices assigned to this agent.

register_for_duty(*context*)
Report that a config agent is ready for duty.

report_dead_hosting_devices(*context*, *hd_ids=None*)
Report that a hosting device cannot be contacted (presumed dead).

Param context: session context

Param hosting_device_ids: list of non-responding hosting devices

Returns None

report_revived_hosting_devices(*context*, *hd_ids=None*)

networking_cisco.plugins.cisco.cfg_agent.cfg_agent.**main**(*manager='networking_cisco.plugins.cisco.cfg_agent*

networking_cisco.plugins.cisco.cfg_agent.cfg_agent.**mock_ncclient**()

networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions module

Exceptions by Cisco Configuration Agent.

exception networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.**ConnectionException**(***kwa*

Bases: *networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.DriverException*

Connection exception when connecting to IOS XE hosting device.

message = 'Failed connecting to Device. Reason: %(reason)s. Connection params are Use

exception networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.**DriverException**(***kwargs*)

Bases: neutron_lib.exceptions.NeutronException

Exception created by the Driver class.

exception networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.**DriverExpectedKeyNotSetEx**

Bases: *networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.DriverException*

An attribute expected to be set by plugin is missing

message = 'Value for expected key: %(key)s is missing.Driver cannot proceed'

exception networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.**DriverNotExist**(***kwargs*)

Bases: *networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.DriverException*

message = 'Driver %(driver)s does not exist.'

exception networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.**DriverNotFound**(***kwargs*)

Bases: *networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.DriverException*

message = 'Driver not found for %(resource)s id:%(id)s.'

exception networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.**DriverNotSetForMissingPa**

Bases: *networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.DriverException*

message = 'Driver cannot be set for missing parameter:%(p)s.'

exception networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.**HAParamsMissingException**

Bases: *networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.DriverException*

MissingParams exception thrown when HA params are missing

message = 'For router: %(r_id)s and port: %(p_id)s, HA_ENABLED is set, but port ha i

exception networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.**IOSXEConfigException**(***kw*

Bases: *networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.DriverException*

Configuration exception thrown when modifying the running config.

message = 'Error executing snippet:%(snippet)s. Hosting device:%(dev_id)s Mgmt IP:%(ip

exception networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.**IOSXEMissingInterfaceExce**
Bases: *networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.DriverException*

Configuration exception thrown when modifying the running config.

message = 'Interface corresponding to port:%(id)s and mac-address:%(mac)s is missing i

exception networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.**IOSXEUnknownValueExceptio**
Bases: *networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.DriverException*

IOS XE device exception thrown when an unknown value is received.

message = 'Data in attribute: %(attribute)s does not correspond to expected value. Va

exception networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.**InitializationException**(
Bases: *networking_cisco.plugins.cisco.cfg_agent.cfg_exceptions.DriverException*

Exception when initialization of Routing Driver object.

message = 'Critical device parameter missing. Failed initializing routing driver objec

networking_cisco.plugins.cisco.cfg_agent.device_status module

class networking_cisco.plugins.cisco.cfg_agent.device_status.**DeviceStatus**
Bases: object

Device status and backlog processing.

backlog_hosting_device(*hosting_device*)

backlog_hosting_devices(*hosting_devices*)

check_backlogged_hosting_devices(*driver_mgr*)
Checks the status of backlogged hosting devices.

Skips newly spun up instances during their booting time as specified in the boot time parameter.

Each hosting-device tracked has a key, hd_state, that represents the last known state for the hosting device. Valid values for hd_state are ['Active', 'Unknown', 'Dead']

Each time check_backlogged_hosting_devices is invoked, a ping-test is performed to determine the current state. If the current state differs, hd_state is updated.

The hd_state transitions/actions are represented by the following table.

current / last state	Active	Unknown	Dead
Active	Device is reachable. No state change	Device was temporarily unreachable.	Dead device recovered. Trigger resync
Unknown	Device connectivity test failed. Set backlog timestamp and wait for dead timeout to occur.	Device connectivity test failed. Dead timeout has not occurred yet.	Not a valid state transition.
Dead	Not a valid state transition.	Dead timeout for device has elapsed. Notify plugin	Device is still dead. No state change.

Returns A dict of the format:

```
{"reachable": [<hd_id>,..],
"dead": [<hd_id>,..],
"revived": [<hd_id>,..]}
```

- reachable - a list of hosting devices that are now reachable
- dead - a list of hosting devices deemed dead
- revived - a list of hosting devices (dead to active)

get_backlogged_hosting_devices()

get_backlogged_hosting_devices_info()

get_dead_hosting_devices_info()
Get a list of hosting devices that have been marked dead :return: List of dead hosting device ids

get_monitored_hosting_devices_info(*hd_state_filter=None*)
This function returns a list of all hosting devices monitored by this agent

is_hosting_device_reachable(*hosting_device*)
Check the hosting device which hosts this resource is reachable.

If the resource is not reachable, it is added to the backlog.

- heartbeat revision

We want to enqueue all hosting-devices into the backlog for monitoring purposes

adds key/value pairs to hd (aka hosting_device dictionary)

_is_pingable [if it returns true,] hd['hd_state']='Active'

_is_pingable [if it returns false,] hd['hd_state']='Unknown'

:param hosting_device : dict of the hosting device :returns: True if device is reachable, else None

Module contents

networking_cisco.plugins.cisco.common package

Submodules

networking_cisco.plugins.cisco.common.cisco_constants module

networking_cisco.plugins.cisco.common.cisco_ios_xe_simulator module

class networking_cisco.plugins.cisco.common.cisco_ios_xe_simulator.**CiscoIOSXESimulator**(*path*, *host_*, *net-mask*, *port*, *user-name*, *pass-word*, *de-vice_mgm*, *time-out*)

Bases: object

edit_config(*snippet*)

exclamation = {'vrf definition', ' exit-address-family'}

get_config()

log_only_commands = set()

parent_bound_commands = {'exit-address-family'}

class networking_cisco.plugins.cisco.common.cisco_ios_xe_simulator.**FakeRunningConfig**(*rc*)

Bases: object

networking_cisco.plugins.cisco.common.htparser module

class networking_cisco.plugins.cisco.common.htparser.**HTParser**(*cfg*)

Bases: object

A simple hierarchical text parser.

Indents in the text are used to derive parent child hierarchy.

find_children(*linespec*)

Find lines and immediate children that match the linespec regex.

Parameters **linespec** – regular expression of line to match

Returns list of lines. These correspond to the lines that were matched and their immediate children

find_lines(*linespec*)

Find lines that match the linespec regex.

find_objects(*linespec*)
Find lines that match the linespec regex.

Parameters **linespec** – regular expression of line to match

Returns list of LineItem objects

class networking_cisco.plugins.cisco.common.htparser.**LineItem**(*line*)
Bases: object

add_children(*child*)

re_match(*regex*, *group=1*, *default=''*)

re_search_children(*linespec*)

str_list()

networking_cisco.plugins.cisco.common.utils module

exception networking_cisco.plugins.cisco.common.utils.**DriverNotFound**(***kwargs*)
Bases: neutron_lib.exceptions.NotFound

message = 'Driver %(driver)s does not exist'

networking_cisco.plugins.cisco.common.utils.**convert_validate_driver_class**(*driver_class_name*)

networking_cisco.plugins.cisco.common.utils.**retry**(*ExceptionToCheck*, *tries=4*, *delay=3*, *backoff=2*)
Retry calling the decorated function using an exponential backoff.

Reference: http://www.saltycrane.com/blog/2009/11/trying-out-retry -decorator-python/

Parameters

- **ExceptionToCheck** – the exception to check. may be a tuple of exceptions to check
- **tries** – number of times to try (not retry) before giving up
- **delay** – initial delay between retries in seconds
- **backoff** – backoff multiplier e.g. value of 2 will double the delay each retry

Module contents

networking_cisco.plugins.cisco.cpnr package

Submodules

networking_cisco.plugins.cisco.cpnr.cpnr_client module

exception networking_cisco.plugins.cisco.cpnr.cpnr_client.**ConnectionError**(***kwargs*)
Bases: *networking_cisco.plugins.cisco.cpnr.cpnr_client.CpnrException*

message = 'CPNR failed to connect: %(msg)s'

class networking_cisco.plugins.cisco.cpnr.cpnr_client.**CpnrClient**(*scheme*, *address*, *port*, *username*, *password*, *insecure*, *timeout=20*)

Bases: object

Class implementing REST APIs for CPNR Server.

create_ccm_host(*data*, *viewid=None*, *zoneid=None*)

create_ccm_reverse_zone(*data*, *viewid=None*)

create_ccm_zone(*data*, *viewid=None*)

create_client_class(*data*)

create_client_entry(*data*)

create_dns_forwarder(*data*)

create_dns_view(*data*)

create_scope(*data*)

create_vpn(*data*)

delete_ccm_host(*name*, *viewid=None*, *zoneid=None*)

delete_ccm_reverse_zone(*name*, *viewid=None*)

delete_ccm_zone(*name*, *viewid=None*)

delete_client_class(*client_class_name*)

delete_client_entry(*client_entry_name*)

delete_dns_forwarder(*name*)

delete_dns_view(*name*)

delete_scope(*scope_name*)

delete_vpn(*vpn_name*)

get_ccm_host(*name*, *viewid='.*'*, *zoneid='.*'*)

get_ccm_hosts(*viewid='.*'*, *zoneid='.*'*)

get_ccm_reverse_zone(*name*, *viewid='.*'*)

get_ccm_reverse_zones(*viewid='.*'*)

get_ccm_zone(*name*, *viewid='.*'*)

get_ccm_zones(*viewid='.*'*)

get_client_class(*client_class_name*)
Returns a specific client class details from CPNR server.

get_client_classes()
Returns a list of all the client classes from CPNR server.

get_client_entries()
Returns a list of all the client entries from CPNR server.

get_client_entry(*client_entry_name*)
Returns a specific client entry name details from CPNR server.

get_dhcp_server()
Returns a dictionary with all the objects of DHCP server.

get_dns_forwarder(*name*)

get_dns_forwarders()

get_dns_server()

get_dns_view(*name*)

get_dns_views()

get_leases(*vpnid='.*'*)

get_scope(*scope_name*)
Returns a specific scope name details from CPNR server.

get_scopes(*vpnid='.*'*)
Returns a list of all the scopes from CPNR server.

get_version()

get_vpn(*vpn_name*)
Returns a specific VPN name details from CPNR server.

get_vpns()
Returns a list of all the VPNs from CPNR server.

release_address(*address*, *vpnid*)
Release a specific lease, called after delete_client_entry

reload_dhcp_server()

reload_dns_server()

reload_needed()

reload_server(*force_reload=False*)

update_ccm_host(*name*, *data*, *viewid=None*, *zoneid=None*)

update_ccm_reverse_zone(*name*, *data*, *viewid=None*)

update_ccm_zone(*name*, *data*, *viewid=None*)

update_client_class(*client_class_name*, *data*)

update_client_entry(*client_entry_name*, *data*)

update_dhcp_server(*data*)

update_dns_forwarder(*name*, *data*)

update_dns_server(*data*)

update_dns_view(*name*, *data*)

update_scope(*scope_name*, *data*)

update_vpn(*vpn_name*, *data*)

exception networking_cisco.plugins.cisco.cpnr.cpnr_client.**CpnrException**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

message = 'CPNR exception occurred'

exception networking_cisco.plugins.cisco.cpnr.cpnr_client.**ServerError**(***kwargs*)
Bases: *networking_cisco.plugins.cisco.cpnr.cpnr_client.CpnrException*

message = 'CPNR received error response: %(status)i %(msg)s'

exception networking_cisco.plugins.cisco.cpnr.cpnr_client.**Timeout**(***kwargs*)
Bases: *networking_cisco.plugins.cisco.cpnr.cpnr_client.CpnrException*

message = 'CPNR callout to server timed out: %(msg)s'

exception networking_cisco.plugins.cisco.cpnr.cpnr_client.**UnexpectedError**(***kwargs*)
Bases: *networking_cisco.plugins.cisco.cpnr.cpnr_client.CpnrException*

message = 'CPNR unexpected error: %(msg)s'

networking_cisco.plugins.cisco.cpnr.cpnr_dhcp_relay_agent module

class networking_cisco.plugins.cisco.cpnr.cpnr_dhcp_relay_agent.**DhcpPacket**
Bases: object

data()

get_ciaddr()

get_relay_option(*code*)

classmethod parse(*buf*)
Parse DHCP Packet.

1. To get client IP Address(ciaddr).
2. To get relaying gateway IP Address(giaddr).
3. **To get DHCP Relay Agent Information Option Suboption** such as Link Selection, VSS, Server Identifier override.

set_giaddr(*addr*)

set_relay_option(*code*, *value*)

classmethod struct(*fmt*)

structcache = {}

class networking_cisco.plugins.cisco.cpnr.cpnr_dhcp_relay_agent.**DhcpRelayAgent**
Bases: object

Relay DHCP packets between neutron networks and external DHCP server.

Receives broadcast and unicast DHCP requests via sockets which are opened in each neutron dhcp network namespace. Additional DHCP options are appended to the request to indicate from which network the request originated. Requests are then forwarded to the configured DHCP server address.

Receives unicast DHCP responses from the DHCP server via socket opened in the global network namespace. Additional options are stripped from the response. The response is then forwarded to the originating network.

serve()

networking_cisco.plugins.cisco.cpnr.cpnr_dhcp_relay_agent.**main**()

networking_cisco.plugins.cisco.cpnr.cpnr_dhcp_relay_service module

networking_cisco.plugins.cisco.cpnr.cpnr_dhcp_relay_service.**main**()

networking_cisco.plugins.cisco.cpnr.cpnr_dns_relay_agent module

class networking_cisco.plugins.cisco.cpnr.cpnr_dns_relay_agent.**DnsPacket**
Bases: object

COUNTS_LENGTH = 8

IDENTIFIER_FLAGS_AND_CODES_LENGTH = 4

OPTIONAL_RR = 41

QUERY_TYPE_AND_CLASS = 4

TXT_RR = bytearray(b'\n_cpnr_info\x05cisco\x03com\x00\x00\x10\x00\x01\x00\x00\x00\x00'

TYPE_CLASS_AND_TTL_LENGTH = 8

data()

get_msgid()

classmethod parse(*buf*, *buflen*)

set_viewid(*id*)

classmethod skip_over_domain_name(*buf*, *pos*)

classmethod struct(*fmt*)

structcache = {}

class networking_cisco.plugins.cisco.cpnr.cpnr_dns_relay_agent.**DnsRelayAgent**
Bases: object

Relay DNS packets between neutron networks and external DNS server.

Receives unicast DNS requests via sockets which are opened in each neutron network namespace. Additional DNS options are appended to the request to indicate from which network the request originated. Requests are then forwarded to the configured DNS server address.

Receives unicast DNS responses from the DNS server via socket opened in the global network namespace. Additional options are stripped from the response. The response is then forwarded to the originating network.

serve()

networking_cisco.plugins.cisco.cpnr.cpnr_dns_relay_agent.**main**()

networking_cisco.plugins.cisco.cpnr.cpnr_dns_relay_service module

networking_cisco.plugins.cisco.cpnr.cpnr_dns_relay_service.**main**()

networking_cisco.plugins.cisco.cpnr.debug_stats module

class networking_cisco.plugins.cisco.cpnr.debug_stats.**DebugStats**(*stats_type*)
Bases: object

add_network_stats(*net_id*)

del_network_stats(*net_id*)

increment_pkts_from_client(*net_id*)

increment_pkts_from_server(*net_id*)

increment_pkts_to_client(*net_id*)

increment_pkts_to_server(*net_id*)

write_stats_to_file()

class networking_cisco.plugins.cisco.cpnr.debug_stats.**PacketStats**(*net_id*)
Bases: object

write_to_file(*f*)

networking_cisco.plugins.cisco.cpnr.dhcp_driver module

class networking_cisco.plugins.cisco.cpnr.dhcp_driver.**CpnrDriver**(*conf*, *network*, *root_helper='sudo'*, *version=None*, *plugin=None*)
Bases: *networking_cisco.plugins.cisco.cpnr.dhcp_driver.SimpleCpnrDriver*

classmethod check_version()

disable(*retain_port=False*)

enable()

reload_allocations()

restart()

classmethod start_threads()

update_device(*disabled=False*)

update_server(*disabled=False*)

class networking_cisco.plugins.cisco.cpnr.dhcp_driver.**RemoteServerDriver**(*conf*, *network*, *root_helper='sudo'*, *version=None*, *plugin=None*)
Bases: neutron.agent.linux.dhcp.DhcpBase

active

classmethod check_version()

disable(*retain_port=False*)
Teardown DHCP.

Disable DHCP for this network by updating the remote server and then destroying any local device and namespace.

enable()
Setup DHCP.

Enables DHCP for this network by updating the remote server and then sets up a local device for relaying DHCP requests.

classmethod existing_dhcp_networks(*conf*)
Return a list of existing networks ids that we have configs for.

classmethod get_isolated_subnets(*network*)
Return a indicator whether or not a subnet is isolated.

classmethod recover_devices()
Track devices.

Creates global dict to track device names across driver invocations and populates based on current devices configured on the system.

classmethod recover_networks()
Track Network Objects.

Creates global dict to track network objects across driver invocations and populates using the model module.

reload_allocations()
Reload DHCP.

Reload DHCP for this network by updating remote server and updating local device, if subnets have changed.

restart()
Restart DHCP.

Restart DHCP for this network by updating remote server and re-establishing local device if necessary.

classmethod should_enable_metadata(*conf*, *network*)
Determine whether the metadata proxy is needed for a network.

update_device(*disabled=False*)

update_server(*disabled=False*)

class networking_cisco.plugins.cisco.cpnr.dhcp_driver.**SimpleCpnrDriver**(*conf*, *network*, *root_helper='sudo'*, *version=None*, *plugin=None*)
Bases: *networking_cisco.plugins.cisco.cpnr.dhcp_driver.RemoteServerDriver*

MIN_VERSION = 8.3

classmethod check_version()
Checks server version against minimum required version.

classmethod existing_dhcp_networks(*conf*)
Return a list of existing networks ids that we have configs for.

update_server(*disabled=False*)

networking_cisco.plugins.cisco.cpnr.dhcpopts module

networking_cisco.plugins.cisco.cpnr.dhcpopts.**format_for_options**(*name*, *value*)

networking_cisco.plugins.cisco.cpnr.dhcpopts.**format_for_pnr**(*name*, *value*)

networking_cisco.plugins.cisco.cpnr.model module

class networking_cisco.plugins.cisco.cpnr.model.**ClientEntry**(*data=None*, *vpnid='0'*)

Bases: object

create()

delete()

classmethod from_neutron(*network*, *port*)

classmethod from_pnr(*ce*)

update(*new*)

class networking_cisco.plugins.cisco.cpnr.model.**ForwardZone**(*data=None*)

Bases: object

create(*retry_count=10*)

delete()

classmethod from_neutron(*network*)

classmethod from_pnr(*zone*)

update(*new*)

class networking_cisco.plugins.cisco.cpnr.model.**Host**(*data=None*, *viewid=0*)

Bases: object

static addr_to_hostname(*addr*)

create()

delete()

classmethod from_neutron(*network*, *addr*)

classmethod from_pnr(*host*, *viewid*)

update(*new*)

class networking_cisco.plugins.cisco.cpnr.model.**Network**

Bases: object

create()

delete()

static filter_ipv4_subnets(*subnets*)

classmethod from_neutron(*network*)

update(*new*)

class networking_cisco.plugins.cisco.cpnr.model.**Policy**(*data=None*)

Bases: object

classmethod from_neutron_port(*network*, *port*)

classmethod from_neutron_subnet(*network*, *subnet*)

class networking_cisco.plugins.cisco.cpnr.model.**ReverseZone**(*data=None*)

Bases: object

create()

delete()

classmethod from_neutron(*network*, *subnet*)

classmethod from_pnr(*rzone*)

update(*new*)

class networking_cisco.plugins.cisco.cpnr.model.**Scope**(*data=None*)
Bases: object

create()

delete()

classmethod from_neutron(*network*, *subnet*)

classmethod from_pnr(*scope*)

update(*new*)

class networking_cisco.plugins.cisco.cpnr.model.**View**(*data=None*)
Bases: object

create()

delete()

classmethod from_neutron(*network*)

classmethod from_pnr(*view*)

static net_to_view_id(*netid*)

update(*new*)

class networking_cisco.plugins.cisco.cpnr.model.**Vpn**(*data=None*)
Bases: object

create()

delete()

classmethod from_neutron(*network*)

classmethod from_pnr(*vpn*)

static net_to_vpn_id(*netid*)

static net_to_vpn_rfc(*netid*)

update(*new*)

networking_cisco.plugins.cisco.cpnr.model.**configure_pnr**()

networking_cisco.plugins.cisco.cpnr.model.**get_version**()

networking_cisco.plugins.cisco.cpnr.model.**recover_networks**()

networking_cisco.plugins.cisco.cpnr.model.**reload_needed**()

networking_cisco.plugins.cisco.cpnr.model.**reload_server**(*timeout=120*)

networking_cisco.plugins.cisco.cpnr.netns module

netns - context manager for network namespaces

class networking_cisco.plugins.cisco.cpnr.netns.**Namespace**(*name*)
Bases: object

networking_cisco.plugins.cisco.cpnr.netns.**iflist**(*ignore=set()*)

networking_cisco.plugins.cisco.cpnr.netns.**increase_ulimit**(*ulimit*)

networking_cisco.plugins.cisco.cpnr.netns.**nslist**()

Module contents

networking_cisco.plugins.cisco.db package

Subpackages

networking_cisco.plugins.cisco.db.device_manager package

Submodules

networking_cisco.plugins.cisco.db.device_manager.hd_models module

class networking_cisco.plugins.cisco.db.device_manager.hd_models.**HostedHostingPortBinding**(*
Bases: sqlalchemy.ext.declarative.api.Base

Represents binding of logical resource's port to its hosting port.

hosting_port

hosting_port_id

logical_port

logical_port_id

logical_resource_id

network_type

port_type

segmentation_id

class networking_cisco.plugins.cisco.db.device_manager.hd_models.**HostingDevice**(**kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base, neutron_lib.db.model_base.HasId, neutron_lib.db.model_base.HasProject

Represents an appliance hosting Neutron router(s).

When the hosting device is a Nova VM 'id' is uuid of that VM.

admin_state_up

auto_delete

cfg_agent

cfg_agent_id

complementary_id

created_at

credentials_id

description

device_id

id

management_ip_address

management_port

management_port_id

name

project_id

protocol_port

status

template

template_id

tenant_bound

tenant_id

class networking_cisco.plugins.cisco.db.device_manager.hd_models.**HostingDeviceTemplate**(***kw*

Bases: sqlalchemy.ext.declarative.api.Base, neutron_lib.db.model_base.HasId, neutron_lib.db.model_base.HasProject

Represents a template for devices used to host service.

Such devices may be physical or virtual.

booting_time

configuration_mechanism

default_credentials_id

desired_slots_free

device_driver

enabled

flavor

host_category

id

image

name

plugging_driver

project_id

protocol_port

service_types

slot_capacity

tenant_bound

tenant_id

class networking_cisco.plugins.cisco.db.device_manager.hd_models.**SlotAllocation**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Tracks allocation of slots in hosting devices.

hosting_device_id

logical_resource_id

logical_resource_owner

logical_resource_service

logical_resource_type

num_allocated

template_id

tenant_bound

networking_cisco.plugins.cisco.db.device_manager.hosting_device_manager_db module

class networking_cisco.plugins.cisco.db.device_manager.hosting_device_manager_db.**HostingDev**
Bases: *networking_cisco.plugins.cisco.db.device_manager.hosting_devices_db.HostingDeviceDBMixin*

A class implementing a resource manager for hosting devices.

The caller should make sure that HostingDeviceManagerMixin is a singleton.

acquire_hosting_device_slots(*context*, *hosting_device*, *resource*, *resource_type*, *resource_service*, *num*, *exclusive=False*)
Assign <num> slots in <hosting_device> to logical <resource>.

If exclusive is True the hosting device is bound to the resource's tenant. Otherwise it is not bound to any tenant.

Returns True if allocation was granted, False otherwise.

delete_all_hosting_devices(*context*, *force_delete=False*)
Deletes all hosting devices.

delete_all_hosting_devices_by_template(*context*, *template*, *force_delete=False*)
Deletes all hosting devices based on <template>.

get_device_info_for_agent(*context*, *hosting_device_db*)
Returns information about <hosting_device> needed by config agent.

Convenience function that service plugins can use to populate their resources with information about the device hosting their logical resource.

get_hosting_device_config(*context*, *id*)

get_hosting_device_driver(*context*, *id*)
Returns device driver for hosting device template with <id>.

get_hosting_device_plugging_driver(*context*, *id*)
Returns plugging driver for hosting device template with <id>.

get_hosting_devices_qry(*context*, *hosting_device_ids*, *load_agent=True*)
Returns hosting devices with <hosting_device_ids>.

get_slot_allocation(*context*, *template_id=None*, *hosting_device_id=None*, *resource_id=None*)

handle_non_responding_hosting_devices(*context*, *cfg_agent*, *hosting_device_ids*)

classmethod l3_tenant_id()
Returns id of tenant owning hosting device resources.

classmethod mgmt_nw_id()
Returns id of the management network.

classmethod mgmt_sec_grp_id()
Returns id of security group used by the management network.

classmethod mgmt_subnet_id()

release_hosting_device_slots(*context*, *hosting_device*, *resource*, *num*)
Free <num> slots in <hosting_device> from logical resource <id>.

Returns True if deallocation was successful. False otherwise.

report_hosting_device_shortage(*context*, *template*, *requested=0*)
Used to report shortage of hosting devices based on <template>.

svc_vm_mgr

networking_cisco.plugins.cisco.db.device_manager.hosting_devices_db module

class networking_cisco.plugins.cisco.db.device_manager.hosting_devices_db.**HostingDeviceDBM**
Bases: *networking_cisco.plugins.cisco.extensions.ciscohostingdevicemanager.CiscoHostingDevicePluginBase*, neutron.db.common_db_mixin.CommonDbMixin

A class implementing DB functionality for hosting devices.

create_hosting_device(*context*, *hosting_device*)

create_hosting_device_template(*context*, *hosting_device_template*)

delete_hosting_device(*context*, *id*)

delete_hosting_device_template(*context*, *id*)

get_hosting_device(*context*, *id*, *fields=None*)

get_hosting_device_template(*context*, *id*, *fields=None*)

get_hosting_device_templates(*context*, *filters=None*, *fields=None*, *sorts=None*, *limit=None*, *marker=None*, *page_reverse=False*)

get_hosting_devices(*context*, *filters=None*, *fields=None*, *sorts=None*, *limit=None*, *marker=None*, *page_reverse=False*)

get_hosting_devices_db(*context*, *filters=None*, *sorts=None*, *limit=None*, *marker=None*, *page_reverse=False*)

update_hosting_device(*context*, *id*, *hosting_device*)

update_hosting_device_template(*context*, *id*, *hosting_device_template*)

Module contents

networking_cisco.plugins.cisco.db.l3 package

Submodules

networking_cisco.plugins.cisco.db.l3.ha_db module

class networking_cisco.plugins.cisco.db.l3.ha_db.**HA_db_mixin**
Bases: object

Mixin class to support VRRP, HSRP, and GLBP based HA for routing.

get_router_for_floatingip(*context*, *internal_port*, *internal_subnet*, *external_network_id*)
We need to over-load this function so that we only return the user visible router and never its redundancy routers (as they never have floatingips associated with them).

class networking_cisco.plugins.cisco.db.l3.ha_db.**RouterHAGroup**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base, neutron_lib.db.model_base.HasId, neutron_lib.db.model_base.HasProject

Represents an HA group as used in VRRP, HSRP, and GLBP.

extra_port

extra_port_id

group_identity

ha_port

ha_port_id

ha_type

id

other_config

project_id

subnet_id

tenant_id

timers_config

tracking_config

user_router_id

class networking_cisco.plugins.cisco.db.l3.ha_db.**RouterHASetting**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents HA settings for router visible to user.

ha_type

priority

probe_connectivity

probe_interval

probe_target

redundancy_level

router

router_id

state

class networking_cisco.plugins.cisco.db.l3.ha_db.**RouterRedundancyBinding**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents binding between an HA enabled router and its redundancy routers.

priority

redundancy_router

redundancy_router_id

state

user_router

user_router_id

networking_cisco.plugins.cisco.db.l3.l3_models module

class networking_cisco.plugins.cisco.db.l3.l3_models.**RouterHostingDeviceBinding**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Represents binding between Neutron routers and their hosting devices.

auto_schedule

hosting_device

hosting_device_id

inflated_slot_need

role

router

router_id

router_type

router_type_id

share_hosting_device

class networking_cisco.plugins.cisco.db.l3.l3_models.**RouterType**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base, neutron_lib.db.model_base.HasId, neutron_lib.db.model_base.HasProject

Represents Neutron router types.

A router type is associated with a with hosting device template. The template is used when hosting device for the router type is created.

Only 'id', 'name', 'description' are visible in non-admin context.

cfg_agent_driver

cfg_agent_service_helper

description

driver

ha_enabled_by_default

id

name

project_id

scheduler

shared

slot_need

template

template_id

tenant_id

networking_cisco.plugins.cisco.db.l3.routertype_db module

class networking_cisco.plugins.cisco.db.l3.routertype_db.**RoutertypeDbMixin**
Bases: *networking_cisco.plugins.cisco.extensions.routertype.RoutertypePluginBase*

Mixin class for Router types.

create_routertype(*context*, *routertype*)
Creates a router type.

Also binds it to the specified hosting device template.

delete_routertype(*context*, *id*)

get_routertype(*context*, *id*, *fields=None*)

get_routertype_by_id_name(*context*, *id_or_name*)

get_routertype_db_by_id_name(*context*, *id_or_name*)

get_routertypes(*context*, *filters=None*, *fields=None*, *sorts=None*, *limit=None*, *marker=None*, *page_reverse=False*)

update_routertype(*context*, *id*, *routertype*)

Module contents

networking_cisco.plugins.cisco.db.scheduler package

Submodules

networking_cisco.plugins.cisco.db.scheduler.cfg_agentschedulers_db module

class networking_cisco.plugins.cisco.db.scheduler.cfg_agentschedulers_db.**CfgAgentSchedulerI**
Bases: *networking_cisco.plugins.cisco.extensions.ciscocfgagentscheduler.*

CfgAgentSchedulerPluginBase, neutron.db.agentschedulers_db.AgentSchedulerDbMixin

Mixin class to add cfg agent scheduler extension.

assign_hosting_device_to_cfg_agent(*context*, *cfg_agent_id*, *hosting_device_id*)
Make config agent handle an (unassigned) hosting device.

auto_schedule_hosting_devices(*context*, *host*)

cfg_agent_scheduler = None

get_cfg_agents(*context*, *active=None*, *filters=None*)

get_cfg_agents_for_hosting_devices(*context*, *hosting_device_ids*, *admin_state_up=None*, *schedule=False*)

classmethod is_agent_down(*heart_beat_time*, *timeout=30*)

list_cfg_agents_handling_hosting_device(*context*, *hosting_device_id*)

list_hosting_devices_handled_by_cfg_agent(*context*, *cfg_agent_id*)

set_monitor_timestamp(*agent*, *timestamp*)

classmethod should_check_agent(*heart_beat_time*, *timeout=20*)

unassign_hosting_device_from_cfg_agent(*context*, *cfg_agent_id*, *hosting_device_id*)
Make config agent handle an (unassigned) hosting device.

networking_cisco.plugins.cisco.db.scheduler.l3_routertype_aware_schedulers_db module

class networking_cisco.plugins.cisco.db.scheduler.l3_routertype_aware_schedulers_db.**L3Route**
Bases: neutron.db.l3_agentschedulers_db.L3AgentSchedulerDbMixin

Mixin class to add L3 router type-aware scheduler capability.

This class can schedule Neutron routers to hosting devices and to L3 agents on network nodes.

add_router_to_hosting_device(*context*, *hosting_device_id*, *router_id*)
Add a (non-hosted) router to a hosting device.

cfg_list_router_ids_on_host(*context*, *host*, *router_ids=None*, *hosting_device_ids=None*)

get_active_routers_for_host(*context*, *host*)

get_hosts_for_routers(*context*, *routers*, *admin_state_up=None*, *check_active=False*)

get_number_of_agents_for_scheduling(*context*)
Return number of agents on which the router will be scheduled.

list_active_sync_routers_on_hosting_devices(*context*, *host*, *router_ids=None*, *hosting_device_ids=None*)

list_all_routers_on_hosting_devices(*context*)

list_hosting_devices_hosting_router(*context*, *router_id*)

list_routers_on_hosting_device(*context*, *hosting_device_id*)

remove_router_from_hosting_device(*context*, *hosting_device_id*, *router_id*)
Remove the router from hosting device.

After removal, the router will be non-hosted until there is update which leads to re-schedule or be added to another hosting device manually.

validate_hosting_device_router_combination(*context*, *binding_info*, *hosting_device_id*)

Module contents

Module contents

networking_cisco.plugins.cisco.device_manager package

Subpackages

networking_cisco.plugins.cisco.device_manager.hosting_device_drivers package

Submodules

networking_cisco.plugins.cisco.device_manager.hosting_device_drivers.noop_hd_driver module

class networking_cisco.plugins.cisco.device_manager.hosting_device_drivers.noop_hd_driver.**N**

Bases: *networking_cisco.plugins.cisco.device_manager.hosting_device_drivers.HostingDeviceDriver*

create_config(*context*, *credentials_info*, *connectivity_info*)

hosting_device_name()

Module contents

class networking_cisco.plugins.cisco.device_manager.hosting_device_drivers.**HostingDeviceDr**

Bases: object

This class defines the API for hosting device drivers.

These are used by Cisco (routing service) plugin to perform various (plugin independent) operations on hosting devices.

create_config(*context*, *credentials_info*, *connectivity_info*)

Creates configuration(s) for a service VM.

This function can be used to make initial configurations. The configuration(s) is/are injected in the VM's file system using Nova's configdrive feature.

Called when a service VM-based hosting device is to be created. This function should cleanup after itself in case of error.

Parameters

- **context** – contains user information
- **credentials_info** – dictionary with login credentials to be injected into the hosting device:

```
{'user_name': <user name>,
 'password': <password>}
```

- **connectivity_info** – dictionary with management connectivity information needed by hosting device to communicate:

```
{'mgmt_port': <neutron port for management>,
 'gateway_ip': <gateway ip address of management subnet>,
 'netmask': <netmask of management subnet>
 'name_server_1: <ip of domain name server 1>,
 'name_server_2: <ip of domain name server 2>}
```

Returns Dict with file names and their corresponding content strings: {filename1: content_string1, filename2: content_string2, ... } The file system of the VM will contain files with the specified file names and content. If the dict is empty no config drive will be used.

hosting_device_name()

networking_cisco.plugins.cisco.device_manager.plugging_drivers package

Submodules

networking_cisco.plugins.cisco.device_manager.plugging_drivers.aci_vlan_trunking_driver module

exception networking_cisco.plugins.cisco.device_manager.plugging_drivers.aci_vlan_trunking_
Bases: neutron_lib.exceptions.BadRequest

message = 'The ACI Driver config file format is invalid'

exception networking_cisco.plugins.cisco.device_manager.plugging_drivers.aci_vlan_trunking_
Bases: neutron_lib.exceptions.BadRequest

message = 'The ACI Driver config is missing a cidr_exposed parameter for %(ext_net)s.'

exception networking_cisco.plugins.cisco.device_manager.plugging_drivers.aci_vlan_trunking_
Bases: neutron_lib.exceptions.BadRequest

message = 'The ACI Driver config is missing a gateway_ip parameter for %(ext_net)s.'

exception networking_cisco.plugins.cisco.device_manager.plugging_drivers.aci_vlan_trunking_
Bases: neutron_lib.exceptions.BadRequest

message = 'The ACI Driver config is missing a segmentation_id parameter for %(ext_net)

exception networking_cisco.plugins.cisco.device_manager.plugging_drivers.aci_vlan_trunking_
Bases: neutron_lib.exceptions.BadRequest

message = 'The ACI plugin driver is either not installed or the neutron configuration

class networking_cisco.plugins.cisco.device_manager.plugging_drivers.aci_vlan_trunking_driv
Bases: *networking_cisco.plugins.cisco.device_manager.plugging_drivers.hw_vlan_trunking_driver.HwVLANTrunkingPlugDriver*

Driver class for Cisco ACI-based devices.

The driver works with VLAN segmented Neutron networks. It determines which workflow is active (GBP or Neutron), and uses that implementation to get the information needed for the networks between the hosting device and the ACI fabric.

allocate_hosting_port(*context*, *router_id*, *port_db*, *network_type*, *hosting_device_id*)
Get the VLAN and port for this hosting device

The VLAN used between the APIC and the external router is stored by the APIC driver. This calls into the APIC driver to first get the ACI VRF information associated with this port, then uses that to look up the VLAN to use for this port to the external router (kept as part of the L3 Out policy in ACI).

apic_driver
Get APIC driver

There are different drivers for the GBP workflow and Neutron workflow for APIC. First see if the GBP workflow is active, and if so get the APIC driver for it. If the GBP service isn't installed, try to get the driver from the Neutron (APIC ML2) workflow.

extend_hosting_port_info(*context*, *port_db*, *hosting_device*, *hosting_info*)
Get the segmenetation ID and interface

This extends the hosting info attribute with the segmentation ID and physical interface used on the external router to connect to the ACI fabric. The segmentation ID should have been set already by the call to allocate_hosting_port, but if it's not present, use the value from the port resource.

get_ext_net_name

get_vrf_context

l3_plugin

transit_nets_cfg

networking_cisco.plugins.cisco.device_manager.plugging_drivers.hw_vlan_trunking_driver module

class networking_cisco.plugins.cisco.device_manager.plugging_drivers.hw_vlan_trunking_drive
Bases: *networking_cisco.plugins.cisco.device_manager.plugging_drivers.PluginSidePluggingDriver*

Driver class for Cisco hardware-based devices.

The driver works with VLAN segmented Neutron networks.

allocate_hosting_port(*context*, *router_id*, *port_db*, *network_type*, *hosting_device_id*)

create_hosting_device_resources(*context*, *complementary_id*, *tenant_id*, *mgmt_context*, *max_hosted*)

delete_hosting_device_resources(*context*, *tenant_id*, *mgmt_port*, ***kwargs*)

extend_hosting_port_info(*context*, *port_db*, *hosting_device*, *hosting_info*)

get_hosting_device_resources(*context*, *id*, *complementary_id*, *tenant_id*, *mgmt_nw_id*)

setup_logical_port_connectivity(*context*, *port_db*, *hosting_device_id*)

teardown_logical_port_connectivity(*context*, *port_db*, *hosting_device_id*)

networking_cisco.plugins.cisco.device_manager.plugging_drivers.noop_plugging_driver module

class networking_cisco.plugins.cisco.device_manager.plugging_drivers.noop_plugging_driver.N
Bases: *networking_cisco.plugins.cisco.device_manager.plugging_drivers.PluginSidePluggingDriver*, *networking_cisco.plugins.cisco.device_manager.plugging_drivers.utils.PluggingDriverUtilsMixin*

This class defines a no-op plugging driver.

allocate_hosting_port(*context*, *router_id*, *port_db*, *network_type*, *hosting_device_id*)

create_hosting_device_resources(*context*, *complementary_id*, *tenant_id*, *mgmt_context*, *max_hosted*)

delete_hosting_device_resources(*context*, *tenant_id*, *mgmt_port*, ***kwargs*)

extend_hosting_port_info(*context*, *port_db*, *hosting_device*, *hosting_info*)

get_hosting_device_resources(*context*, *id*, *complementary_id*, *tenant_id*, *mgmt_nw_id*)

setup_logical_port_connectivity(*context*, *port_db*, *hosting_device_id*)

teardown_logical_port_connectivity(*context*, *port_db*, *hosting_device_id*)

networking_cisco.plugins.cisco.device_manager.plugging_drivers.utils module

class networking_cisco.plugins.cisco.device_manager.plugging_drivers.utils.**PluggingDriverUt**

Bases: object

networking_cisco.plugins.cisco.device_manager.plugging_drivers.utils.**retry**(*ExceptionToCheck*, *tries=4*, *delay=3*, *backoff=2*)

Retry calling the decorated function using an exponential backoff. Reference: http://www.saltycrane.com/blog/2009/11/trying-out-retry -decorator-python/

Parameters

- **ExceptionToCheck** – the exception to check. may be a tuple of exceptions to check
- **tries** – number of times to try (not retry) before giving up
- **delay** – initial delay between retries in seconds
- **backoff** – backoff multiplier e.g. value of 2 will double the delay each retry

networking_cisco.plugins.cisco.device_manager.plugging_drivers.vif_hotplug_plugging_driver module

exception networking_cisco.plugins.cisco.device_manager.plugging_drivers.vif_hotplug_pluggi

Bases: neutron_lib.exceptions.InUse

message = 'Port: %(port_id)s not unbound yet.'

class networking_cisco.plugins.cisco.device_manager.plugging_drivers.vif_hotplug_plugging_d

Bases: *networking_cisco.plugins.cisco.device_manager.plugging_drivers.PluginSidePluggingDriver*, *networking_cisco.plugins.cisco.device_manager.plugging_drivers.utils.PluggingDriverUtilsMixin*

Driver class for service VMs used with Neutron plugins supporting VIF hot-plug.

allocate_hosting_port(*context*, *router_id*, *port_db*, *network_type*, *hosting_device_id*)

Allocates a hosting port for a logical port.

We create a hosting port for the router port

create_hosting_device_resources(*context*, *complementary_id*, *tenant_id*, *mgmt_context*, *max_hosted*)

Create resources for a hosting device in a plugin specific way.

delete_hosting_device_resources (*context*, *tenant_id*, *mgmt_port*, ***kwargs*)
Deletes resources for a hosting device in a plugin specific way.

extend_hosting_port_info (*context*, *port_db*, *hosting_device*, *hosting_info*)
Extends hosting information for a logical port.

get_hosting_device_resources (*context*, *id*, *complementary_id*, *tenant_id*, *mgmt_nw_id*)
Returns information about all resources for a hosting device.

setup_logical_port_connectivity (*context*, *port_db*, *hosting_device_id*)
Establishes connectivity for a logical port.

This is done by hot plugging the interface(VIF) corresponding to the port from the VM.

teardown_logical_port_connectivity (*context*, *port_db*, *hosting_device_id*)
Removes connectivity for a logical port.

Unplugs the corresponding data interface from the VM.

Module contents

class networking_cisco.plugins.cisco.device_manager.plugging_drivers.**PluginSidePluggingDriv**
Bases: object

This class defines the API for plugging drivers.

These are used used by Cisco (routing service) plugin to perform various operations on the logical ports of logical (service) resources in a plugin compatible way.

allocate_hosting_port (*context*, *router_id*, *port_db*, *network_type*, *hosting_device_id*)
Allocates a hosting port for a logical port.

Schedules a logical port to a hosting port. Note that the hosting port may be the logical port itself.

Returns a dict *{'allocated_port_id': <id of allocated port>, 'allocated_vlan': <allocated VLAN or None>}* or None if allocation failed

Parameters

- **context** – Neutron api request context.
- **router_id** – id of Neutron router the logical port belongs to.
- **port_db** – Neutron logical router port.
- **network_type** – Type of network for logical router port
- **hosting_device_id** – id of hosting device

create_hosting_device_resources (*context*, *complementary_id*, *tenant_id*, *mgmt_context*, *max_hosted*)
Create resources for a hosting device in a plugin specific way.

Called when a hosting device is to be created so resources like networks and ports can be created for it in a plugin compatible way. This is primarily useful to service VMs.

returns: a dict {'mgmt_port': <mgmt port or None>, 'ports': <list of ports>, ... arbitrary driver items }

Parameters

- **context** – Neutron api request context.
- **complementary_id** – complementary id of hosting device

- **tenant_id** – id of tenant owning the hosting device resources
- **mgmt_context** – dict with members: mgmt_ip_address: ip address of hosting device's management port mgmt_nw_id: id of management network for hosting devices mgmt_sec_grp_id: id of security group for management network
- **max_hosted** – maximum number of logical resources.

delete_hosting_device_resources(*context*, *tenant_id*, *mgmt_port*, ***kwargs*)
Deletes resources for a hosting device in a plugin specific way.

Called when a hosting device has been deleted (or when its creation has failed) so resources like networks and ports can be deleted in a plugin compatible way. This it primarily useful to service VMs.

Parameters

- **context** – Neutron api request context.
- **tenant_id** – id of tenant owning the hosting device resources.
- **mgmt_port** – id of management port for the hosting device.
- **kwargs** – dictionary for any driver specific parameters.

extend_hosting_port_info(*context*, *port_db*, *hosting_device*, *hosting_info*)
Extends hosting information for a logical port.

Allows a driver to add driver specific information to the hosting information for a logical port.

Parameters

- **context** – Neutron api request context.
- **port_db** – Neutron port that hosting information concerns.
- **hosting_device** – Device that hosts the port.
- **hosting_info** – dict with hosting port information to be extended.

get_hosting_device_resources(*context*, *id*, *complementary_id*, *tenant_id*, *mgmt_nw_id*)
Returns information about all resources for a hosting device.

Called just before a hosting device is to be deleted so that information about the resources the hosting device uses can be collected.

returns: a dict {'mgmt_port': <mgmt port or None>, 'ports': <list of ports>, ... arbitrary driver items }

Parameters

- **context** – Neutron api request context.
- **id** – id of hosting device.
- **complementary_id** – complementary id of hosting device
- **tenant_id** – id of tenant owning the hosting device resources.
- **mgmt_nw_id** – id of management network for hosting devices.

setup_logical_port_connectivity(*context*, *port_db*, *hosting_device_id*)
Establishes connectivity for a logical port.

Performs the configuration tasks needed in the infrastructure to establish connectivity for a logical port.

Parameters

- **context** – Neutron api request context.
- **port_db** – Neutron port that has been created.
- **hosting_device_id** – ID of the device where this port is attached.

teardown_logical_port_connectivity(*context*, *port_db*, *hosting_device_id*)
Removes connectivity for a logical port.

Performs the configuration tasks needed in the infrastructure to disconnect a logical port.

Example: Remove a VLAN that is trunked to a service VM.

Parameters

- **context** – Neutron api request context.
- **port_db** – Neutron port about to be deleted.
- **hosting_device_id** – ID of the device where this port is attached.

networking_cisco.plugins.cisco.device_manager.rpc package

Submodules

networking_cisco.plugins.cisco.device_manager.rpc.devices_cfgagent_rpc_cb module

class networking_cisco.plugins.cisco.device_manager.rpc.devices_cfgagent_rpc_cb.**DeviceMgrC**
Bases: object

Cisco cfg agent rpc support in Device mgr service plugin.

get_hosting_devices_for_agent(*context*, *host*)
Fetches routers that a Cisco cfg agent is managing.

This function is supposed to be called when the agent has started, is ready to take on assignments and before any callbacks to fetch logical resources are issued.

Parameters

- **context** – contains user information
- **host** – originator of callback

Returns dict of hosting devices managed by the cfg agent

register_for_duty(*context*, *host*)
Report that Cisco cfg agent is ready for duty.

This function is supposed to be called when the agent has started, is ready to take on assignments and before any callbacks to fetch logical resources are issued.

Parameters

- **context** – contains user information
- **host** – originator of callback

Returns True if successfully registered, False if not successfully registered, None if no handler found. If unsuccessful the agent should retry registration a few seconds later

report_non_responding_hosting_devices(*context*, *host*, *hosting_device_ids*)
Report that a hosting device is determined to be dead.

Parameters

- **context** – contains user information
- **host** – originator of callback
- **hosting_device_ids** – list of non-responding hosting devices

target = <Target version=1.1>

update_hosting_device_status(*context*, *host*, *status_info*)
Report status changes for hosting devices.

Parameters

- **context** – contains user information
- **host** – originator of callback
- **status_info** – Dictionary with list of hosting device ids for each type of hosting device status to be updated i.e.:

```
{
    HD_ACTIVE: list_of_ids_of_active_hds,
    HD_NOT_RESPONDING: list_of_ids_of_not_responding_hds,
    HD_DEAD: list_of_ids_of_dead_hds,
    ...
}
```

networking_cisco.plugins.cisco.device_manager.rpc.devmgr_rpc_cfgagent_api module

class networking_cisco.plugins.cisco.device_manager.rpc.devmgr_rpc_cfgagent_api.**DeviceMgrC**

Bases: object

API for Device manager service plugin to notify Cisco cfg agent.

agent_updated(*context*, *admin_state_up*, *host*)
Updates cfg agent on <host> to enable or disable it.

get_hosting_device_configuration(*context*, *id*)
Fetch configuration of hosting device with id.

The configuration agent should respond with the running config of the hosting device.

hosting_devices_assigned_to_cfg_agent(*context*, *ids*, *host*)
Notify cfg agent to now handle some hosting devices.

This notification relieves the cfg agent in <host> of responsibility to monitor and configure hosting devices with id specified in <ids>.

hosting_devices_removed(*context*, *hosting_data*, *deconfigure*, *host*)
Notify cfg agent that some hosting devices have been removed.

This notification informs the cfg agent in <host> that the hosting devices in the <hosting_data> dictionary have been removed from the hosting device pool. The <hosting_data> dictionary also contains the ids of the affected logical resources for each hosting devices:

```
{'hd_id1': {'routers': [id1, id2, ...],
            'fw': [id1, ...],
            ...},
 'hd_id2': {'routers': [id3, id4, ...]},
```

```
            'fw': [id1, ...],
            ...},
 ...}
```

The <deconfigure> argument is True if any configurations for the logical resources should be removed from the hosting devices

hosting_devices_unassigned_from_cfg_agent (*context*, *ids*, *host*)
Notify cfg agent to no longer handle some hosting devices.

This notification relieves the cfg agent in <host> of responsibility to monitor and configure hosting devices with id specified in <ids>.

Module contents

networking_cisco.plugins.cisco.device_manager.scheduler package

Submodules

networking_cisco.plugins.cisco.device_manager.scheduler.hosting_device_cfg_agent_scheduler module

class networking_cisco.plugins.cisco.device_manager.scheduler.hosting_device_cfg_agent_sche
Bases: object

A scheduler for Cisco (hosting) device manager service plugin.

It schedules hosting devices to Cisco cfg agents. The scheduling is a simple random selection among qualified candidates.

auto_schedule_hosting_devices (*plugin*, *context*, *agent_host*)
Schedules unassociated hosting devices to Cisco cfg agent.

Schedules hosting devices to agent running on <agent_host>.

schedule_hosting_device (*plugin*, *context*, *hosting_device*)
Selects Cisco cfg agent that will configure <hosting_device>.

class networking_cisco.plugins.cisco.device_manager.scheduler.hosting_device_cfg_agent_sche
Bases: *networking_cisco.plugins.cisco.device_manager.scheduler.hosting_device_cfg_agent_scheduler.HostingDeviceCfgAgentScheduler*

This scheduler will assign a hosting device to an agent that has the fewest hosting devices associated to it

schedule_hosting_device (*plugin*, *context*, *hosting_device*)

Module contents

Submodules

networking_cisco.plugins.cisco.device_manager.config module

networking_cisco.plugins.cisco.device_manager.config.**get_specific_config** (*prefix*)
Retrieve config based on the format [<prefix>:<value>].

returns: a dict, {<UUID>: {<key1>:<value1>, <key2>:<value2>, ... }}

networking_cisco.plugins.cisco.device_manager.config.**obtain_hosting_device_credentials_fro**
Obtains credentials from config file and stores them in memory. To be called before hosting device templates defined in the config file are created.

networking_cisco.plugins.cisco.device_manager.config.**uuidify**(*val*)
Takes an integer and transforms it to a UUID format.

returns: UUID formatted version of input.

networking_cisco.plugins.cisco.device_manager.config.**verify_resource_dict**(*res_dict*, *is_create*, *attr_info*)
Verifies required attributes are in resource dictionary, res_dict.

Also checking that an attribute is only specified if it is allowed for the given operation (create/update).

Attribute with default values are considered to be optional.

This function contains code taken from function 'prepare_request_body' in attributes.py.

networking_cisco.plugins.cisco.device_manager.service_vm_lib module

class networking_cisco.plugins.cisco.device_manager.service_vm_lib.**ServiceVMManager**(*is_auth_v user=Nor passwd=l l3_admin auth_url= key- stone_ses*

Bases: object

delete_service_vm(*context*, *vm_id*)

delete_service_vm_fake(*context*, *vm_id*)

delete_service_vm_real(*context*, *vm_id*)

dispatch_service_vm(*context*, *instance_name*, *vm_image*, *vm_flavor*, *hosting_device_drv*, *credentials_info*, *connectivity_info*, *ports=None*)

dispatch_service_vm_fake(*context*, *instance_name*, *vm_image*, *vm_flavor*, *hosting_device_drv*, *credentials_info*, *connectivity_info*, *ports=None*)

dispatch_service_vm_real(*context*, *instance_name*, *vm_image*, *vm_flavor*, *hosting_device_drv*, *credentials_info*, *connectivity_info*, *ports=None*)

get_service_vm_status(*vm_id*)

interface_attach(*vm_id*, *port_id*)

interface_detach(*vm_id*, *port_id*)

nova_services_up()
Checks if required Nova services are up and running.

returns: True if all needed Nova services are up, False otherwise

vm_interface_list(*vm_id*)

Module contents

networking_cisco.plugins.cisco.extensions package

Submodules

networking_cisco.plugins.cisco.extensions.ciscocfgagentscheduler module

class networking_cisco.plugins.cisco.extensions.ciscocfgagentscheduler.**CfgAgentSchedulerPlu**
Bases: object

REST API to operate the cfg agent scheduler.

All of method must be in an admin context.

assign_hosting_device_to_cfg_agent(*context*, *id*, *hosting_device_id*)

list_cfg_agents_handling_hosting_device(*context*, *hosting_device_id*)

list_hosting_devices_handled_by_cfg_agent(*context*, *id*)

unassign_hosting_device_from_cfg_agent(*context*, *id*, *hosting_device_id*)

class networking_cisco.plugins.cisco.extensions.ciscocfgagentscheduler.**CfgAgentsHandlingHos**
Bases: neutron.wsgi.Controller

get_plugin()

index(*request*, ***kwargs*)

class networking_cisco.plugins.cisco.extensions.ciscocfgagentscheduler.**Ciscocfgagentschedul**
Bases: neutron_lib.api.extensions.ExtensionDescriptor

Extension class supporting configuration agent scheduler.

classmethod get_alias()

classmethod get_description()

get_extended_resources(*version*)

classmethod get_name()

classmethod get_namespace()

classmethod get_resources()
Returns Ext Resources.

classmethod get_updated()

exception networking_cisco.plugins.cisco.extensions.ciscocfgagentscheduler.**HostingDeviceAss**
Bases: neutron_lib.exceptions.Conflict

message = 'The hosting device %(hosting_device_id)s is already assigned to Cisco cfg a

exception networking_cisco.plugins.cisco.extensions.ciscocfgagentscheduler.**HostingDeviceNot**
Bases: neutron_lib.exceptions.NotFound

message = 'The hosting device %(hosting_device_id)s is currently not assigned to Cisco

class networking_cisco.plugins.cisco.extensions.ciscocfgagentscheduler.**HostingDeviceSchedul**
Bases: neutron.wsgi.Controller

create(*request*, *body*, ***kwargs*)

delete(*request*, ***kwargs*)

get_plugin()

index(*request*, ***kwargs*)

exception networking_cisco.plugins.cisco.extensions.ciscocfgagentscheduler.**HostingDeviceSch**
Bases: neutron_lib.exceptions.Conflict

message = 'Failed to assign hosting device %(hosting_device_id)s to Cisco cfg agent %(

exception networking_cisco.plugins.cisco.extensions.ciscocfgagentscheduler.**InvalidCfgAgent**
Bases: neutron_lib.exceptions.agent.AgentNotFound

message = 'Agent %(agent_id)s is not a Cisco cfg agent or has been disabled'

networking_cisco.plugins.cisco.extensions.ciscocfgagentscheduler.**notify**(*context*, *action*, *hosting_device_id*, *cfg_agent_id*)

networking_cisco.plugins.cisco.extensions.ciscohostingdevicemanager module

class networking_cisco.plugins.cisco.extensions.ciscohostingdevicemanager.**CiscoHostingDevic**
Bases: neutron_lib.services.base.ServicePluginBase

create_hosting_device(*context*, *hosting_device*)

create_hosting_device_template(*context*, *hosting_device_template*)

delete_hosting_device(*context*, *id*)

delete_hosting_device_template(*context*, *id*)

get_hosting_device(*context*, *id*, *fields=None*)

get_hosting_device_config(*context*, *id*)

get_hosting_device_template(*context*, *id*, *fields=None*)

get_hosting_device_templates(*context*, *filters=None*, *fields=None*, *sorts=None*, *limit=None*, *marker=None*, *page_reverse=False*)

get_hosting_devices(*context*, *filters=None*, *fields=None*, *sorts=None*, *limit=None*, *marker=None*, *page_reverse=False*)

get_plugin_description()

get_plugin_name()

get_plugin_type()

update_hosting_device(*context*, *id*, *hosting_device*)

update_hosting_device_template(*context*, *id*, *hosting_device_template*)

class networking_cisco.plugins.cisco.extensions.ciscohostingdevicemanager.**Ciscohostingdevic**
Bases: neutron_lib.api.extensions.ExtensionDescriptor

Hosting device and template extension.

classmethod get_alias()

classmethod get_description()

get_extended_resources(*version*)

classmethod get_name()

classmethod get_namespace()

classmethod get_resources()
Returns Ext Resources.

classmethod get_updated()

exception networking_cisco.plugins.cisco.extensions.ciscohostingdevicemanager.**HostingDevice**
Bases: neutron_lib.exceptions.InUse

message = 'Hosting device %(id)s in use.'

exception networking_cisco.plugins.cisco.extensions.ciscohostingdevicemanager.**HostingDevice**
Bases: neutron_lib.exceptions.InvalidInput

message = 'Invalid value for port %(port)s'

exception networking_cisco.plugins.cisco.extensions.ciscohostingdevicemanager.**HostingDevice**
Bases: neutron_lib.exceptions.InUse

message = 'Specified management port %(id)s does not exist.'

exception networking_cisco.plugins.cisco.extensions.ciscohostingdevicemanager.**HostingDevice**
Bases: neutron_lib.exceptions.NotFound

message = 'Hosting device %(id)s does not exist'

exception networking_cisco.plugins.cisco.extensions.ciscohostingdevicemanager.**HostingDevice**
Bases: neutron_lib.exceptions.InUse

message = 'Hosting device template %(id)s in use.'

exception networking_cisco.plugins.cisco.extensions.ciscohostingdevicemanager.**HostingDevice**
Bases: neutron_lib.exceptions.NotFound

message = 'Hosting device template %(id)s does not exist'

exception networking_cisco.plugins.cisco.extensions.ciscohostingdevicemanager.**TenantBoundNo**
Bases: neutron_lib.exceptions.NetworkNotFound

message = 'Attribute tenant_bound must be a list of tenant ids or None'

networking_cisco.plugins.cisco.extensions.ciscohostingdevicemanager.**convert_empty_string_to**

networking_cisco.plugins.cisco.extensions.ciscohostingdevicemanager.**convert_validate_port_v**

networking_cisco.plugins.cisco.extensions.ha module

exception networking_cisco.plugins.cisco.extensions.ha.**HADisabled**(***kwargs*)
Bases: neutron_lib.exceptions.Conflict

message = 'HA support is disabled'

exception networking_cisco.plugins.cisco.extensions.ha.**HADisabledHAType**(***kwargs*)
Bases: neutron_lib.exceptions.Conflict

message = 'HA type %(ha_type)s is administratively disabled'

exception networking_cisco.plugins.cisco.extensions.ha.**HARedundancyLevel**(***kwargs*)
Bases: neutron_lib.exceptions.BadRequest

message = 'Redundancy level for HA must be 1, 2, or 3'

exception networking_cisco.plugins.cisco.extensions.ha.**HATypeCannotBeChanged**(***kwargs*)
Bases: neutron_lib.exceptions.Conflict

message = 'HA type cannot be changed for a router with HA enabled'

exception networking_cisco.plugins.cisco.extensions.ha.**HATypeNotCompatibleWithFloatingIP**(**
Bases: neutron_lib.exceptions.BadRequest

message = 'HA type %(ha_type)s cannot be used with FloatingIP'

class networking_cisco.plugins.cisco.extensions.ha.**Ha**
Bases: neutron_lib.api.extensions.ExtensionDescriptor

Extension class to support HA by VRRP, HSRP and GLBP.

This class is used by Neutron's extension framework to support HA redundancy by VRRP, HSRP and GLBP for Neutron Routers.

Attribute 'ha_type' can be one of 'vrrp', 'hsrp' and 'glbp' Attribute 'redundancy_level' specifies the number of routers added for redundancy and can be 1, 2, or 3.

To create a router with HSRP-based HA with 2 extra routers for redundancy using the CLI with admin rights:

```
(shell) router-create <router_name> --ha:ha_type hsrp        --ha:redundancy_
↪level 2
```

classmethod get_alias()

classmethod get_description()

get_extended_resources(*version*)

classmethod get_name()

classmethod get_namespace()

classmethod get_updated()

networking_cisco.plugins.cisco.extensions.routerhostingdevice module

class networking_cisco.plugins.cisco.extensions.routerhostingdevice.**Routerhostingdevice**
Bases: neutron_lib.api.extensions.ExtensionDescriptor

Extension class to introduce hosting device information for routers.

This class is used by Neutron's extension framework to add hosting_device attribute to Neutron Routers implemented in virtual/physical appliances.

classmethod get_alias()

classmethod get_description()

get_extended_resources(*version*)

classmethod get_name()

classmethod get_namespace()

classmethod get_updated()

networking_cisco.plugins.cisco.extensions.routerrole module

class networking_cisco.plugins.cisco.extensions.routerrole.**Routerrole**
Bases: neutron_lib.api.extensions.ExtensionDescriptor

Extension class to introduce role information for routers.

This class is used by Neutron's extension framework to add role attribute to Neutron Routers implemented in virtual/physical appliances.

classmethod get_alias()

classmethod get_description()

get_extended_resources(*version*)

classmethod get_name()

classmethod get_namespace()

classmethod get_updated()

networking_cisco.plugins.cisco.extensions.routertype module

exception networking_cisco.plugins.cisco.extensions.routertype.**HostingDeviceTemplateUsedBy**
Bases: neutron_lib.exceptions.NeutronException

message = 'Router type %(type)s already defined for Hosting device template with id %(

exception networking_cisco.plugins.cisco.extensions.routertype.**MultipleRouterTypes**(***kwargs*)
Bases: neutron_lib.exceptions.NeutronException

message = 'Multiple router type with same name %(name)s exist. Id must be used to spec

exception networking_cisco.plugins.cisco.extensions.routertype.**NoSuchHostingDeviceTemplate**
Bases: neutron_lib.exceptions.NeutronException

message = 'No hosting device template with id %(type)s exists'

exception networking_cisco.plugins.cisco.extensions.routertype.**RouterTypeAlreadyDefined**(***k*
Bases: neutron_lib.exceptions.NeutronException

message = 'Router type %(type)s already exists'

exception networking_cisco.plugins.cisco.extensions.routertype.**RouterTypeHasRouters**(***kwargs*
Bases: neutron_lib.exceptions.NeutronException

message = 'Router type %(type)s cannot be deleted since routers of that type exists'

exception networking_cisco.plugins.cisco.extensions.routertype.**RouterTypeInUse**(***kwargs*)
Bases: neutron_lib.exceptions.InUse

message = 'Router type %(id)s in use.'

exception networking_cisco.plugins.cisco.extensions.routertype.**RouterTypeNotFound**(***kwargs*)
Bases: neutron_lib.exceptions.NotFound

message = 'Router type %(id)s does not exist'

class networking_cisco.plugins.cisco.extensions.routertype.**Routertype**
Bases: neutron_lib.api.extensions.ExtensionDescriptor

Extension class to define different types of Neutron routers.

This class is used by Neutron's extension framework to support definition of different types of Neutron Routers.

Attribute 'router_type:id' is the uuid or name of a certain router type. It can be set during creation of Neutron router. If a Neutron router is moved (by admin user) to a hosting device of a different hosting device type, the router type of the Neutron router will also change. Non-admin users can request that a Neutron router's type is changed.

To create a router of router type <name>:

(shell) router-create <router_name> –router_type:id <uuid_or_name>

classmethod get_alias()

classmethod get_description()

get_extended_resources(*version*)

classmethod get_name()

classmethod get_namespace()

classmethod get_resources()
Returns Ext Resources.

classmethod get_updated()

class networking_cisco.plugins.cisco.extensions.routertype.**RoutertypePluginBase**
Bases: object

REST API to manage router types.

All methods except listing require admin context.

create_routertype(*context*, *routertype*)
Creates a router type. Also binds it to the specified hosting device template.

delete_routertype(*context*, *id*)
Deletes a router type.

get_routertype(*context*, *id*, *fields=None*)
Lists defined router type.

get_routertypes(*context*, *filters=None*, *fields=None*, *sorts=None*, *limit=None*, *marker=None*, *page_reverse=False*)
Lists defined router types.

update_routertype(*context*, *id*, *routertype*)
Updates a router type.

exception networking_cisco.plugins.cisco.extensions.routertype.**SchedulerNotFound**(***kwargs*)
Bases: neutron_lib.exceptions.NetworkNotFound

message = 'Scheduler %(scheduler)s does not exist'

networking_cisco.plugins.cisco.extensions.routertypeawarescheduler module

class networking_cisco.plugins.cisco.extensions.routertypeawarescheduler.**HostingDevicesHost**
Bases: neutron.wsgi.Controller

get_plugin()

index(*request*, ***kwargs*)

exception networking_cisco.plugins.cisco.extensions.routertypeawarescheduler.**InvalidHosting**
Bases: neutron_lib.exceptions.NotFound

message = 'Hosting device %(hosting_device_id)s does not exist or has been disabled.'

exception networking_cisco.plugins.cisco.extensions.routertypeawarescheduler.**RouterHostedBy**
Bases: neutron_lib.exceptions.Conflict

message = 'The router %(router_id)s is already hosted by the hosting device %(hosting_

exception networking_cisco.plugins.cisco.extensions.routertypeawarescheduler.**RouterHostingI**
Bases: neutron_lib.exceptions.Conflict

message = 'Cannot host %(router_type)s router %(router_id)s on hosting device %(hostin

class networking_cisco.plugins.cisco.extensions.routertypeawarescheduler.**RouterHostingDevic**
Bases: neutron.wsgi.Controller

create(*request*, *body*, ***kwargs*)

delete(*request*, ***kwargs*)

get_plugin()

index(*request*, ***kwargs*)

exception networking_cisco.plugins.cisco.extensions.routertypeawarescheduler.**RouterNotHoste**
Bases: neutron_lib.exceptions.Conflict

message = 'The router %(router_id)s is not hosted by hosting device %(hosting_device_i

exception networking_cisco.plugins.cisco.extensions.routertypeawarescheduler.**RouterReschedu**
Bases: neutron_lib.exceptions.Conflict

message = 'Failed rescheduling router %(router_id)s: no eligible hosting device found

exception networking_cisco.plugins.cisco.extensions.routertypeawarescheduler.**RouterSchedul**
Bases: neutron_lib.exceptions.Conflict

message = 'Failed scheduling router %(router_id)s to hosting device %(hosting_device_i

class networking_cisco.plugins.cisco.extensions.routertypeawarescheduler.**RouterTypeAwareSc**
Bases: object

REST API to operate the routertype-aware scheduler.

All of method must be in an admin context.

add_router_to_hosting_device(*context*, *hosting_device_id*, *router_id*)

list_hosting_devices_hosting_router(*context*, *router_id*)

list_routers_on_hosting_device(*context*, *hosting_device_id*)

remove_router_from_hosting_device(*context*, *hosting_device_id*, *router_id*)

class networking_cisco.plugins.cisco.extensions.routertypeawarescheduler.**Routertypeawaresch**
Bases: neutron_lib.api.extensions.ExtensionDescriptor

Extension class supporting l3 agent scheduler.

classmethod get_alias()

classmethod get_description()

get_extended_resources(*version*)

classmethod get_name()

classmethod get_namespace()

classmethod get_resources()
Returns Ext Resources.

classmethod get_updated()

networking_cisco.plugins.cisco.extensions.routertypeawarescheduler.**notify**(*context*, *action*, *router_id*, *hosting_device_id*)

Module contents

networking_cisco.plugins.cisco.l3 package

Subpackages

networking_cisco.plugins.cisco.l3.drivers package

Subpackages

networking_cisco.plugins.cisco.l3.drivers.asr1k package

Submodules

networking_cisco.plugins.cisco.l3.drivers.asr1k.asr1k_routertype_driver module

class networking_cisco.plugins.cisco.l3.drivers.asr1k.asr1k_routertype_driver.**ASR1kL3Router**

Bases: *networking_cisco.plugins.cisco.l3.drivers.L3RouterBaseDriver*

add_router_interface_postcommit(*context*, *r_port_context*)

add_router_interface_precommit(*context*, *r_port_context*)

create_floatingip_postcommit(*context*, *fip_context*)

create_floatingip_precommit(*context*, *fip_context*)

create_router_postcommit(*context*, *router_context*)

create_router_precommit(*context*, *router_context*)

delete_floatingip_postcommit(*context*, *fip_context*)

delete_floatingip_precommit(*context*, *fip_context*)

delete_router_postcommit(*context*, *router_context*)

delete_router_precommit(*context*, *router_context*)

generate_ha_group_id(*context*, *router*, *port*, *ha_settings_db*, *ha_group_uuid*)

ha_interface_ip_address_needed(*context*, *router*, *port*, *ha_settings_db*, *ha_group_uuid*)

post_backlog_processing(*context*)

pre_backlog_processing(*context*)

remove_router_interface_postcommit(*context*, *r_port_context*)

remove_router_interface_precommit(*context*, *r_port_context*)

schedule_router_postcommit(*context*, *router_context*)

schedule_router_precommit(*context*, *router_context*)

unschedule_router_postcommit(*context*, *router_context*)

unschedule_router_precommit(*context*, *router_context*)

update_floatingip_postcommit(*context*, *fip_context*)

update_floatingip_precommit(*context*, *fip_context*)

update_router_postcommit(*context*, *router_context*)

update_router_precommit(*context*, *router_context*)

exception networking_cisco.plugins.cisco.l3.drivers.asr1k.asr1k_routertype_driver.**TopologyN**
Bases: neutron_lib.exceptions.Conflict

message = 'Requested topology cannot be supported by router.'

Module contents

Submodules

networking_cisco.plugins.cisco.l3.drivers.driver_context module

class networking_cisco.plugins.cisco.l3.drivers.driver_context.**FloatingipContext**(*fip*, *old_fip=None*)
Bases: *networking_cisco.plugins.cisco.l3.drivers.driver_context.L3ContextBase*

current

current_router

original

original_router

class networking_cisco.plugins.cisco.l3.drivers.driver_context.**L3ContextBase**
Bases: object

params

class networking_cisco.plugins.cisco.l3.drivers.driver_context.**RouterContext**(*router*, *old_router=None*)
Bases: *networking_cisco.plugins.cisco.l3.drivers.driver_context.L3ContextBase*

current

original

class networking_cisco.plugins.cisco.l3.drivers.driver_context.**RouterPortContext**(*port*, *router*, *old_port=Non*, *sub-net_id=None*)
Bases: *networking_cisco.plugins.cisco.l3.drivers.driver_context.L3ContextBase*

current

current_router

current_subnet_id

original

original_router

router_context

networking_cisco.plugins.cisco.l3.drivers.noop_routertype_driver module

class networking_cisco.plugins.cisco.l3.drivers.noop_routertype_driver.**NoopL3RouterDriver**
Bases: *networking_cisco.plugins.cisco.l3.drivers.L3RouterBaseDriver*

add_router_interface_postcommit(*context*, *r_port_context*)

add_router_interface_precommit(*context*, *r_port_context*)

create_floatingip_postcommit(*context*, *fip_context*)

create_floatingip_precommit(*context*, *fip_context*)

create_router_postcommit(*context*, *router_context*)

create_router_precommit(*context*, *router_context*)

delete_floatingip_postcommit(*context*, *fip_context*)

delete_floatingip_precommit(*context*, *fip_context*)

delete_router_postcommit(*context*, *router_context*)

delete_router_precommit(*context*, *router_context*)

post_backlog_processing(*context*)

pre_backlog_processing(*context*)

remove_router_interface_postcommit(*context*, *r_port_context*)

remove_router_interface_precommit(*context*, *r_port_context*)

schedule_router_postcommit(*context*, *router_context*)

schedule_router_precommit(*context*, *router_context*)

unschedule_router_postcommit(*context*, *router_context*)

unschedule_router_precommit(*context*, *router_context*)

update_floatingip_postcommit(*context*, *fip_context*)

update_floatingip_precommit(*context*, *fip_context*)

update_router_postcommit(*context*, *router_context*)

update_router_precommit(*context*, *router_context*)

Module contents

class networking_cisco.plugins.cisco.l3.drivers.**L3RouterBaseDriver**
Bases: object

add_router_interface_postcommit(*context*, *r_port_context*)

add_router_interface_precommit(*context*, *r_port_context*)

create_floatingip_postcommit(*context*, *fip_context*)
Create a floatingip.

Parameters

- **context** – the neutron context of the request
- **fip_context** – FloatingipContext instance describing the new state of the floatingip

Called after the transaction commits. Call can block, though will block the entire process so care should be taken to not drastically affect performance. Raising an exception will cause the deletion of the resource.

create_flotingip_postcommit is called for all changes to the router state. It is up to the routertype driver to ignore state or state changes that it does not know or care about.

create_floatingip_precommit(*context*, *fip_context*)
Create a floatingip.

Parameters

- **context** – the neutron context of the request
- **fip_context** – FloatingipContext instance describing the new state of the floatingip

Called before the transaction commits. Call can block, though will block the entire process so care should be taken to not drastically affect performance. Raising an exception will cause the deletion of the resource.

create_flotingip_precommit will not be used by most drivers. The only way a routertype driver can be known is to assume the default router type. This API was introduced to support allocation of floating IPs from NAT pools for Group Based Policy (GBP) workflow.

create_router_postcommit(*context*, *router_context*)
Create a router.

Parameters

- **context** – the neutron context of the request
- **router_context** – RouterContext instance describing the new router.

Called after the transaction commits. Call can block, though will block the entire process so care should be taken to not drastically affect performance. Raising an exception will cause the deletion of the resource.

create_router_precommit(*context*, *router_context*)
Perform operations specific to the router type in preparation for the creation of a new router.

Parameters

- **context** – the neutron context of the request
- **router_context** – RouterContext instance describing the new router.

Create a new router, allocating resources as necessary in the database. Called inside transaction context on session. Call cannot block. Raising an exception will result in a rollback of the current transaction.

delete_floatingip_postcommit(*context*, *fip_context*)
Delete a floatingip.

Parameters

- **context** – the neutron context of the request
- **fip_context** – FloatingipContext instance describing the current state of the floatingip, prior to the call to delete it.

Called after the transaction commits. Call can block, though will block the entire process so care should be taken to not drastically affect performance. Runtime errors are not expected, and will not prevent the resource from being deleted.

delete_floatingip_precommit(*context*, *fip_context*)
Perform operations specific to the routertype in preparation for the deletion of a floatingip.

Parameters

- **context** – the neutron context of the request
- **fip_context** – FloatingipContext instance describing the current state of the floatingip, prior to the call to delete it.

Delete floatingip resources previously allocated by this routertype driver for a floatingip. Called inside transaction context on session. Runtime errors are not expected, but raising an exception will result in rollback of the transaction.

delete_router_postcommit(*context*, *router_context*)
Delete a router.

Parameters

- **context** – the neutron context of the request
- **router_context** – RouterContext instance describing the current state of the router, prior to the call to delete it.

Called after the transaction commits. Call can block, though will block the entire process so care should be taken to not drastically affect performance. Runtime errors are not expected, and will not prevent the resource from being deleted.

delete_router_precommit(*context*, *router_context*)
Perform operations specific to the routertype in preparation for the deletion of a router.

Parameters

- **context** – the neutron context of the request
- **router_context** – RouterContext instance describing the current state of the router, prior to the call to delete it.

Delete router resources previously allocated by this routertype driver for a router. Called inside transaction context on session. Runtime errors are not expected, but raising an exception will result in rollback of the transaction.

generate_ha_group_id(*context*, *router*, *port*, *ha_settings_db*, *ha_group_uuid*)
Returns None or a unique integer value for use as identifier of an HSRP, VRRP or GLBP group configuration.

Parameters

- **context** – the neutron context of the request
- **router** – dictionary of HA router
- **port** – dictionary of port associated with the HA group
- **ha_settings_db** – db object with ha settings
- **ha_group_uuid** – uuid of HA group's DB entry

:returns - an integer value group identifier or None

If None is returned, an identifier will be generated in a non-driver specific manner.

get_ha_group_timers_parameters(*context*, *router*, *port*, *ha_settings_db*, *ha_group_uuid*)
Returns timers specific parameters for an HA group.

Parameters

- **context** – the neutron context of the request
- **router** – dictionary of HA router
- **port** – dictionary of port associated with the HA group
- **ha_settings_db** – db object with ha settings
- **ha_group_uuid** – uuid of HA group's DB entry

:returns - a text string with the parameters

The text string is supposed to be understandable by the relevant config agent drivers corresponding to this router type driver.

get_ha_group_tracking_parameters(*context*, *router*, *port*, *ha_settings_db*, *ha_group_uuid*)
Returns object tracking parameters for an HA group.

Parameters

- **context** – the neutron context of the request
- **router** – dictionary describing router
- **port** – dictionary of port associated with the HA group
- **ha_settings_db** – db object with ha settings
- **ha_group_uuid** – uuid of HA group's DB entry

:returns - a text string with the parameters

The text string is supposed to be understandable by the relevant config agent drivers corresponding to this router type driver.

get_other_ha_group_parameters(*context*, *router*, *port*, *ha_settings_db*, *ha_group_uuid*)
Returns arbitrary HA parameters for an HA group.

Parameters

- **context** – the neutron context of the request
- **router** – dictionary describing router
- **port** – dictionary of port associated with the HA group
- **ha_settings_db** – db object with ha settings
- **ha_group_uuid** – uuid of HA group's DB entry

Returns a text string with the parameters

The text string is supposed to be understandable by the relevant config agent drivers corresponding to this router type driver.

ha_interface_ip_address_needed(*context*, *router*, *port*, *ha_settings_db*, *ha_group_uuid*)
Determines if a router interface for an HA enabled router needs an extra IP address (in addition to the VIP address)

Parameters

- **context** – the neutron context of the request
- **router** – dictionary of HA router

- **port** – dictionary of port associated with the HA group
- **ha_settings_db** – db object with ha settings
- **ha_group_uuid** – uuid of HA group's DB entry

:returns - True if an extra IP address is needed, False otherwise.

post_backlog_processing(*context*)
Perform driver specific processing after backlog is processed.

Parameters **context** – admin neutron context

This function is invoked after the router plugin has processed the backlog of unscheduled routers. It allows the router type driver to perform any tasks that should be performed after the backlog has been processed.

pre_backlog_processing(*context*)
Perform driver specific processing before backlog is processed.

Parameters **context** – admin neutron context

This function is invoked before the router plugin is processing the backlog of unscheduled routers. It allows the router type driver to perform any tasks that should be performed before the backlog is processed.

remove_router_interface_postcommit(*context*, *r_port_context*)

remove_router_interface_precommit(*context*, *r_port_context*)

schedule_router_postcommit(*context*, *router_context*)
Schedule the router.

Parameters

- **context** – the neutron context of the request
- **router_context** – RouterContext instance describing the current state of the router, prior to the call to schedule it.

Called after the transaction commits. Call can block, though will block the entire process so care should be taken to not drastically affect performance. Raising an exception will cause un-scheduling of the router.

schedule_router_precommit(*context*, *router_context*)
Perform operations specific to the routertype in preparation for scheduling of a router.

Parameters

- **context** – the neutron context of the request
- **router_context** – RouterContext instance describing the current state of the router, prior to the call to schedule it.

Perform operations that need to happen before scheduling of routers of this routertype. Called inside transaction context on session. Raising an exception will result in rollback of the transaction.

unschedule_router_postcommit(*context*, *router_context*)
Un-schedule the router.

Parameters

- **context** – the neutron context of the request
- **router_context** – RouterContext instance describing the current state of the router, prior to the call to schedule it.

Called after the transaction commits. Call can block, though will block the entire process so care should be taken to not drastically affect performance. Raising an exception will cause re-scheduling of the router.

unschedule_router_precommit(*context*, *router_context*)
Perform operations specific to the routertype in preparation for un-scheduling of a router.

Parameters

- **context** – the neutron context of the request
- **router_context** – RouterContext instance describing the current state of the router, prior to the call to schedule it.

Perform operations that need to happen before un-scheduling of routers of this routertype. Called inside transaction context on session. Raising an exception will result in rollback of the transaction.

update_floatingip_postcommit(*context*, *fip_context*)
Update a floatingip.

Parameters

- **context** – the neutron context of the request
- **fip_context** – FloatingipContext instance describing the new state of the floatingip, as well as the original state prior to the update_floatingip call.

Called after the transaction commits. Call can block, though will block the entire process so care should be taken to not drastically affect performance. Raising an exception will cause the deletion of the resource.

update_flotingip_postcommit is called for all changes to the router state. It is up to the routertype driver to ignore state or state changes that it does not know or care about.

update_floatingip_precommit(*context*, *fip_context*)
Perform operations specific to the routertype in preparation for the update of a floatingip.

Parameters

- **context** – the neutron context of the request
- **fip_context** – FloatingipContext instance describing the new state of the floatingip, as well as the original state prior to the update_floatingip call.

Update values of a floatingip, updating the associated resources in the database. Called inside transaction context on session. Raising an exception will result in rollback of the transaction.

update_floatingip_precommit is called for all changes to the floatingip state. It is up to the routertype driver to ignore state or state changes that it does not know or care about.

update_router_postcommit(*context*, *router_context*)
Update a router.

Parameters

- **context** – the neutron context of the request
- **router_context** – RouterContext instance describing the new state of the router, as well as the original state prior to the update_router call.

Called after the transaction commits. Call can block, though will block the entire process so care should be taken to not drastically affect performance. Raising an exception will cause the deletion of the resource.

update_router_postcommit is called for all changes to the router state. It is up to the routertype driver to ignore state or state changes that it does not know or care about.

update_router_precommit(*context*, *router_context*)
Perform operations specific to the router type in preparation for the update of a router.

Parameters

- **context** – the neutron context of the request
- **router_context** – RouterContext instance describing the new state of the router, as well as the original state prior to the update_router call.

Update values of a router, updating the associated resources in the database. Called inside transaction context on session. Raising an exception will result in rollback of the transaction.

update_router_precommit is called for all changes to the router state. It is up to the router type driver to ignore state or state changes that it does not know or care about.

networking_cisco.plugins.cisco.l3.rpc package

Submodules

networking_cisco.plugins.cisco.l3.rpc.l3_router_cfg_agent_rpc_cb module

class networking_cisco.plugins.cisco.l3.rpc.l3_router_cfg_agent_rpc_cb.**L3RouterCfgRpcCallba**

Bases: object

Cisco cfg agent rpc support in L3 routing service plugin.

cfg_sync_all_hosted_routers(*context*, *host*)

cfg_sync_routers(*context*, *host*, *router_ids=None*, *hosting_device_ids=None*)

Sync routers according to filters to a specific Cisco cfg agent.

Parameters

- **context** – contains user information
- **host** – originator of callback
- **router_ids** – list of router ids to return information about
- **hosting_device_ids** – list of hosting device ids to get routers for.

Returns a list of routers with their hosting devices, interfaces and floating_ips

get_cfg_router_ids(*context*, *host*, *router_ids=None*, *hosting_device_ids=None*)

Returns IDs of routers scheduled to l3 agent on <host>

report_status(*context*, *host*, *status_list*)

Report status of a particular Neutron router by Cisco cfg agent.

This is called by Cisco cfg agent when it has performed an operation on a Neutron router. Note that the agent may include status updates for multiple routers in one message.

Parameters

- **context** – contains user information
- **host** – originator of callback
- **status_list** – list of status dicts for routers. Each list item is:

```
{'router_id': <router_id>,
 'operation': <attempted operation>
 'status': <'SUCCESS'|'FAILURE'>,
 'details': <optional explaining details>}
```

target = <Target version=1.3>

update_floatingip_statuses_cfg(*context*, *router_id*, *fip_statuses*)
Update operational status for one or several floating IPs.

This is called by Cisco cfg agent to update the status of one or several floatingips.

Parameters

- **context** – contains user information
- **router_id** – id of router associated with the floatingips
- **router_id** – dict with floatingip_id as key and status as value

update_port_statuses_cfg(*context*, *port_ids*, *status*)
Update the operational statuses of a list of router ports.

This is called by the Cisco cfg agent to update the status of a list of ports.

Parameters

- **context** – contains user information
- **port_ids** – list of ids of all the ports for the given status
- **status** – PORT_STATUS_ACTIVE/PORT_STATUS_DOWN.

networking_cisco.plugins.cisco.l3.rpc.l3_router_rpc_cfg_agent_api module

class networking_cisco.plugins.cisco.l3.rpc.l3_router_rpc_cfg_agent_api.**L3RouterCfgAgentNot**

Bases: object

API for plugin to notify Cisco cfg agent.

router_added_to_hosting_device(*context*, *router*)
Notify cfg agent about router added to hosting device.

router_deleted(*context*, *router*)
Notifies cfg agents about a deleted router.

router_removed_from_hosting_device(*context*, *router*)
Notify cfg agent about router removed from hosting device.

routers_removed_from_hosting_device(*context*, *router_ids*, *hosting_device*)
Notify cfg agent that routers have been removed from hosting device. @param: context - information about tenant, user etc @param: router-ids - list of ids @param: hosting_device - device hosting the routers

routers_updated(*context*, *routers*, *operation=None*, *data=None*, *shuffle_agents=False*)
Notify cfg agents about configuration changes to routers.

This includes operations performed on the router like when a router interface is added or removed.

networking_cisco.plugins.cisco.l3.rpc.l3_rpc_agent_api_noop module

class networking_cisco.plugins.cisco.l3.rpc.l3_rpc_agent_api_noop.**L3AgentNotifyAPINoOp**(*topic*
Bases: object

API for plugin to notify L3 agent but without actions.

BASE_RPC_API_VERSION = '1.0'

agent_updated(*context*, *admin_state_up*, *host*)

router_added_to_agent(*context*, *routers*, *host*)

router_deleted(*context*, *router_id*)

router_removed_from_agent(*context*, *router_id*, *host*)

routers_updated(*context*, *routers*, *operation=None*, *data=None*)

Module contents

networking_cisco.plugins.cisco.l3.schedulers package

Submodules

networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler module

class networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.**Cand**
Bases: object

class networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.**L3Rou**
Bases: object

Slot-aware base scheduler of Neutron routers to hosting devices.

get_candidates(*plugin*, *context*, *r_hd_binding_db*)

Selection criteria: Hosting devices that... ... are based on the template required by router's type AND ... are administratively up AND ... are active (i.e., has status HD_ACTIVE) AND ... are bound to tenant owning router OR is unbound AND ... are enough slots available to host the router

Among hosting devices meeting these criteria the device with less allocated slots is preferred.

schedule_router(*plugin*, *context*, *r_hd_binding_db*)

unschedule_router(*plugin*, *context*, *r_hd_binding_db*)

class networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.**L3Rou**
Bases: *networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.CandidatesHAFilterMixin*, *networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.L3RouterHostingDeviceLongestRunningScheduler*

Schedules a Neutron router on a hosting device.

The scheduler is HA aware and will ignore hosting device candidates that are used by other Neutron routers in the same HA group.

Selection criteria: The longest running hosting device that is not already hosting a router in the HA group and which has enough slots available to host the router.

Hosting devices with creation date/time less than EQUIVALENCE_TIME_DIFF are considered equally old.

Among hosting devices meeting these criteria and that are of same age the device with less allocated slots is preferred.

class networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.**L3Rou**
Bases: *networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.CandidatesHAFilterMixin*, *networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.L3RouterHostingDeviceRandomScheduler*

Schedules a Neutron router on a hosting device.

The scheduler is HA aware and will ignore hosting device candidates that are used by other Neutron routers in the same HA group.

Selection criteria: A randomly selected hosting device that is not already hosting a router in the HA group and which has enough slots available to host the router.

class networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.**L3Rou**
Bases: *networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.L3RouterHostingDeviceBaseScheduler*

Schedules a Neutron router on a hosting device.

Selection criteria: The longest running hosting device that has enough slots available to host the router

Hosting devices with creation date/time less than EQUIVALENCE_TIME_DIFF are considered equally old.

Among hosting devices meeting these criteria and that are of same age the device with less allocated slots is preferred.

schedule_router(*plugin*, *context*, *r_hd_binding_db*)

unschedule_router(*plugin*, *context*, *r_hd_binding_db*)

class networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.**L3Rou**
Bases: *networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.L3RouterHostingDeviceBaseScheduler*

Schedules a Neutron router on a hosting device.

Selection criteria: A randomly selected hosting device that has enough slots available to host the router.

schedule_router(*plugin*, *context*, *r_hd_binding_db*)

unschedule_router(*plugin*, *context*, *r_hd_binding_db*)

networking_cisco.plugins.cisco.l3.schedulers.l3_routertype_aware_agent_scheduler module

class networking_cisco.plugins.cisco.l3.schedulers.l3_routertype_aware_agent_scheduler.**L3Ro**
Bases: neutron.scheduler.l3_agent_scheduler.L3Scheduler

A router type aware l3 agent scheduler for Cisco router service plugin.

It schedules Neutron routers with router type representing network namespace based routers to l3 agents.

schedule(*plugin*, *context*, *router*, *candidates=None*, *hints=None*)

networking_cisco.plugins.cisco.l3.schedulers.noop_l3_router_hosting_device_scheduler module

class networking_cisco.plugins.cisco.l3.schedulers.noop_l3_router_hosting_device_scheduler
Bases: object

No-op scheduler of Neutron routers on hosting devices.

schedule_router(*plugin*, *context*, *r_hd_binding*)

unschedule_router(*plugin*, *context*, *r_hd_binding*)

Module contents

Module contents

networking_cisco.plugins.cisco.service_plugins package

Submodules

networking_cisco.plugins.cisco.service_plugins.cisco_device_manager_plugin module

class networking_cisco.plugins.cisco.service_plugins.cisco_device_manager_plugin.**CiscoDevi**
Bases: *networking_cisco.plugins.cisco.db.device_manager.hosting_device_manager_db.HostingDeviceManagerMixin*, *networking_cisco.plugins.cisco.db.scheduler.cfg_agentschedulers_db.CfgAgentSchedulerDbMixin*

Implementation of Cisco Device Manager Service Plugin for Neutron.

This class implements a (hosting) device manager service plugin that provides hosting device template and hosting device resources. As such it manages associated REST API processing. All DB functionality is implemented in class hosting_device_manager_db.HostingDeviceManagerMixin.

path_prefix = '/dev_mgr'

setup_rpc()

supported_extension_aliases = ['dev_mgr', 'cisco-cfg-agent-scheduler']

networking_cisco.plugins.cisco.service_plugins.cisco_router_plugin module

class networking_cisco.plugins.cisco.service_plugins.cisco_router_plugin.**CiscoRouterPlugin**
Bases: neutron.db.common_db_mixin.CommonDbMixin, *networking_cisco.plugins.cisco.db.l3.routertype_db.RoutertypeDbMixin*, *networking_cisco.plugins.cisco.db.l3.ha_db.HA_db_mixin*, networking_cisco.plugins.cisco.db.l3.l3_router_appliance_db.L3RouterApplianceDBMixin, *networking_cisco.plugins.cisco.db.scheduler.l3_routertype_aware_schedulers_db.L3RouterTypeAwareSchedulerDbMixin*, neutron.db.dns_db.DNSDbMixin

Implementation of Cisco L3 Router Service Plugin for Neutron.

This class implements a L3 service plugin that provides router and floatingip resources and manages associated request/response. All DB functionality is implemented in class l3_router_appliance_db.L3RouterApplianceDBMixin.

create_floatingip(*context*, *floatingip*)
Create floating IP.

Parameters

- **context** – Neutron request context
- **floatingip** – data for the floating IP being created

Returns A floating IP object on success

As the l3 router plugin asynchronously creates floating IPs leveraging the l3 agent and l3 cfg agent, the initial status for the floating IP object will be DOWN.

get_plugin_description()

get_plugin_type()

setup_rpc()

supported_extension_aliases = ['router', 'standard-attr-description', 'extraroute', 'l

Module contents

Module contents

Module contents

networking_cisco.services package

Subpackages

networking_cisco.services.trunk package

Submodules

networking_cisco.services.trunk.nexus_trunk module

class networking_cisco.services.trunk.nexus_trunk.**NexusTrunkDriver**(*name*, *interfaces*, *segmentation_types*, *agent_type=None*, *can_trunk_bound_port=False*)

Bases: neutron.services.trunk.drivers.base.DriverBase

Cisco Nexus Trunk Driver.

This class contains methods required to work with the trunk infrastruture.

classmethod create()

is_loaded

register(*resource*, *event*, *trigger*, ***kwargs*)

class networking_cisco.services.trunk.nexus_trunk.**NexusTrunkHandler**

Bases: object

Cisco Nexus Trunk Handler.

This class contains methods called by the trunk infrastruture to be processed by the cisco_nexus MD.

subport_postcommit(*resource*, *event*, *trunk_plugin*, *payload*)

trunk_update_postcommit(*resource*, *event*, *trunk_plugin*, *payload*)

networking_cisco.services.trunk.trunkstubs module

class networking_cisco.services.trunk.trunkstubs.**DriverBase**(*name*, *interfaces*, *segmentation_types*, *agent_type=None*, *can_trunk_bound_port=False*)

Bases: object

class networking_cisco.services.trunk.trunkstubs.**SubPort**

Bases: object

classmethod get_object(*context*, **args*, ***kwargs*)

class networking_cisco.services.trunk.trunkstubs.**Trunk**

Bases: object

classmethod get_object(*context*, ***kargs*)

class networking_cisco.services.trunk.trunkstubs.**TrunkObject**

Bases: object

classmethod update(***kargs*)

Module contents

Module contents

networking_cisco.tests package

Subpackages

networking_cisco.tests.unit package

Subpackages

networking_cisco.tests.unit.cisco package

Subpackages

networking_cisco.tests.unit.cisco.cfg_agent package

Submodules

networking_cisco.tests.unit.cisco.cfg_agent.cfg_agent_test_support module

class networking_cisco.tests.unit.cisco.cfg_agent.cfg_agent_test_support.**CfgAgentTestSuppor**

Bases: object

create_router_port(*network_uuid*, *vlan_tag*, *k*, *num_subnets*, *router_id*, *admin_state_up=True*, *mac_address='ca:fe:de:ad:be:ef'*, *device_owner='network:router_interface'*, *ha_enabled=True*, *ha_group=1060*, *is_user_visible=True*)

prepare_hosting_device_params()

prepare_router_data(*set_gateway=True*, *enable_snat=None*, *num_ext_subnets=1*, *num_internal_ports=1*, *same_internal_nw=False*, *is_global=False*, *ha_enabled=True*, *is_user_visible=True*)

networking_cisco.tests.unit.cisco.cfg_agent.test_asr1k_cfg_syncer module

class networking_cisco.tests.unit.cisco.cfg_agent.test_asr1k_cfg_syncer.**ASR1kCfgSyncer**(**args*, ***kw*)

Bases: neutron.tests.base.BaseTestCase

setUp()

test_clean_acls_basic_running_cfg()
region 1 acls should be ignored

test_clean_ha_backup_router_with_two_subnet_gw()

test_clean_ha_backup_routers_with_two_subnet_gw_and_single_subnet_gw()

test_clean_interfaces_R2_run_cfg_present_multi_region_enabled()
In this test, we are simulating a cfg-sync, clean_interfaces for region 0000002 cfg-agent. Existing running-cfg exists for region 0000001 and 0000002.

At the end of test, we should expect zero entries in invalid_cfg.

test_clean_interfaces_R2_with_invalid_intfs()
In this test, we are simulating a cfg-sync, clean_interfaces for region 0000002 cfg-agent. Existing running-cfg exists for region 0000001 and 0000002.

At the end of test, we should expect two invalid intfs detected.

invalid tenant router, int Po10.2536 (invalid segment-id) invalid ext-gw-port, int Po10.3000 (invalid HSRP VIP)

test_clean_interfaces_basic_multi_region_enabled()
In this test, we are simulating a cfg-sync, clean_interfaces for region 0000002 cfg-agent. Running-cfg only exists for region 0000001.

At the end of test, we should expect zero entries in invalid_cfg.

test_clean_interfaces_multi_region_disabled()
In this test, we are simulating a cfg-sync, clean_interfaces for region 0000002 cfg-agent. Running-cfg only exists for region 0000001, but multi_region is disabled.

At the end of test, we should expect zero entries in invalid_cfg.

test_clean_nat_pool_overload_basic_running_cfg()
region 1 acls should be ignored

test_clean_router_with_two_subnet_gw()

test_clean_routers_with_two_subnet_gw_and_single_subnet_gw()

test_delete_invalid_cfg_empty_routers_list()
expected invalid_cfg:

```
[u'ip nat inside source static 10.2.0.5 172.16.0.126 vrf'
  ' nrouter-3ea5f9 redundancy neutron-hsrp-1064-3000',
 u'ip nat inside source list neutron_acl_2564 pool'
  ' nrouter-3ea5f9_nat_pool vrf nrouter-3ea5f9 overload',
 u'ip nat pool nrouter-3ea5f9_nat_pool 172.16.0.124'
  ' 172.16.0.124 netmask 255.255.0.0',
```

```
u'ip route vrf nrouter-3ea5f9 0.0.0.0 0.0.0.0'
 ' Port-channel10.3000 172.16.0.1',
u'ip access-list standard neutron_acl_2564',
<IOSCfgLine # 83 'interface Port-channel10.2564'>,
<IOSCfgLine # 96 'interface Port-channel10.3000'>,
u'nrouter-3ea5f9']
```

test_delete_invalid_cfg_with_multi_region_and_empty_routers_list()

This test verifies that the cfg-syncer will delete invalid cfg if the neutron-db (routers dictionary list) happens to be empty.

Since the neutron-db router_db_info is empty, all region 0000002 running-config should be deleted.

Expect 8 invalid configs found:

```
['ip nat inside source static 10.2.0.5 172.16.0.126 '
 'vrf nrouter-3ea5f9-0000002 redundancy neutron-hsrp-1064-3000',
 'ip nat inside source list neutron_acl_0000002_2564 pool '
 'nrouter-3ea5f9-0000002_nat_pool vrf nrouter-3ea5f9-0000002 '
 'overload',
 'ip nat pool nrouter-3ea5f9-0000002_nat_pool '
 '172.16.0.124 172.16.0.124 netmask 255.255.0.0',
 'ip route vrf nrouter-3ea5f9-0000002 0.0.0.0 0.0.0.0 '
 'Port-channel10.3000 172.16.0.1',
 'ip access-list standard neutron_acl_0000002_2564',
 <IOSCfgLine # 83 'interface Port-channel10.2564'>,
 <IOSCfgLine # 96 'interface Port-channel10.3000'>,
 'nrouter-3ea5f9-0000002']
```

networking_cisco.tests.unit.cisco.cfg_agent.test_asr1k_routing_driver module

class networking_cisco.tests.unit.cisco.cfg_agent.test_asr1k_routing_driver.**ASR1kRoutingDr**

Bases: neutron.tests.base.BaseTestCase, *networking_cisco.tests.unit.cisco.cfg_agent.cfg_agent_test_support.CfgAgentTestSupportMixin*

assert_edit_run_cfg(*snippet_name*, *args*)

setUp()

tearDown()

test_disable_interface_redundancy_router()

test_disable_interface_user_visible_router()

test_driver_disable_internal_network_NAT()

test_driver_disable_internal_network_NAT_with_multi_region()

test_driver_enable_internal_network_NAT()

test_driver_enable_internal_network_NAT_with_multi_region()

test_enable_interface_redundancy_router()

test_enable_interface_user_visible_router()

test_external_gateway_removed_global_router()

test_external_gateway_removed_non_ha()

test_external_gateway_removed_redundancy_router()

test_external_gateway_removed_user_visible_router()

test_external_gateway_removed_with_multi_region()

test_external_network_added_non_ha()

test_external_network_added_redundancy_router()

test_external_network_added_user_visible_router()

test_external_network_added_with_multi_region()

test_floating_ip_added()

test_floating_ip_added_with_multi_region()

test_floating_ip_removed()

test_floating_ip_removed_with_multi_region()

test_get_configuration()

test_internal_network_added()

test_internal_network_added_global_router()

test_internal_network_added_global_router_secondary_subnet()

test_internal_network_added_global_router_with_multi_region()

test_internal_network_added_global_router_with_multi_region_sec_sn()

test_internal_network_added_with_multi_region()

networking_cisco.tests.unit.cisco.cfg_agent.test_cfg_agent module

class networking_cisco.tests.unit.cisco.cfg_agent.test_cfg_agent.**TestCiscoCfgAgentWithState**

Bases: neutron.tests.base.BaseTestCase

setUp()

test_agent_registration_fail_always()

test_agent_registration_no_device_mgr()

test_agent_registration_success()

test_agent_registration_success_after_2_tries()

test_assigned_hosting_devices_monitored_from_start()

test_assigned_hosting_devices_monitored_from_start_retry()

test_get_hosting_device_configuration()

test_get_hosting_device_configuration_no_hosting_device()

test_get_hosting_device_configuration_no_svc_helper()

test_no_hosting_devices_monitored_from_start_if_rpc_fail()

test_plugin_not_notified_about_revived_hosting_devices_heartbeat_off()

test_plugin_notified_about_revived_hosting_devices_heartbeat_on()

test_report_state()

test_report_state_attribute_error(*agent_registration*)

test_report_state_report_iteration_check_full_report()

test_report_state_report_iteration_check_partial_report()

networking_cisco.tests.unit.cisco.cfg_agent.test_cfg_agent.**prepare_router_data**(*enable_snat=No*
num_internal_po

networking_cisco.tests.unit.cisco.cfg_agent.test_device_status module

class networking_cisco.tests.unit.cisco.cfg_agent.test_device_status.**TestBacklogHostingDev**

Bases: neutron.tests.base.BaseTestCase

This test class covers test cases pertaining to hosting device backlog handling in the DeviceStatus class. The "backlog" represents a set of hosting-devices that should be monitored for liveness and is used to retry router update scenarios and for heartbeat / device syncing.

setUp()

test_check_backlog_above_BT_not_pingable_aboveDeadTime()
Test for backlog processing after dead time interval.

This test simulates a hosting device which has passed the created time but greater than the 'declared dead' time. Hosting device is still not pingable.

test_check_backlog_above_BT_not_pingable_below_deadtime()
Test for backlog processing in dead time interval.

This test simulates a hosting device which has passed the created time but less than the 'declared dead' time. Hosting device is still not pingable.

test_check_backlog_above_BT_reachable_hosting_device()
Test reviving a hosting device after it's been deemed unresponsive and then becomes pingable before it's deemed dead.

test_check_backlog_above_BT_revived_hosting_device()
Test reviving a hosting device after it's been deemed dead

This test simulates a hosting device which has died is now reachable again.

test_check_backlog_above_booting_time_pingable()
Test for backlog processing after booting.

Simulates a hosting device which has passed the created time. The device should now be pingable.

test_check_backlog_below_booting_time()

test_check_backlog_empty()

class networking_cisco.tests.unit.cisco.cfg_agent.test_device_status.**TestHostingDevice**(**arg*
***kw*

Bases: neutron.tests.base.BaseTestCase

setUp()

test_hosting_devices_object()

test_is_hosting_device_reachable_negative()

test_is_hosting_device_reachable_negative_dead_hd()

test_is_hosting_device_reachable_negative_heartbeat_disabled()
Even if heartbeat is disabled, unreachable hosting device should still be backlogged

test_is_hosting_device_reachable_positive()

test_is_hosting_device_reachable_positive_heartbeat_disabled()

test_is_hosting_device_reachable_positive_heartbeat_enabled()

test_test_is_hosting_device_reachable_negative_exisiting_hd()

networking_cisco.tests.unit.cisco.cfg_agent.test_device_status.**create_timestamp**(*seconds_from_*, *type='string'*)

networking_cisco.tests.unit.cisco.cfg_agent.test_routing_svc_helper module

class networking_cisco.tests.unit.cisco.cfg_agent.test_routing_svc_helper.**TestBasicRouting**

Bases: neutron.tests.base.BaseTestCase, *networking_cisco.tests.unit.cisco.cfg_agent.cfg_agent_test_support.CfgAgentTestSupportMixin*

setUp()

test_collect_state()

test_get_router_ids_from_removed_devices_info()

test_obfuscated_password_is_unobfuscated_for_driver()

test_process_global_msn_router()

test_process_global_router()

test_process_msn_router()

test_process_router(*test_admin_state=True*)

test_process_router_delete()

test_process_router_internal_network_added_raises_HAMissingError()

test_process_router_internal_network_added_unexpected_error()

test_process_router_internal_network_removed_unexpected_error()

test_process_router_multiple_ports_on_same_multiple_subnet_network()

test_process_router_throw_config_error()

test_process_router_throw_multiple_ipv4_subnets_error()

test_process_router_throw_no_ip_address_on_subnet_error()

test_process_router_throw_session_close()

test_process_routers()

test_process_routers_floatingips_add()

test_process_routers_floatingips_remap()

test_process_routers_floatingips_remove()

test_process_routers_rearrange_for_global()

test_process_routers_skips_routers_on_other_hosting_devices()

test_process_services_full_sync_different_devices(*mock_spawn*)

test_process_services_full_sync_same_device(*mock_spawn*)

test_process_services_with_deviceid(*mock_spawn*)

test_process_services_with_removed_routers(*mock_spawn*)

test_process_services_with_removed_routers_info(*mock_spawn*)

test_process_services_with_rpc_error(*mock_spawn*)

test_process_services_with_updated_routers(*mock_spawn*)

test_router_deleted()

test_routers_updated()

test_routers_with_admin_state_down()

test_routing_table_update()

test_sort_resources_per_hosting_device()

class networking_cisco.tests.unit.cisco.cfg_agent.test_routing_svc_helper.**TestDeviceSyncOpe**

Bases: neutron.tests.base.BaseTestCase

setUp()

test_handle_sync_devices()

test_handle_sync_devices_exceed_max_retries()

test_handle_sync_devices_retry()

test_message_timeout_reduces_sync_chunk_size()

test_successful_fetch_increases_sync_chunk_size()

class networking_cisco.tests.unit.cisco.cfg_agent.test_routing_svc_helper.**TestRouterInfo**(**a*
**

Bases: neutron.tests.base.BaseTestCase, *networking_cisco.tests.unit.cisco.cfg_agent.cfg_agent_test_support.CfgAgentTestSupportMixin*

setUp()

test_router_info_create()

test_router_info_create_snat_disabled()

test_router_info_create_with_router()

networking_cisco.tests.unit.cisco.cfg_agent.test_routing_svc_helper_aci module

class networking_cisco.tests.unit.cisco.cfg_agent.test_routing_svc_helper_aci.**TestBasicRout**

Bases: *networking_cisco.tests.unit.cisco.cfg_agent.test_routing_svc_helper.TestBasicRoutingOperations*

setUp()

test_collect_state()

test_get_router_ids_from_removed_devices_info()

test_obfuscated_password_is_unobfuscated_for_driver()

test_process_global_msn_router()

test_process_global_router()

test_process_msn_router()

test_process_router()

test_process_router_2_rids_1_vrf()

test_process_router_delete()

test_process_router_internal_network_added_raises_HAMissingError()

test_process_router_internal_network_added_unexpected_error()

test_process_router_internal_network_removed_unexpected_error()

test_process_router_multiple_ports_on_same_multiple_subnet_network()

test_process_router_throw_config_error()

test_process_router_throw_multiple_ipv4_subnets_error()

test_process_router_throw_no_ip_address_on_subnet_error()

test_process_router_throw_session_close()

test_process_routers()

test_process_routers_floatingips_add()

test_process_routers_floatingips_remap()

test_process_routers_floatingips_remove()

test_process_routers_rearrange_for_global()

test_process_routers_skips_routers_on_other_hosting_devices()

test_process_services_full_sync_different_devices(*mock_spawn*)

test_process_services_full_sync_same_device(*mock_spawn*)

test_process_services_with_deviceid(*mock_spawn*)

test_process_services_with_removed_routers(*mock_spawn*)

test_process_services_with_removed_routers_info(*mock_spawn*)

test_process_services_with_rpc_error(*mock_spawn*)

test_process_services_with_updated_routers(*mock_spawn*)

test_router_deleted()

test_routers_updated()

test_routers_with_admin_state_down()

test_routing_table_update()

test_sort_resources_per_hosting_device()

class networking_cisco.tests.unit.cisco.cfg_agent.test_routing_svc_helper_aci.**TestNetworkR**

Bases: neutron.tests.base.BaseTestCase, *networking_cisco.tests.unit.cisco.cfg_agent.cfg_agent_test_support.CfgAgentTestSupportMixin*

setUp()

test_process_router_2_rids_1_vrf_1_network()

test_process_router_2_rids_1_vrf_2_networks()

test_process_router_2_rids_2_vrfs_1_network()

test_process_router_2_rids_2_vrfs_2_networks()

networking_cisco.tests.unit.cisco.cfg_agent.test_routing_svc_helper_aci.**create_hosting_info**

Module contents

networking_cisco.tests.unit.cisco.common package

Submodules

networking_cisco.tests.unit.cisco.common.test_htparser module

class networking_cisco.tests.unit.cisco.common.test_htparser.**TestHTParser**(**args*, ***kwds*)

Bases: neutron.tests.base.BaseTestCase

setUp()

test_build_indent_based_list__multiline_indent()

test_build_indent_based_list__multiline_noindent()

test_build_indent_based_list__singleline_comment()

test_build_indent_based_list__singleline_indent()

test_build_indent_based_list__singleline_noindent()

test_find_children_acl()

test_find_children_acl_multiline()

test_find_children_acl_no_find()

test_find_children_interface()

test_find_children_interface_spec()

test_find_lines_multiple()

test_find_lines_mutliline()

test_find_lines_similiarlines()

test_find_lines_singleline()

test_find_lines_vrf_def()

test_find_objects()

test_find_objects_no_find()

test_init__multiline_blank()

test_init__multiline_mixed()

test_init__multiline_mixed_different_indent()

test_re_match_nat()

test_re_search_children_acl()

test_re_search_children_interface()

Module contents

networking_cisco.tests.unit.cisco.cpnr package

Submodules

networking_cisco.tests.unit.cisco.cpnr.fake_networks module

networking_cisco.tests.unit.cisco.cpnr.test_cpnr_client module

class networking_cisco.tests.unit.cisco.cpnr.test_cpnr_client.**TestCpnrClient**(*methodName='runT*

Bases: unittest.case.TestCase

setUp()

test_buildurl()

test_create_ccm_host()

test_create_ccm_reverse_zone()

test_create_ccm_zone()

test_create_client_class()

test_create_client_entry()

test_create_dns_forwarder()

test_create_dns_view()

test_create_scope()

test_create_vpn()

test_delete_ccm_host()

test_delete_ccm_reverse_zone()

test_delete_ccm_zone()

test_delete_client_class()

test_delete_client_entry()

test_delete_dns_forwarder()

test_delete_dns_view()

test_delete_scope()

test_delete_vpn()

test_get_ccm_host()

test_get_ccm_hosts()

test_get_ccm_reverse_zone()

test_get_ccm_reverse_zones()

test_get_ccm_zone()

test_get_ccm_zones()

test_get_client_class()

test_get_client_classes()

test_get_client_entries()

test_get_client_entry()

test_get_dhcp_server()

test_get_dns_forwarder()

test_get_dns_forwarders()

test_get_dns_server()

test_get_dns_view()

test_get_dns_views()

test_get_leases()

test_get_scope()

test_get_scopes()

test_get_vpns()

test_update_ccm_host()

test_update_ccm_reverse_zone()

test_update_ccm_zone()

test_update_client_class()

test_update_client_entry()

test_update_dhcp_server()

test_update_dns_forwarder()

test_update_dns_server()

test_update_dns_view()

test_update_scope()

test_update_vpn()

networking_cisco.tests.unit.cisco.cpnr.test_dhcp_relay module

class networking_cisco.tests.unit.cisco.cpnr.test_dhcp_relay.**TestDhcpPacket**(*methodName='runTe*

Bases: unittest.case.TestCase

get_relay_opt_hex(*value*)

test_data()

test_parse()

class networking_cisco.tests.unit.cisco.cpnr.test_dhcp_relay.**TestDhcpRelayAgent**(*methodName='*
Bases: unittest.case.TestCase

test_convert_ns_to_vpnid()

test_open_dhcp_ext_socket(*mock_socket*, *mock_netns*)

test_open_dhcp_int_socket(*mock_socket*, *mock_netns*)

networking_cisco.tests.unit.cisco.cpnr.test_dhcpopts module

class networking_cisco.tests.unit.cisco.cpnr.test_dhcpopts.**TestDhcpopts**(**args*, ***kwds*)
Bases: neutron.tests.base.BaseTestCase

test_format_bool_value()

test_format_comma_separated_value()

test_format_for_classless_static_routes_options()

test_format_for_dhcp_renewal_time_options()

test_format_for_domain_name_options()

test_format_for_ip_forwarding_options()

test_format_for_options_exception(*mock_format*)

test_format_for_options_unknown_option(*mock_format*)

test_format_int32_value()

test_format_ip_value()

test_format_none_value()

test_format_route_list_value()

test_format_route_list_value_exception()

test_format_string_value()

networking_cisco.tests.unit.cisco.cpnr.test_dns_relay module

class networking_cisco.tests.unit.cisco.cpnr.test_dns_relay.**TestDnsPacket**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_data()

test_parse()

test_set_viewid()

test_skip_over_domain_name()

class networking_cisco.tests.unit.cisco.cpnr.test_dns_relay.**TestDnsRelayAgent**(*methodName='ru*
Bases: unittest.case.TestCase

test_convert_namespace_to_viewid()

test_open_dns_ext_socket(*mock_socket*, *mock_netns*)

test_open_dns_int_socket(*mock_socket*, *mock_netns*)

networking_cisco.tests.unit.cisco.cpnr.test_model module

class networking_cisco.tests.unit.cisco.cpnr.test_model.**TestModel**(**args*, ***kwds*)

Bases: neutron.tests.base.BaseTestCase

test_get_version(*mock_client*)

test_network_create(*mock_client*)

test_network_delete(*mock_client*)

test_network_init(*mock_client*)

test_policy_from_port(*mock_client*)

test_policy_from_subnet(*mock_client*)

test_port_add(*mock_client*)

test_port_remove(*mock_client*)

test_recover_networks(*mock_client*)

test_reload(*mock_client*)

test_scope_from_subnet(*mock_client*)

networking_cisco.tests.unit.cisco.cpnr.test_netns module

class networking_cisco.tests.unit.cisco.cpnr.test_netns.**TestNetNs**(*methodName='runTest'*)

Bases: unittest.case.TestCase

test_iflist(*mock_check_output*)

test_nslist(*mock_listdir*, *mock_path*)

Module contents

networking_cisco.tests.unit.cisco.device_manager package

Submodules

networking_cisco.tests.unit.cisco.device_manager.device_manager_test_support module

class networking_cisco.tests.unit.cisco.device_manager.device_manager_test_support.**DeviceMa**

Bases: object

class networking_cisco.tests.unit.cisco.device_manager.device_manager_test_support.**FakeReso**

Bases: object

class networking_cisco.tests.unit.cisco.device_manager.device_manager_test_support.**TestCore**
Bases: neutron.tests.unit.extensions.test_l3.TestNoL3NatPlugin, *networking_cisco.plugins.cisco.db.scheduler.cfg_agentschedulers_db.CfgAgentSchedulerDbMixin*, *networking_cisco.plugins.cisco.db.device_manager.hosting_device_manager_db.HostingDeviceManagerMixin*

cleanup_after_test()
This function should be called in the TearDown() function of test classes that use the plugin.

Reset all class variables to their default values. This is needed to avoid tests to pollute subsequent tests.

supported_extension_aliases = ['agent', 'external-net', 'cisco-cfg-agent-scheduler', '

class networking_cisco.tests.unit.cisco.device_manager.device_manager_test_support.**TestDevi**
Bases: object

get_actions()

get_request_extensions()

get_resources()

networking_cisco.tests.unit.cisco.device_manager.plugging_test_driver module

class networking_cisco.tests.unit.cisco.device_manager.plugging_test_driver.**TestPluggingDri**
Bases: *networking_cisco.plugins.cisco.device_manager.plugging_drivers.noop_plugging_driver.NoopPluggingDriver*

Driver class for unit tests.

allocate_hosting_port(*context*, *router_id*, *port_db*, *network_type*, *hosting_device_id*)

create_hosting_device_resources(*context*, *complementary_id*, *tenant_id*, *mgmt_context*, *max_hosted*)

delete_hosting_device_resources(*context*, *tenant_id*, *mgmt_port*, ***kwargs*)

extend_hosting_port_info(*context*, *port_db*, *hosting_device*, *hosting_info*)

get_hosting_device_resources(*context*, *id*, *complementary_id*, *tenant_id*, *mgmt_nw_id*)

setup_logical_port_connectivity(*context*, *port_db*, *hosting_device_id*)

teardown_logical_port_connectivity(*context*, *port_db*, *hosting_device_id*)

networking_cisco.tests.unit.cisco.device_manager.test_aci_vlan_trunking_driver module

class networking_cisco.tests.unit.cisco.device_manager.test_aci_vlan_trunking_driver.**FakePo**

Bases: object

get(*name*)

class networking_cisco.tests.unit.cisco.device_manager.test_aci_vlan_trunking_driver.**TestAc**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3RouterApplianceTestCaseBase*, neutron.tests.unit.extensions.test_l3.L3NatTestCaseMixin

Test class for Base ACI VLAN Trunking Plugging driver

This class tests the functionality of the ACI VLAN Trunking Plugging driver, which is indpendent of the workflow used (GBP or Neutron)

configure_routertypes = False

router_type = 'ASR1k_Neutron_router'

setUp()

tearDown()

test_config_sanity_check()

test_create_hosting_device_resources()

test_create_hosting_device_resources_no_mgmt_context()

test_delete_hosting_device_resources()

test_get_hosting_device_resources_by_complementary_id()

test_get_hosting_device_resources_by_device_id()

test_no_driver()

test_transit_nets_cfg_invalid_file_format()

class networking_cisco.tests.unit.cisco.device_manager.test_aci_vlan_trunking_driver.**TestAc**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3RouterApplianceTestCaseBase*, neutron.tests.unit.extensions.test_l3.L3NatTestCaseMixin

GBP-specific workflow testing of ACI VLAN driver

This tests the GBP-specific workflow for the ACI VLAN Trunking Plugging driver.

configure_routertypes = False

router_type = 'ASR1k_Neutron_router'

setUp()

tearDown()

test_allocate_hosting_port_info_adds_segment_id()

test_allocate_hosting_port_info_exception()

test_allocate_hosting_port_vlan_network_all_unused()

test_allocate_hosting_port_vlan_network_not_found_failure()

test_allocate_hosting_port_vlan_network_vlan_already_allocated()

test_extend_hosting_port_info_adds_global_configuration()

test_extend_hosting_port_info_adds_interface_configuration()

test_extend_hosting_port_info_adds_segmentation_id_external()

test_extend_hosting_port_info_adds_segmentation_id_internal()

test_extend_hosting_port_info_adds_snat_subnets()

class networking_cisco.tests.unit.cisco.device_manager.test_aci_vlan_trunking_driver.**TestAc**

Bases: *networking_cisco.tests.unit.cisco.device_manager.test_aci_vlan_trunking_driver.TestAciVLANTrunkingPlugDriverGbp*

Neutron-specific workflow testing of ACI VLAN driver

This tests the Neutron-specific workflow for the ACI VLAN Trunking Plugging driver.

configure_routertypes = False

router_type = 'ASR1k_Neutron_router'

setUp()

tearDown()

test_allocate_hosting_port_info_adds_segment_id()

test_allocate_hosting_port_info_exception()

test_allocate_hosting_port_no_router()

test_allocate_hosting_port_router_no_gw()

test_allocate_hosting_port_vlan_network_all_unused()

test_allocate_hosting_port_vlan_network_not_found_failure()

test_allocate_hosting_port_vlan_network_vlan_already_allocated()

test_extend_hosting_port_adds_segmentation_id_external_1_vrf()

test_extend_hosting_port_info_adds_global_configuration()

test_extend_hosting_port_info_adds_interface_configuration()

test_extend_hosting_port_info_adds_segmentation_id_external()

test_extend_hosting_port_info_adds_segmentation_id_internal()

test_extend_hosting_port_info_adds_snat_subnets()

test_extend_hosting_port_info_no_snat_subnets_1()

test_extend_hosting_port_info_no_snat_subnets_2()

test_external_net_name()

test_external_net_no_gw()

networking_cisco.tests.unit.cisco.device_manager.test_config module

class networking_cisco.tests.unit.cisco.device_manager.test_config.**TestDeviceManagerConfig**

Bases: neutron.tests.unit.testlib_api.SqlTestCase

setUp()

test_cisco_hosting_devices()

test_obtain_hosting_device_credentials_from_config()

networking_cisco.tests.unit.cisco.device_manager.test_db_device_manager module

class networking_cisco.tests.unit.cisco.device_manager.test_db_device_manager.**DeviceManager**
Bases: object

hosting_device(*template_id*, *management_port_id=None*, *fmt=None*, *admin_state_up=True*, *no_delete=False*, *set_port_device_id=True*, ***kwargs*)

hosting_device_template(*fmt=None*, *name='device_template_1'*, *enabled=True*, *host_category='VM'*, *no_delete=False*, ***kwargs*)

class networking_cisco.tests.unit.cisco.device_manager.test_db_device_manager.**TestDeviceMa**

Bases: neutron.tests.unit.db.test_db_base_plugin_v2.NeutronDbPluginV2TestCase, *networking_cisco.tests.unit.cisco.device_manager.test_db_device_manager.DeviceManagerTestCaseMixin*, *networking_cisco.tests.unit.cisco.device_manager.device_manager_test_support.DeviceManagerTestSupportMixin*

resource_prefix_map = {'hosting_device_templates': '/dev_mgr', 'hosting_devices': '/

setUp(*core_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

tearDown()

test_acquire_slots_in_other_owned_hosting_device_fails()

test_acquire_slots_release_hosting_device_ownership_affects_all()

test_acquire_slots_take_ownership_of_multi_tenant_hosting_device_fails()

test_acquire_slots_take_ownership_of_other_owned_hosting_device_fails()

test_acquire_with_slot_deficit_in_other_owned_hosting_device_fails()

test_acquire_with_slot_deficit_in_owned_hosting_device_fails()

test_acquire_with_slot_deficit_in_shared_hosting_device_fails()

test_acquire_with_slot_surplus_drop_hosting_device_ownership_succeeds()

test_acquire_with_slot_surplus_in_owned_hosting_device_succeeds()

test_acquire_with_slot_surplus_in_shared_hosting_device_succeeds()

test_acquire_with_slot_surplus_take_hosting_device_ownership_succeeds()

test_create_hw_hosting_device()

test_create_hw_hosting_device_template()

test_create_nn_hosting_device_template()

test_create_vm_hosting_device()

test_create_vm_hosting_device_template()

test_delete_all_hosting_devices()

test_delete_all_hosting_devices_by_template()

test_delete_all_managed_hosting_devices()

test_delete_all_managed_hosting_devices_by_template()

test_delete_hosting_device_in_use_fails()

test_delete_hosting_device_not_in_use_succeeds()

test_delete_hosting_device_template_in_use_fails()

test_delete_hosting_device_template_not_in_use_succeeds()

test_failed_managed_vm_based_hosting_device_gets_deleted()

test_failed_non_managed_vm_based_hosting_device_not_deleted()

test_failed_non_vm_based_hosting_device_not_deleted()

test_get_device_info_for_agent()

test_get_device_info_for_agent_no_mgmt_port()

test_get_hosting_device_configuration()

test_get_hosting_device_configuration_no_agent_found()

test_get_hosting_device_driver()

test_get_non_existent_hosting_device_driver_returns_none()

test_get_non_existent_plugging_device_driver_returns_none()

test_get_plugging_device_driver()

test_hosting_device_policy()

test_hosting_device_template_policy()

test_hw_based_hosting_device_no_change()

test_list_hosting_device_templates()

test_list_hosting_devices()

test_release_all_slots_by_negative_num_argument_owned_hosting_device()

test_release_all_slots_by_negative_num_argument_shared_hosting_device()

test_release_all_slots_returns_hosting_device_ownership()

test_release_allocated_slots_in_owned_hosting_device_succeeds()

test_release_allocated_slots_in_shared_hosting_device_succeeds()

test_release_slots_in_other_owned_hosting_device_fails()

test_release_too_many_slots_in_other_owned_hosting_device_fails()

test_release_too_many_slots_in_owned_hosting_device_fails()

test_release_too_many_slots_in_shared_hosting_device_fails()

test_show_hosting_device()

test_show_hosting_device_template()

test_update_hosting_device()

test_update_hosting_device_template()

test_vm_based_hosting_device_excessive_slot_deficit_adds_slots()

test_vm_based_hosting_device_excessive_slot_deficit_no_credentials()

test_vm_based_hosting_device_excessive_slot_surplus_removes_slots()

test_vm_based_hosting_device_marginal_slot_deficit_no_change()

test_vm_based_hosting_device_marginal_slot_surplus_no_change()

networking_cisco.tests.unit.cisco.device_manager.test_device_manager_callbacks module

class networking_cisco.tests.unit.cisco.device_manager.test_device_manager_callbacks.**TestC**

Bases: neutron.tests.base.BaseTestCase

setUp()

test_get_hosting_devices_for_agent()

test_report_for_duty_triggers_scheduling()

test_report_non_responding_hosting_devices()

test_update_hosting_device_status_all()

test_update_hosting_device_status_multiple()

test_update_hosting_device_status_to_active()

test_update_hosting_device_status_to_dead()

test_update_hosting_device_status_to_error()

test_update_hosting_device_status_to_not_responding()

networking_cisco.tests.unit.cisco.device_manager.test_extension_ciscohostingdevicemanager module

class networking_cisco.tests.unit.cisco.device_manager.test_extension_ciscohostingdevicema

Bases: neutron.tests.unit.extensions.base.ExtensionTestCase

fmt = 'json'

setUp()

test_create_hosting_device()

test_create_hosting_device_template()

test_hosting_device_delete()

test_hosting_device_get()

test_hosting_device_list()

test_hosting_device_template_delete()

test_hosting_device_template_get()

test_hosting_device_template_list()

test_hosting_device_template_update()

test_hosting_device_update()

class networking_cisco.tests.unit.cisco.device_manager.test_extension_ciscohostingdevicema

Bases: neutron.tests.base.BaseTestCase

test_convert_validate_driver()

test_convert_validate_port_value()

networking_cisco.tests.unit.cisco.device_manager.test_hosting_device_cfg_agent_scheduler module

class networking_cisco.tests.unit.cisco.device_manager.test_hosting_device_cfg_agent_schedu
Bases: neutron.tests.unit.db.test_agentschedulers_db.AgentSchedulerTestMixIn

class networking_cisco.tests.unit.cisco.device_manager.test_hosting_device_cfg_agent_schedu

Bases: *networking_cisco.tests.unit.cisco.device_manager.test_hosting_device_cfg_agent_scheduler.HostingDeviceConfigAgentSchedulerTestCaseBase*

mock_cfg_agent_notifiers = False

setUp(*core_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_hosting_device_assign_from_cfg_agent_notification_when_schedule()

test_hosting_device_assign_to_cfg_agent_notification()

test_hosting_device_unassign_from_cfg_agent_notification()

class networking_cisco.tests.unit.cisco.device_manager.test_hosting_device_cfg_agent_schedu

Bases: *networking_cisco.tests.unit.cisco.device_manager.test_hosting_device_cfg_agent_scheduler.HostingDeviceConfigAgentSchedulerTestCaseBase*

test__check_config_agents_auto_adds_new_cfg_agents()

test__check_config_agents_dead_cfg_agent_triggers_hd_rescheduling()

test__check_config_agents_stops_monitoring_non_existent_cfg_agents()

test__reschedule_hosting_devices_no_other_cfg_agent()

test__reschedule_hosting_devices_to_other_cfg_agent()

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_assigned_hosting_device_assign_to_cfg_agent()

test_get_cfg_agents()

test_get_cfg_agents_filtered()

test_get_cfg_agents_for_hosting_devices()

test_get_cfg_agents_for_hosting_devices_cfg_agent_admin_down()

test_get_cfg_agents_for_hosting_devices_cfg_agent_admin_down_no_sched()

test_get_cfg_agents_for_hosting_devices_no_schedule()

test_get_cfg_agents_for_hosting_devices_reschedules_from_dead()

test_hosting_device_assign_to_cfg_agent()

test_hosting_device_assign_to_cfg_agent_two_times()

test_hosting_device_assign_to_cfg_agent_with_admin_state_down()

test_hosting_device_assign_to_non_existing_cfg_agent()

test_hosting_device_assign_to_non_existing_hosting_device()

test_hosting_device_scheduling_policy()

test_hosting_device_unassign_from_hosting_device()

test_hosting_device_unassign_from_non_existing_cfg_agent()

test_hosting_device_unassign_from_non_existing_hosting_device()

test_list_cfg_agents_handling_hosting_device()

test_list_cfg_agents_handling_non_existent_hosting_device()

test_list_cfg_agents_handling_unassigned_hosting_device()

test_list_hosting_devices_by_cfg_agent()

test_list_hosting_devices_by_cfg_agent_with_non_existing_cfg_agent()

test_unassigned_hosting_device_unassign_from_hosting_device()

class networking_cisco.tests.unit.cisco.device_manager.test_hosting_device_cfg_agent_schedu

Bases: neutron.tests.unit.db.test_db_base_plugin_v2.NeutronDbPluginV2TestCase, *networking_cisco.tests.unit.cisco.device_manager.test_db_device_manager.DeviceManagerTestCaseMixin*, *networking_cisco.tests.unit.cisco.device_manager.device_manager_test_support.DeviceManagerTestSupportMixin*, *networking_cisco.tests.unit.cisco.device_manager.test_hosting_device_cfg_agent_schedul HostingDeviceCfgAgentSchedulerTestMixIn*

host_category = 'Hardware'

mock_cfg_agent_notifiers = True

resource_prefix_map = {'hosting_device_templates': '/dev_mgr', 'hosting_devices': '/

setUp(*core_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

class networking_cisco.tests.unit.cisco.device_manager.test_hosting_device_cfg_agent_schedu

Bases: *networking_cisco.tests.unit.cisco.device_manager.test_hosting_device_cfg_agent_scheduler.HostingDeviceConfigAgentSchedulerTestCaseBase*

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_random_scheduling()

class networking_cisco.tests.unit.cisco.device_manager.test_hosting_device_cfg_agent_schedu

Bases: *networking_cisco.tests.unit.cisco.device_manager.test_hosting_device_cfg_agent_scheduler.HostingDeviceConfigAgentSchedulerTestCaseBase*

setUp(*core_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_stingy_scheduling()

networking_cisco.tests.unit.cisco.device_manager.test_hw_vlan_trunking_plugging_driver module

class networking_cisco.tests.unit.cisco.device_manager.test_hw_vlan_trunking_plugging_drive

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3RouterApplianceTestCaseBase*, neutron.tests.unit.extensions.test_l3.L3NatTestCaseMixin

configure_routertypes = False

router_type = 'ASR1k_Neutron_router'

setUp()

tearDown()

test_allocate_hosting_port_vlan_network_all_unused()

test_allocate_hosting_port_vlan_network_not_found_failure()

test_allocate_hosting_port_vlan_network_vlan_already_allocated()

test_create_hosting_device_resources()

test_create_hosting_device_resources_no_mgmt_context()

test_delete_hosting_device_resources()

test_extend_hosting_port_info_adds_segmentation_id_external()

test_extend_hosting_port_info_adds_segmentation_id_internal()

test_get_hosting_device_resources_by_complementary_id()

test_get_hosting_device_resources_by_device_id()

networking_cisco.tests.unit.cisco.device_manager.test_vif_hotplug_plugging_driver module

class networking_cisco.tests.unit.cisco.device_manager.test_vif_hotplug_plugging_driver.**Tes**

Bases: neutron.tests.base.BaseTestCase

setUp()

test_create_hosting_device_resources()

test_create_hosting_device_resources_exception()

test_delete_resource_port_fail_always()

test_delete_resource_port_fail_only_twice()

test_delete_resource_port_handle_port_not_found()

test_setup_logical_port_connectivity(*mock_svc_vm_mgr*)

Module contents

networking_cisco.tests.unit.cisco.l3 package

Submodules

networking_cisco.tests.unit.cisco.l3.l3_router_test_support module

class networking_cisco.tests.unit.cisco.l3.l3_router_test_support.**L3RouterTestSupportMixin**
Bases: object

class networking_cisco.tests.unit.cisco.l3.l3_router_test_support.**TestL3RouterBaseExtensio**
Bases: object

get_actions()

get_request_extensions()

get_resources()

class networking_cisco.tests.unit.cisco.l3.l3_router_test_support.**TestL3RouterServicePlugi**
Bases: neutron.db.common_db_mixin.CommonDbMixin, *networking_cisco.plugins.cisco.db.l3.routertype_db.RoutertypeDbMixin*, networking_cisco.plugins.cisco.db.l3.l3_router_appliance_db.L3RouterApplianceDBMixin, *networking_cisco.plugins.cisco.db.scheduler.l3_routertype_aware_schedulers_db.L3RouterTypeAwareSchedulerDbMixin*

cleanup_after_test()
This function should be called in the TearDown() function of test classes that use the plugin.

Reset all class variables to their default values. This is needed to avoid tests to pollute subsequent tests.

get_plugin_description()

get_plugin_type()

supported_extension_aliases = ['router', 'standard-attr-description', 'routerhost', 'r

networking_cisco.tests.unit.cisco.l3.test_agent_scheduler module

class networking_cisco.tests.unit.cisco.l3.test_agent_scheduler.**L3RouterApplianceL3AgentNo**

Bases: neutron.tests.unit.db.test_agentschedulers_db.OvsL3AgentNotifierTestCase, *networking_cisco.tests.unit.cisco.l3.test_db_routertype.RoutertypeTestCaseMixin*, *networking_cisco.tests.unit.cisco.device_manager.test_db_device_manager.DeviceManagerTestCaseMixin*, *networking_cisco.tests.unit.cisco.l3.l3_router_test_support.L3RouterTestSupportMixin*, *networking_cisco.tests.unit.cisco.device_manager.device_manager_test_support.DeviceManagerTestSupportMixin*

resource_prefix_map = {'hosting_device_templates': '/dev_mgr', 'hosting_devices': '/

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

setup_coreplugin(*core_plugin=None*, *load_plugins=True*)

tearDown()

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_agent_updated_l3_agent_notification()

test_router_add_to_l3_agent_notification()

test_router_remove_from_l3_agent_notification()

class networking_cisco.tests.unit.cisco.l3.test_agent_scheduler.**L3RouterApplianceL3AgentSc**

Bases: neutron.tests.unit.db.test_agentschedulers_db.OvsAgentSchedulerTestCase, *networking_cisco.tests.unit.cisco.l3.test_db_routertype.RoutertypeTestCaseMixin*, *networking_cisco.tests.unit.cisco.device_manager.test_db_device_manager.DeviceManagerTestCaseMixin*, *networking_cisco.tests.unit.cisco.l3.l3_router_test_support.L3RouterTestSupportMixin*, *networking_cisco.tests.unit.cisco.device_manager.device_manager_test_support.DeviceManagerTestSupportMixin*

resource_prefix_map = {'hosting_device_templates': '/dev_mgr', 'hosting_devices': '/

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

setup_coreplugin(*core_plugin=None*, *load_plugins=True*)

tearDown()

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_dhcp_agent_keep_services_off()

test_dhcp_agent_keep_services_on()

test_dvr_router_csnat_rescheduling()

test_dvr_router_manual_rescheduling()

test_dvr_router_scheduling_to_only_dvr_snat_agent()

test_is_eligible_agent()

test_l3_agent_keep_services_off()

test_l3_agent_keep_services_on()

test_list_active_networks_on_not_registered_yet_dhcp_agent()

test_list_networks_hosted_by_dhcp_agent_with_invalid_agent()

test_list_router_ids_on_host_no_l3_agent()

test_list_routers_hosted_by_l3_agent_with_invalid_agent()

test_network_add_to_dhcp_agent()

test_network_add_to_dhcp_agent_with_admin_state_down()

test_network_auto_schedule_restart_dhcp_agent()

test_network_auto_schedule_with_disabled()

test_network_auto_schedule_with_hosted()

test_network_auto_schedule_with_hosted_2()

test_network_auto_schedule_with_multiple_agents()

test_network_auto_schedule_with_no_dhcp()

test_network_ha_scheduling_on_port_creation()

test_network_ha_scheduling_on_port_creation_with_new_agent()

test_network_policy()

test_network_remove_from_dhcp_agent()

test_network_scheduler_with_disabled_agent()

test_network_scheduler_with_down_agent()

test_network_scheduler_with_hosted_network()

test_network_scheduling_on_network_creation()

test_network_scheduling_on_port_creation()

test_report_states()

test_reserved_port_after_network_remove_from_dhcp_agent()

test_router_add_to_l3_agent()

test_router_add_to_l3_agent_two_times()

test_router_add_to_l3_agent_with_admin_state_down()

test_router_add_to_two_l3_agents()

test_router_auto_schedule_restart_l3_agent()

test_router_auto_schedule_with_disabled()

test_router_auto_schedule_with_hosted()

test_router_auto_schedule_with_hosted_2()

test_router_auto_schedule_with_invalid_router()

test_router_is_not_rescheduled_from_alive_agent()

test_router_is_not_rescheduled_if_agent_is_back_online()

test_router_no_reschedule_from_dead_admin_down_agent()

test_router_policy()

test_router_reschedule_failed_notification_all_attempts()

test_router_reschedule_from_dead_agent()

test_router_reschedule_no_remove_if_agent_has_dvr_service_ports()

test_router_reschedule_succeeded_after_failed_notification()

test_router_rescheduler_catches_exceptions_on_fetching_bindings()

test_router_rescheduler_catches_rpc_db_and_reschedule_exceptions()

test_router_rescheduler_iterates_after_reschedule_failure()

test_router_sync_data()

test_router_without_l3_agents()

test_rpc_sync_routers()

test_sync_dvr_router()

test_sync_router()

class networking_cisco.tests.unit.cisco.l3.test_agent_scheduler.**TestAgentSchedCorePlugin**

Bases: *networking_cisco.tests.unit.cisco.device_manager.device_manager_test_support.TestCorePlugin*, neutron.db.agentschedulers_db.DhcpAgentSchedulerDbMixin

supported_extension_aliases = ['external-net', 'agent', 'dhcp_agent_scheduler', 'dev_m

networking_cisco.tests.unit.cisco.l3.test_asr1k_routertype_driver module

class networking_cisco.tests.unit.cisco.l3.test_asr1k_routertype_driver.**Asr1kHARouterTypeDr**

Bases: *networking_cisco.tests.unit.cisco.l3.test_asr1k_routertype_driver.Asr1kRouterTypeDriverTestCase*, *networking_cisco.tests.unit.cisco.l3.test_ha_l3_router_appliance_plugin.HAL3RouterTestsMixin*

router_type = 'ASR1k_Neutron_router'

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_create_gateway_router()

test_create_gateway_router_den()

test_create_gateway_router_dt()

test_create_gateway_router_dt_den()

test_create_gateway_router_non_admin()

test_create_gateway_router_non_admin_den()

test_create_gateway_router_non_admin_dt()

test_create_gateway_router_non_admin_dt_den()

test_create_msn_gateway_router()

test_create_msn_gateway_router_den()

test_create_msn_gateway_router_dt()

test_create_msn_gateway_router_dt_den()

test_create_router_adds_no_aux_gw_port_to_global_router()

test_create_router_adds_no_aux_gw_port_to_global_router_dt()

test_create_router_adds_no_aux_gw_port_to_global_router_non_admin()

test_create_router_adds_no_aux_gw_port_to_global_router_non_admin_dt()

test_create_router_adds_no_global_router()

test_create_router_adds_no_global_router_non_admin()

test_delete_gateway_router()

test_delete_gateway_router_den()

test_delete_gateway_router_dt()

test_delete_gateway_router_dt_den()

test_delete_gateway_router_non_admin()

test_delete_gateway_router_non_admin_den()

test_delete_gateway_router_non_admin_dt()

test_delete_gateway_router_non_admin_dt_den()

test_delete_msn_gateway_router()

test_delete_msn_gateway_router_den()

test_delete_msn_gateway_router_dt()

test_delete_msn_gateway_router_dt_den()

test_router_interface_add_refused_for_unsupported_topology()

test_router_interface_add_refused_for_unsupported_topology_dt()

test_router_update_unset_gw()

test_router_update_unset_gw_den()

test_router_update_unset_gw_dt()

test_router_update_unset_gw_dt_den()

test_router_update_unset_gw_non_admin()

test_router_update_unset_gw_non_admin_den()

test_router_update_unset_gw_non_admin_dt()

test_router_update_unset_gw_non_admin_dt_den()

test_router_update_unset_msn_gw()

test_router_update_unset_msn_gw_concurrent_global_delete()

test_router_update_unset_msn_gw_concurrent_global_port_delete()

test_router_update_unset_msn_gw_concurrent_port_delete()

test_router_update_unset_msn_gw_den()

test_router_update_unset_msn_gw_dt()

test_router_update_unset_msn_gw_dt_den()

test_update_router_set_gateway()

test_update_router_set_gateway_den()

test_update_router_set_gateway_dt()

test_update_router_set_gateway_dt_den()

test_update_router_set_gateway_non_admin()

test_update_router_set_gateway_non_admin_den()

test_update_router_set_gateway_non_admin_dt()

test_update_router_set_gateway_non_admin_dt_den()

test_update_router_set_msn_gateway()

test_update_router_set_msn_gateway_den()

test_update_router_set_msn_gateway_dt()

test_update_router_set_msn_gateway_dt_den()

class networking_cisco.tests.unit.cisco.l3.test_asr1k_routertype_driver.**Asr1kRouterTypeDriv**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.L3RoutertypeAwareHostingDeviceSchedulerTestCaseBase*

router_type = 'Nexus_ToR_Neutron_router'

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_create_gateway_router()

test_create_gateway_router_den()

test_create_gateway_router_dt()

test_create_gateway_router_dt_den()

test_create_gateway_router_non_admin()

test_create_gateway_router_non_admin_den()

test_create_gateway_router_non_admin_dt()

test_create_gateway_router_non_admin_dt_den()

test_create_msn_gateway_router()

test_create_msn_gateway_router_den()

test_create_msn_gateway_router_dt()

test_create_msn_gateway_router_dt_den()

test_create_router_adds_no_aux_gw_port_to_global_router()

test_create_router_adds_no_aux_gw_port_to_global_router_dt()

test_create_router_adds_no_aux_gw_port_to_global_router_non_admin()

test_create_router_adds_no_aux_gw_port_to_global_router_non_admin_dt()

test_create_router_adds_no_global_router()

test_create_router_adds_no_global_router_non_admin()

test_delete_gateway_router()

test_delete_gateway_router_den()

test_delete_gateway_router_dt()

test_delete_gateway_router_dt_den()

test_delete_gateway_router_non_admin()

test_delete_gateway_router_non_admin_den()

test_delete_gateway_router_non_admin_dt()

test_delete_gateway_router_non_admin_dt_den()

test_delete_msn_gateway_router()

test_delete_msn_gateway_router_den()

test_delete_msn_gateway_router_dt()

test_delete_msn_gateway_router_dt_den()

test_router_interface_add_refused_for_unsupported_topology()

test_router_interface_add_refused_for_unsupported_topology_dt()

test_router_update_unset_gw()

test_router_update_unset_gw_den()

test_router_update_unset_gw_dt()

test_router_update_unset_gw_dt_den()

test_router_update_unset_gw_non_admin()

test_router_update_unset_gw_non_admin_den()

test_router_update_unset_gw_non_admin_dt()

test_router_update_unset_gw_non_admin_dt_den()

test_router_update_unset_msn_gw()

test_router_update_unset_msn_gw_concurrent_global_delete()

test_router_update_unset_msn_gw_concurrent_global_port_delete()

test_router_update_unset_msn_gw_concurrent_port_delete()

test_router_update_unset_msn_gw_den()

test_router_update_unset_msn_gw_dt()

test_router_update_unset_msn_gw_dt_den()

test_update_router_set_gateway()

test_update_router_set_gateway_den()

test_update_router_set_gateway_dt()

test_update_router_set_gateway_dt_den()

test_update_router_set_gateway_non_admin()

test_update_router_set_gateway_non_admin_den()

test_update_router_set_gateway_non_admin_dt()

test_update_router_set_gateway_non_admin_dt_den()

test_update_router_set_msn_gateway()

test_update_router_set_msn_gateway_den()

test_update_router_set_msn_gateway_dt()

test_update_router_set_msn_gateway_dt_den()

class networking_cisco.tests.unit.cisco.l3.test_asr1k_routertype_driver.**L3CfgAgentAsr1kRout**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.L3RoutertypeAwareHostingDeviceSchedulerTestCaseBase*, *networking_cisco.tests.unit.cisco.l3.test_ha_l3_router_appliance_plugin.HAL3RouterTestsMixin*

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

tearDown()

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_l3_cfg_agent_query_global_router_info()

test_l3_cfg_agent_query_global_router_info_den()

test_l3_cfg_agent_query_global_router_info_dt()

test_l3_cfg_agent_query_global_router_info_dt_den()

networking_cisco.tests.unit.cisco.l3.test_db_routertype module

class networking_cisco.tests.unit.cisco.l3.test_db_routertype.**L3TestRoutertypeExtensionMana**

Bases: *networking_cisco.tests.unit.cisco.l3.l3_router_test_support.TestL3RouterBaseExtensionManager*

get_resources()

class networking_cisco.tests.unit.cisco.l3.test_db_routertype.**RoutertypeTestCaseMixin**

Bases: object

routertype(*template_id*, *name='router type 1'*, *slot_need=2*, *fmt=None*, *no_delete=False*, ***kwargs*)

class networking_cisco.tests.unit.cisco.l3.test_db_routertype.**TestRoutertypeDBPlugin**(**args*, ***kwds*)

Bases: neutron.tests.unit.db.test_db_base_plugin_v2.NeutronDbPluginV2TestCase, *networking_cisco.tests.unit.cisco.l3.test_db_routertype.RoutertypeTestCaseMixin*, *networking_cisco.tests.unit.cisco.device_manager.test_db_device_manager.DeviceManagerTestCaseMixin*

resource_prefix_map = {'hosting_device_templates': '/dev_mgr', 'hosting_devices': '/

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test_create_routertype()

test_delete_routertype()

test_list_routertypes()

test_routertype_policy()

test_show_routertype()

test_show_routertype_non_admin()

test_update_routertype()

networking_cisco.tests.unit.cisco.l3.test_extension_routertype module

class networking_cisco.tests.unit.cisco.l3.test_extension_routertype.**RouterTypeTestCase**(**ar*, ***k*

Bases: neutron.tests.unit.extensions.base.ExtensionTestCase

fmt = 'json'

setUp()

test_create_routertype()

test_routertype_delete()

test_routertype_get()

test_routertype_list()

test_routertype_update()

class networking_cisco.tests.unit.cisco.l3.test_extension_routertype.**TestRouterTypeAttribut**

Bases: neutron.tests.base.BaseTestCase

test_convert_validate_driver()

networking_cisco.tests.unit.cisco.l3.test_ha_l3_router_appliance_plugin module

class networking_cisco.tests.unit.cisco.l3.test_ha_l3_router_appliance_plugin.**HAL3RouterApp**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3RouterApplianceNamespaceTestCase*

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test__notify_subnetpool_address_scope_update()

test_associate_to_dhcp_port_fails()

test_create_floating_non_ext_network_returns_400()

test_create_floatingip_invalid_fixed_ip_address_returns_400()

test_create_floatingip_invalid_fixed_ipv6_address_returns_400()

test_create_floatingip_invalid_floating_network_id_returns_400()

test_create_floatingip_invalid_floating_port_id_returns_400()

test_create_floatingip_ipv6_and_ipv4_network_creates_ipv4()

test_create_floatingip_ipv6_only_network_returns_400()

test_create_floatingip_no_ext_gateway_return_404()

test_create_floatingip_no_public_subnet_returns_400()

test_create_floatingip_non_admin_context_agent_notification()

test_create_floatingip_with_assoc(*expected_status='ACTIVE'*)

test_create_floatingip_with_assoc_to_ipv4_and_ipv6_port()

test_create_floatingip_with_assoc_to_ipv6_subnet()

test_create_floatingip_with_duplicated_specific_ip()

test_create_floatingip_with_multisubnet_id()

test_create_floatingip_with_specific_ip()

test_create_floatingip_with_specific_ip_non_admin()

test_create_floatingip_with_specific_ip_out_of_allocation()

test_create_floatingip_with_specific_ip_out_of_subnet()

test_create_floatingip_with_subnet_and_invalid_fip_address()

test_create_floatingip_with_subnet_id_and_fip_address()

test_create_floatingip_with_subnet_id_non_admin()

test_create_floatingip_with_wrong_subnet_id()

test_create_floatingips_native_quotas()

test_create_multiple_floatingips_same_fixed_ip_same_port()
This tests that if multiple API requests arrive to create floating IPs on same external network to same port with one fixed ip, the latter API requests would be blocked at database side.

test_create_non_router_port_device_id_of_other_teants_router_update()

test_create_router_gateway_fails_nested()

test_create_router_gateway_fails_nested_delete_router_failed()

test_create_router_port_with_device_id_of_other_teants_router()

test_create_routers_native_quotas()

test_delete_ext_net_with_disassociated_floating_ips()

test_first_floatingip_associate_notification()

test_floating_ip_direct_port_delete_returns_409()

test_floating_port_status_not_applicable()

test_floatingip_association_on_unowned_router()

test_floatingip_crd_ops()

test_floatingip_create_different_fixed_ip_same_port()
This tests that it is possible to delete a port that has multiple floating ip addresses associated with it (each floating address associated with a unique fixed address).

test_floatingip_delete_router_intf_with_port_id_returns_409()

test_floatingip_delete_router_intf_with_subnet_id_returns_409()

test_floatingip_disassociate_notification()

test_floatingip_list_with_pagination()

test_floatingip_list_with_pagination_reverse()

test_floatingip_list_with_port_id()

test_floatingip_list_with_sort()

test_floatingip_multi_external_one_internal()

test_floatingip_port_delete()

test_floatingip_same_external_and_internal()

test_floatingip_update(*expected_status='ACTIVE'*)

test_floatingip_update_different_fixed_ip_same_port()

test_floatingip_update_different_port_owner_as_admin()

test_floatingip_update_different_router()

test_floatingip_update_invalid_fixed_ip()

test_floatingip_update_same_fixed_ip_same_port()

test_floatingip_update_subnet_gateway_disabled(*expected_status='ACTIVE'*)
Attach a floating IP to an instance

Verify that the floating IP can be associated to a port whose subnet's gateway ip is not connected to the external router, but the router has an ip in that subnet.

test_floatingip_update_to_same_port_id_twice(*expected_status='ACTIVE'*)

test_floatingip_via_router_interface_returns_201()

test_floatingip_via_router_interface_returns_404()

test_floatingip_with_assoc_fails()

test_floatingip_with_invalid_create_port()

test_janitor_clears_orphaned_floatingip_port()

test_janitor_doesnt_delete_if_fixed_in_interim()

test_janitor_updates_port_device_id()

test_network_update_external()

test_network_update_external_failure()

test_nexthop_is_port_ip()

test_no_destination_route()

test_no_nexthop_route()

test_none_destination()

test_none_nexthop()

test_route_clear_routes_with_None()

test_route_update_with_external_route()

test_route_update_with_multi_routes()

test_route_update_with_one_route()

test_route_update_with_route_via_another_tenant_subnet()

test_router_add_and_remove_gateway()

test_router_add_and_remove_gateway_tenant_ctx()

test_router_add_gateway_dup_subnet1_returns_400()

test_router_add_gateway_dup_subnet2_returns_400()

test_router_add_gateway_invalid_network_returns_400()

test_router_add_gateway_multiple_subnets_ipv6()
Ensure external gateway set doesn't add excess IPs on router gw

Setting the gateway of a router to an external network with more than one IPv4 and one IPv6 subnet should only add an address from the first IPv4 subnet, an address from the first IPv6-stateful subnet, and an address from each IPv6-stateless (SLAAC and DHCPv6-stateless) subnet

test_router_add_gateway_net_not_external_returns_400()

test_router_add_gateway_no_subnet()

test_router_add_gateway_no_subnet_forbidden()

test_router_add_gateway_non_existent_network_returns_404()

test_router_add_iface_ipv6_ext_ra_subnet_returns_400()
Test router-interface-add for in-valid ipv6 subnets.

Verify that an appropriate error message is displayed when an IPv6 subnet configured to use an external_router for Router Advertisements (i.e., ipv6_ra_mode is None and ipv6_address_mode is not None) is attempted to associate with a Neutron Router.

test_router_add_interface_bad_values()

test_router_add_interface_by_port_admin_address_out_of_pool()

test_router_add_interface_by_port_cidr_overlapped_with_gateway()

test_router_add_interface_by_port_fails_nested()

test_router_add_interface_by_port_other_tenant_address_in_pool()

test_router_add_interface_by_port_other_tenant_address_out_of_pool()

test_router_add_interface_by_subnet_other_tenant_subnet_returns_400()

test_router_add_interface_cidr_overlapped_with_gateway()

test_router_add_interface_delete_port_after_failure()

test_router_add_interface_dup_port()
This tests that if multiple routers add one port as their interfaces. Only the first router's interface would be added to this port. All the later requests would return exceptions.

test_router_add_interface_dup_subnet1_returns_400()

test_router_add_interface_dup_subnet2_returns_400()

test_router_add_interface_empty_port_and_subnet_ids()

test_router_add_interface_ipv6_port_existing_network_returns_400()
Ensure unique IPv6 router ports per network id.

Adding a router port containing one or more IPv6 subnets with the same network id as an existing router port should fail. This is so there is no ambiguity regarding on which port to add an IPv6 subnet when executing router-interface-add with a subnet and no port.

test_router_add_interface_ipv6_subnet()
Test router-interface-add for valid ipv6 subnets.

Verify the valid use-cases of an IPv6 subnet where we are allowed to associate to the Neutron Router are successful.

test_router_add_interface_ipv6_subnet_without_gateway_ip()

test_router_add_interface_multiple_ipv4_subnet_port_returns_400()
Test adding router port with multiple IPv4 subnets fails.

Multiple IPv4 subnets are not allowed on a single router port. Ensure that adding a port with multiple IPv4 subnets to a router fails.

test_router_add_interface_multiple_ipv4_subnets()
Test router-interface-add for multiple ipv4 subnets.

Verify that adding multiple ipv4 subnets from the same network to a router places them all on different router interfaces.

test_router_add_interface_multiple_ipv6_subnet_port()
A port with multiple IPv6 subnets can be added to a router

Create a port with multiple associated IPv6 subnets and attach it to a router. The action should succeed.

test_router_add_interface_multiple_ipv6_subnets_different_net()
Test router-interface-add for ipv6 subnets on different networks.

Verify that adding multiple ipv6 subnets from different networks to a router places them on different router interfaces.

test_router_add_interface_multiple_ipv6_subnets_same_net()
Test router-interface-add for multiple ipv6 subnets on a network.

Verify that adding multiple ipv6 subnets from the same network to a router places them all on the same router interface.

test_router_add_interface_no_data_returns_400()

test_router_add_interface_overlapped_cidr_returns_400()

test_router_add_interface_port()

test_router_add_interface_port_bad_tenant_returns_404()

test_router_add_interface_port_without_ips()

test_router_add_interface_subnet()

test_router_add_interface_subnet_with_bad_tenant_returns_404()

test_router_add_interface_subnet_with_port_from_other_tenant()

test_router_add_interface_with_both_ids_returns_400()

test_router_clear_gateway_callback_failure_returns_409()

test_router_concurrent_delete_upon_subnet_create()

test_router_create()

test_router_create_call_extensions()

test_router_create_with_gwinfo()

test_router_create_with_gwinfo_ext_ip()

test_router_create_with_gwinfo_ext_ip_non_admin()

test_router_create_with_gwinfo_ext_ip_subnet()

test_router_delete()

test_router_delete_callback()

test_router_delete_denied_for_plugin_managed_router()

test_router_delete_dhcpv6_stateless_subnet_inuse_returns_409()

test_router_delete_ipv6_slaac_subnet_inuse_returns_409()

test_router_delete_race_with_interface_add()

test_router_delete_subnet_inuse_returns_409()

test_router_delete_with_floatingip_existed_returns_409()

test_router_delete_with_port_existed_returns_409()

test_router_interface_add_denied_for_plugin_managed_router()

test_router_interface_in_use_by_route()

test_router_list()

test_router_list_with_pagination()

test_router_list_with_pagination_reverse()

test_router_list_with_parameters()

test_router_list_with_sort()

test_router_remove_interface_callback_failure_returns_409()

test_router_remove_interface_inuse_returns_409()

test_router_remove_interface_nothing_returns_400()

test_router_remove_interface_returns_200()

test_router_remove_interface_with_both_ids_returns_200()

test_router_remove_interface_wrong_port_returns_404()

test_router_remove_interface_wrong_subnet_returns_400()

test_router_remove_ipv6_subnet_from_interface()
Delete a subnet from a router interface

Verify that deleting a subnet with router-interface-delete removes that subnet when there are multiple subnets on the interface and removes the interface when it is the last subnet on the interface.

test_router_show()

test_router_specify_id_backend()

test_router_update()

test_router_update_delete_routes()

test_router_update_denied_for_plugin_managed_router()

test_router_update_gateway()

test_router_update_gateway_add_multiple_prefixes_ipv6()

test_router_update_gateway_to_empty_with_existed_floatingip()

test_router_update_gateway_upon_subnet_create_ipv6()

test_router_update_gateway_upon_subnet_create_max_ips_ipv6()
Create subnet should not cause excess fixed IPs on router gw

If a router gateway port has the maximum of one IPv4 and one IPv6 fixed, create subnet should not add any more IP addresses to the port (unless this is the subnet is a SLAAC/DHCPv6-stateless subnet in which case the addresses are added automatically)

test_router_update_gateway_with_different_external_subnet()

test_router_update_gateway_with_existed_floatingip()

test_router_update_gateway_with_external_ip_used_by_gw()

test_router_update_gateway_with_invalid_external_ip()

test_router_update_gateway_with_invalid_external_subnet()

test_router_update_on_external_port()

test_router_update_with_dup_address()

test_router_update_with_invalid_ip_address()

test_router_update_with_invalid_nexthop_ip()

test_router_update_with_nexthop_is_outside_port_subnet()

test_router_update_with_too_many_routes()

test_routes_update_for_multiple_routers()

test_two_fips_one_port_invalid_return_409()

test_update_port_device_id_to_different_tenants_router()

test_update_subnet_gateway_for_external_net()
Test to make sure notification to routers occurs when the gateway ip address of a subnet of the external network is changed.

class networking_cisco.tests.unit.cisco.l3.test_ha_l3_router_appliance_plugin.**HAL3RouterApp**

Bases: *networking_cisco.tests.unit.cisco.l3.test_ha_l3_router_appliance_plugin.HAL3RouterTestsMixin*, *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3RouterApplianceVMTestCase*

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test__notify_subnetpool_address_scope_update()

test_associate_to_dhcp_port_fails()

test_create_floating_non_ext_network_returns_400()

test_create_floatingip_invalid_fixed_ip_address_returns_400()

test_create_floatingip_invalid_fixed_ipv6_address_returns_400()

test_create_floatingip_invalid_floating_network_id_returns_400()

test_create_floatingip_invalid_floating_port_id_returns_400()

test_create_floatingip_ipv6_and_ipv4_network_creates_ipv4()

test_create_floatingip_ipv6_only_network_returns_400()

test_create_floatingip_no_ext_gateway_return_404()

test_create_floatingip_no_public_subnet_returns_400()

test_create_floatingip_non_admin_context_agent_notification()

test_create_floatingip_with_assoc(*expected_status='ACTIVE'*)

test_create_floatingip_with_assoc_to_ipv4_and_ipv6_port()

test_create_floatingip_with_assoc_to_ipv6_subnet()

test_create_floatingip_with_duplicated_specific_ip()

test_create_floatingip_with_multisubnet_id()

test_create_floatingip_with_specific_ip()

test_create_floatingip_with_specific_ip_non_admin()

test_create_floatingip_with_specific_ip_out_of_allocation()

test_create_floatingip_with_specific_ip_out_of_subnet()

test_create_floatingip_with_subnet_and_invalid_fip_address()

test_create_floatingip_with_subnet_id_and_fip_address()

test_create_floatingip_with_subnet_id_non_admin()

test_create_floatingip_with_wrong_subnet_id()

test_create_floatingips_native_quotas()

test_create_ha_router_when_ha_support_disabled_fails()

test_create_ha_router_with_defaults()

test_create_ha_router_with_defaults_non_admin_succeeds()

test_create_ha_router_with_disabled_ha_type_fails()

test_create_ha_router_with_ha_specification()

test_create_ha_router_with_ha_specification_invalid_HA_type_fails()

test_create_ha_router_with_ha_specification_non_admin_fails()

test_create_ha_router_with_ha_specification_validation_fails()

test_create_multiple_floatingips_same_fixed_ip_same_port()
This tests that if multiple API requests arrive to create floating IPs on same external network to same port with one fixed ip, the latter API requests would be blocked at database side.

test_create_non_gw_ha_router_when_ha_support_disabled_fails()

test_create_non_gw_ha_router_with_defaults()

test_create_non_gw_ha_router_with_defaults_non_admin_succeeds()

test_create_non_gw_ha_router_with_disabled_ha_type_fails()

test_create_non_gw_ha_router_with_ha_spec_invalid_HA_type_fails()

test_create_non_gw_ha_router_with_ha_spec_non_admin_fails()

test_create_non_gw_ha_router_with_ha_specification()

test_create_non_gw_ha_router_with_ha_specification_validation_fails()

test_create_non_router_port_device_id_of_other_teants_router_update()

test_create_router_gateway_fails_nested()

test_create_router_gateway_fails_nested_delete_router_failed()

test_create_router_port_with_device_id_of_other_teants_router()

test_create_routers_native_quotas()

test_decrease_ha_router_redundancy_level()

test_decrease_non_gw_ha_router_redundancy_level()

test_delete_ext_net_with_disassociated_floating_ips()

test_enable_ha_on_non_gw_router_no_itfcs_succeeds()

test_enable_ha_on_non_gw_router_non_admin_succeeds()

test_enable_ha_on_non_gw_router_succeeds()

test_enable_ha_on_non_gw_router_succeeds_no_ha_spec()

test_enable_ha_on_router_no_itfcs_succeeds_succeeds()

test_enable_ha_on_router_non_admin_succeeds()

test_enable_ha_on_router_succeeds()

test_enable_ha_on_router_succeeds_no_ha_spec()

test_enable_ha_when_ha_support_disabled_fails()

test_enable_ha_when_ha_support_disabled_fails_non_gw()

test_enable_ha_with_disabled_ha_type_fails()

test_enable_ha_with_disabled_ha_type_fails_non_gw()

test_first_floatingip_associate_notification()

test_floating_ip_direct_port_delete_returns_409()

test_floating_port_status_not_applicable()

test_floatingip_association_on_unowned_router()

test_floatingip_crd_ops()

test_floatingip_create_different_fixed_ip_same_port()
This tests that it is possible to delete a port that has multiple floating ip addresses associated with it (each floating address associated with a unique fixed address).

test_floatingip_delete_router_intf_with_port_id_returns_409()

test_floatingip_delete_router_intf_with_subnet_id_returns_409()

test_floatingip_disassociate_notification()

test_floatingip_list_with_pagination()

test_floatingip_list_with_pagination_reverse()

test_floatingip_list_with_port_id()

test_floatingip_list_with_sort()

test_floatingip_multi_external_one_internal()

test_floatingip_port_delete()

test_floatingip_same_external_and_internal()

test_floatingip_update(*expected_status='ACTIVE'*)

test_floatingip_update_different_fixed_ip_same_port()

test_floatingip_update_different_port_owner_as_admin()

test_floatingip_update_different_router()

test_floatingip_update_invalid_fixed_ip()

test_floatingip_update_same_fixed_ip_same_port()

test_floatingip_update_subnet_gateway_disabled(*expected_status='ACTIVE'*)
Attach a floating IP to an instance

Verify that the floating IP can be associated to a port whose subnet's gateway ip is not connected to the external router, but the router has an ip in that subnet.

test_floatingip_update_to_same_port_id_twice(*expected_status='ACTIVE'*)

test_floatingip_via_router_interface_returns_201()

test_floatingip_via_router_interface_returns_404()

test_floatingip_with_assoc_fails()

test_floatingip_with_invalid_create_port()

test_ha_non_gw_router_add_gateway_succeeds()

test_ha_router_add_and_remove_interface_port()

test_ha_router_disable_ha_non_admin_succeeds()

test_ha_router_disable_ha_succeeds()

test_ha_router_remove_gateway_succeeds()

test_ha_update_admin_state_up()

test_hidden_port_creation_includes_dns_attribute()

test_increase_ha_router_redundancy_level()

test_increase_non_gw_ha_router_redundancy_level()

test_janitor_clears_orphaned_floatingip_port()

test_janitor_doesnt_delete_if_fixed_in_interim()

test_janitor_updates_port_device_id()

test_name_change_of_redundancy_router_does_not_create_another_one()

test_network_update_external()

test_network_update_external_failure()

test_nexthop_is_port_ip()

test_no_destination_route()

test_no_nexthop_route()

test_non_gw_ha_router_add_and_remove_interface_port()

test_non_gw_ha_router_disable_ha_non_admin_succeeds()

test_non_gw_ha_router_disable_ha_succeeds()

test_none_destination()

test_none_nexthop()

test_redundancy_router_routes_includes_user_visible_router()

test_redundancy_router_routes_is_from_user_visible_router()

test_route_clear_routes_with_None()

test_route_update_with_external_route()

test_route_update_with_multi_routes()

test_route_update_with_one_route()

test_route_update_with_route_via_another_tenant_subnet()

test_router_add_and_remove_gateway()

test_router_add_and_remove_gateway_tenant_ctx()

test_router_add_gateway_dup_subnet1_returns_400()

test_router_add_gateway_dup_subnet2_returns_400()

test_router_add_gateway_invalid_network_returns_400()

test_router_add_gateway_multiple_subnets_ipv6()
Ensure external gateway set doesn't add excess IPs on router gw

Setting the gateway of a router to an external network with more than one IPv4 and one IPv6 subnet should only add an address from the first IPv4 subnet, an address from the first IPv6-stateful subnet, and an address from each IPv6-stateless (SLAAC and DHCPv6-stateless) subnet

test_router_add_gateway_net_not_external_returns_400()

test_router_add_gateway_no_subnet()

test_router_add_gateway_no_subnet_forbidden()

test_router_add_gateway_non_existent_network_returns_404()

test_router_add_iface_ipv6_ext_ra_subnet_returns_400()
Test router-interface-add for in-valid ipv6 subnets.

Verify that an appropriate error message is displayed when an IPv6 subnet configured to use an external_router for Router Advertisements (i.e., ipv6_ra_mode is None and ipv6_address_mode is not None) is attempted to associate with a Neutron Router.

test_router_add_interface_bad_values()

test_router_add_interface_by_port_admin_address_out_of_pool()

test_router_add_interface_by_port_cidr_overlapped_with_gateway()

test_router_add_interface_by_port_fails_nested()

test_router_add_interface_by_port_other_tenant_address_in_pool()

test_router_add_interface_by_port_other_tenant_address_out_of_pool()

test_router_add_interface_by_subnet_other_tenant_subnet_returns_400()

test_router_add_interface_cidr_overlapped_with_gateway()

test_router_add_interface_delete_port_after_failure()

test_router_add_interface_dup_port()
This tests that if multiple routers add one port as their interfaces. Only the first router's interface would be added to this port. All the later requests would return exceptions.

test_router_add_interface_dup_subnet1_returns_400()

test_router_add_interface_dup_subnet2_returns_400()

test_router_add_interface_empty_port_and_subnet_ids()

test_router_add_interface_ipv6_port_existing_network_returns_400()
Ensure unique IPv6 router ports per network id.

Adding a router port containing one or more IPv6 subnets with the same network id as an existing router port should fail. This is so there is no ambiguity regarding on which port to add an IPv6 subnet when executing router-interface-add with a subnet and no port.

test_router_add_interface_ipv6_subnet()
Test router-interface-add for valid ipv6 subnets.

Verify the valid use-cases of an IPv6 subnet where we are allowed to associate to the Neutron Router are successful.

test_router_add_interface_ipv6_subnet_without_gateway_ip()

test_router_add_interface_multiple_ipv4_subnet_port_returns_400()
Test adding router port with multiple IPv4 subnets fails.

Multiple IPv4 subnets are not allowed on a single router port. Ensure that adding a port with multiple IPv4 subnets to a router fails.

test_router_add_interface_multiple_ipv4_subnets()
Test router-interface-add for multiple ipv4 subnets.

Verify that adding multiple ipv4 subnets from the same network to a router places them all on different router interfaces.

test_router_add_interface_multiple_ipv6_subnet_port()
A port with multiple IPv6 subnets can be added to a router

Create a port with multiple associated IPv6 subnets and attach it to a router. The action should succeed.

test_router_add_interface_multiple_ipv6_subnets_different_net()
Test router-interface-add for ipv6 subnets on different networks.

Verify that adding multiple ipv6 subnets from different networks to a router places them on different router interfaces.

test_router_add_interface_multiple_ipv6_subnets_same_net()
Test router-interface-add for multiple ipv6 subnets on a network.

Verify that adding multiple ipv6 subnets from the same network to a router places them all on the same router interface.

test_router_add_interface_no_data_returns_400()

test_router_add_interface_overlapped_cidr_returns_400()

test_router_add_interface_port()

test_router_add_interface_port_bad_tenant_returns_404()

test_router_add_interface_port_without_ips()

test_router_add_interface_subnet()

test_router_add_interface_subnet_with_bad_tenant_returns_404()

test_router_add_interface_subnet_with_port_from_other_tenant()

test_router_add_interface_with_both_ids_returns_400()

test_router_clear_gateway_callback_failure_returns_409()

test_router_concurrent_delete_upon_subnet_create()

test_router_create()

test_router_create_call_extensions()

test_router_create_with_gwinfo()

test_router_create_with_gwinfo_ext_ip()

test_router_create_with_gwinfo_ext_ip_non_admin()

test_router_create_with_gwinfo_ext_ip_subnet()

test_router_delete()

test_router_delete_callback()

test_router_delete_denied_for_plugin_managed_router()

test_router_delete_dhcpv6_stateless_subnet_inuse_returns_409()

test_router_delete_ipv6_slaac_subnet_inuse_returns_409()

test_router_delete_race_with_interface_add()

test_router_delete_subnet_inuse_returns_409()

test_router_delete_with_floatingip_existed_returns_409()

test_router_delete_with_port_existed_returns_409()

test_router_interface_add_denied_for_plugin_managed_router()

test_router_interface_in_use_by_route()

test_router_list()

test_router_list_with_pagination()

test_router_list_with_pagination_reverse()

test_router_list_with_parameters()

test_router_list_with_sort()

test_router_remove_interface_callback_failure_returns_409()

test_router_remove_interface_inuse_returns_409()

test_router_remove_interface_nothing_returns_400()

test_router_remove_interface_returns_200()

test_router_remove_interface_with_both_ids_returns_200()

test_router_remove_interface_wrong_port_returns_404()

test_router_remove_interface_wrong_subnet_returns_400()

test_router_remove_ipv6_subnet_from_interface()

Delete a subnet from a router interface

Verify that deleting a subnet with router-interface-delete removes that subnet when there are multiple subnets on the interface and removes the interface when it is the last subnet on the interface.

test_router_show()

test_router_specify_id_backend()

test_router_update()

test_router_update_change_external_gateway_and_routes()

test_router_update_change_name_changes_redundancy_routers()

test_router_update_delete_routes()

test_router_update_denied_for_plugin_managed_router()

test_router_update_gateway()

test_router_update_gateway_add_multiple_prefixes_ipv6()

test_router_update_gateway_to_empty_with_existed_floatingip()

test_router_update_gateway_upon_subnet_create_ipv6()

test_router_update_gateway_upon_subnet_create_max_ips_ipv6()
Create subnet should not cause excess fixed IPs on router gw

If a router gateway port has the maximum of one IPv4 and one IPv6 fixed, create subnet should not add any more IP addresses to the port (unless this is the subnet is a SLAAC/DHCPv6-stateless subnet in which case the addresses are added automatically)

test_router_update_gateway_with_different_external_subnet()

test_router_update_gateway_with_existed_floatingip()

test_router_update_gateway_with_external_ip_used_by_gw()

test_router_update_gateway_with_invalid_external_ip()

test_router_update_gateway_with_invalid_external_subnet()

test_router_update_on_external_port()

test_router_update_with_dup_address()

test_router_update_with_invalid_ip_address()

test_router_update_with_invalid_nexthop_ip()

test_router_update_with_nexthop_is_outside_port_subnet()

test_router_update_with_too_many_routes()

test_routes_update_for_multiple_routers()

test_show_ha_router_non_admin()

test_show_non_gw_ha_router_non_admin()

test_two_fips_one_port_invalid_return_409()

test_update_ha_type_on_non_gw_router_with_ha_enabled_fails()

test_update_ha_type_on_router_with_ha_enabled_fails()

test_update_non_gw_router_ha_settings()

test_update_non_gw_router_ha_settings_non_admin_fails()

test_update_port_device_id_to_different_tenants_router()

test_update_router_ha_settings()

test_update_router_ha_settings_non_admin_fails()

test_update_subnet_gateway_for_external_net()
Test to make sure notification to routers occurs when the gateway ip address of a subnet of the external network is changed.

class networking_cisco.tests.unit.cisco.l3.test_ha_l3_router_appliance_plugin.**HAL3RouterTes**
Bases: object

class networking_cisco.tests.unit.cisco.l3.test_ha_l3_router_appliance_plugin.**L3AgentHARout**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3AgentRouterApplianceTestCase*

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test_floatingips_create_precommit_event()

test_floatingips_op_agent()

test_interfaces_op_agent()

test_l3_agent_routers_query_floatingips()

test_l3_agent_routers_query_gateway()

test_l3_agent_routers_query_ignore_interfaces_with_moreThanOneIp()

test_l3_agent_routers_query_interfaces()

test_l3_agent_sync_interfaces()
Test L3 interfaces query return valid result

test_router_create_event_exception_preserved()

test_router_create_precommit_event()

test_router_delete_event_exception_preserved()

test_router_delete_precommit_event()

test_router_gateway_op_agent()

test_router_update_event_exception_preserved()

test_router_update_precommit_event()

class networking_cisco.tests.unit.cisco.l3.test_ha_l3_router_appliance_plugin.**L3CfgAgentHA**

Bases: *networking_cisco.tests.unit.cisco.l3.test_ha_l3_router_appliance_plugin.HAL3RouterTestsMixin*, *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3CfgAgentRouterApplianceTestCase*

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

tearDown()

test__allocate_hosting_port_returns_none_if_binding_fails()

test_failed_add_gw_hosting_port_info_changes_router_status(*num=1*)

test_failed_add_hosting_port_info_changes_router_status(*num=1*)

test_floatingips_create_precommit_event()

test_floatingips_op_agent()

test_ha_floatingip_update_cfg_agent()

test_ha_floatingips_op_cfg_agent()

test_ha_routes_op_cfg_agent()

test_interfaces_op_agent()

test_l3_agent_routers_query_floatingips()

test_l3_agent_routers_query_gateway()

test_l3_agent_routers_query_ignore_interfaces_with_moreThanOneIp()

test_l3_agent_routers_query_interfaces()

test_l3_agent_sync_interfaces()
Test L3 interfaces query return valid result

test_l3_cfg_agent_query_ha_rdcy_router_routes_include_user_vsbl_router()

test_l3_cfg_agent_query_ha_rdcy_router_routes_is_from_user_vsbl_router()

test_l3_cfg_agent_query_ha_router_with_fips()

test_populate_port_ha_information_all_retries_fail()

test_populate_port_ha_information_no_port()

test_populate_port_ha_information_no_port_gw_port()

test_populate_port_ha_information_red_router_no_port()

test_populate_port_ha_information_red_router_no_port_gw_port()

test_populate_port_ha_information_retries_succeed()

test_router_create_event_exception_preserved()

test_router_create_precommit_event()

test_router_delete_event_exception_preserved()

test_router_delete_precommit_event()

test_router_gateway_op_agent()

test_router_update_event_exception_preserved()

test_router_update_precommit_event()

class networking_cisco.tests.unit.cisco.l3.test_ha_l3_router_appliance_plugin.**TestApplianc**

Bases: *networking_cisco.plugins.cisco.db.l3.ha_db.HA_db_mixin*, *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.TestApplianceL3RouterServicePlugin*

cleanup_after_test()

Reset all class variables to their default values. This is needed to avoid tests to pollute subsequent tests.

supported_extension_aliases = ['router', 'standard-attr-description', 'routerhost', 'r

class networking_cisco.tests.unit.cisco.l3.test_ha_l3_router_appliance_plugin.**TestHAL3Route**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.TestL3RouterApplianceExtensionManager*

get_resources()

networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin module

class networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.**L3AgentRouterApp**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3RouterApplianceTestCaseBase*, neutron.tests.unit.extensions.test_l3.L3AgentDbTestCaseBase

router_type = 'Namespace_Neutron_router'

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test_floatingips_create_precommit_event()

test_floatingips_op_agent()

test_interfaces_op_agent()

test_l3_agent_routers_query_floatingips()

test_l3_agent_routers_query_gateway()

test_l3_agent_routers_query_ignore_interfaces_with_moreThanOneIp()

test_l3_agent_routers_query_interfaces()

test_l3_agent_sync_interfaces()
Test L3 interfaces query return valid result

test_router_create_event_exception_preserved()

test_router_create_precommit_event()

test_router_delete_event_exception_preserved()

test_router_delete_precommit_event()

test_router_gateway_op_agent()

test_router_update_event_exception_preserved()

test_router_update_precommit_event()

class networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.**L3CfgAgentRoute**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3RouterApplianceTestCaseBase*, neutron.tests.unit.extensions.test_l3.L3AgentDbTestCaseBase

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

tearDown()

test__allocate_hosting_port_returns_none_if_binding_fails()

test_failed_add_gw_hosting_port_info_changes_router_status(*num=1*)

test_failed_add_hosting_port_info_changes_router_status(*num=1*)

test_floatingips_create_precommit_event()

test_floatingips_op_agent()

test_interfaces_op_agent()

test_l3_agent_routers_query_floatingips()

test_l3_agent_routers_query_gateway()

test_l3_agent_routers_query_ignore_interfaces_with_moreThanOneIp()

test_l3_agent_routers_query_interfaces()

test_l3_agent_sync_interfaces()
Test L3 interfaces query return valid result

test_router_create_event_exception_preserved()

test_router_create_precommit_event()

test_router_delete_event_exception_preserved()

test_router_delete_precommit_event()

test_router_gateway_op_agent()

test_router_update_event_exception_preserved()

test_router_update_precommit_event()

class networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.**L3RouterApplianc**

Bases: neutron.tests.unit.extensions.test_l3.L3NatTestCaseMixin, *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3RouterApplianceTestCaseBase*

router_type = 'ASR1k_Neutron_router'

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test_create_floatingip_gbp()

test_is_gbp_workflow()

test_update_floatingip_gbp()

class networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.**L3RouterApplianc**

Bases: neutron.tests.unit.extensions.test_l3.L3NatTestCaseBase, neutron.tests.unit.extensions.test_extraroute.ExtraRouteDBTestCaseBase, *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3RouterApplianceTestCaseBase*

router_interface_remove_denied_for_plugin_managed_router()

router_type = 'Namespace_Neutron_router'

test__notify_subnetpool_address_scope_update()

test_associate_to_dhcp_port_fails()

test_create_floating_non_ext_network_returns_400()

test_create_floatingip_invalid_fixed_ip_address_returns_400()

test_create_floatingip_invalid_fixed_ipv6_address_returns_400()

test_create_floatingip_invalid_floating_network_id_returns_400()

test_create_floatingip_invalid_floating_port_id_returns_400()

test_create_floatingip_ipv6_and_ipv4_network_creates_ipv4()

test_create_floatingip_ipv6_only_network_returns_400()

test_create_floatingip_no_ext_gateway_return_404()

test_create_floatingip_no_public_subnet_returns_400()

test_create_floatingip_non_admin_context_agent_notification()

test_create_floatingip_with_assoc(*expected_status='ACTIVE'*)

test_create_floatingip_with_assoc_to_ipv4_and_ipv6_port()

test_create_floatingip_with_assoc_to_ipv6_subnet()

test_create_floatingip_with_duplicated_specific_ip()

test_create_floatingip_with_multisubnet_id()

test_create_floatingip_with_specific_ip()

test_create_floatingip_with_specific_ip_non_admin()

test_create_floatingip_with_specific_ip_out_of_allocation()

test_create_floatingip_with_specific_ip_out_of_subnet()

test_create_floatingip_with_subnet_and_invalid_fip_address()

test_create_floatingip_with_subnet_id_and_fip_address()

test_create_floatingip_with_subnet_id_non_admin()

test_create_floatingip_with_wrong_subnet_id()

test_create_floatingips_native_quotas()

test_create_multiple_floatingips_same_fixed_ip_same_port()
This tests that if multiple API requests arrive to create floating IPs on same external network to same port with one fixed ip, the latter API requests would be blocked at database side.

test_create_non_router_port_device_id_of_other_teants_router_update()

test_create_router_gateway_fails_nested()

test_create_router_gateway_fails_nested_delete_router_failed()

test_create_router_port_with_device_id_of_other_teants_router()

test_create_routers_native_quotas()

test_delete_ext_net_with_disassociated_floating_ips()

test_first_floatingip_associate_notification()

test_floating_ip_direct_port_delete_returns_409()

test_floating_port_status_not_applicable()

test_floatingip_association_on_unowned_router()

test_floatingip_crd_ops()

test_floatingip_create_different_fixed_ip_same_port()
This tests that it is possible to delete a port that has multiple floating ip addresses associated with it (each floating address associated with a unique fixed address).

test_floatingip_delete_router_intf_with_port_id_returns_409()

test_floatingip_delete_router_intf_with_subnet_id_returns_409()

test_floatingip_disassociate_notification()

test_floatingip_list_with_pagination()

test_floatingip_list_with_pagination_reverse()

test_floatingip_list_with_port_id()

test_floatingip_list_with_sort()

test_floatingip_multi_external_one_internal()

test_floatingip_port_delete()

test_floatingip_same_external_and_internal()

test_floatingip_update(*expected_status='ACTIVE'*)

test_floatingip_update_different_fixed_ip_same_port()

test_floatingip_update_different_port_owner_as_admin()

test_floatingip_update_different_router()

test_floatingip_update_invalid_fixed_ip()

test_floatingip_update_same_fixed_ip_same_port()

test_floatingip_update_subnet_gateway_disabled(*expected_status='ACTIVE'*)
Attach a floating IP to an instance

Verify that the floating IP can be associated to a port whose subnet's gateway ip is not connected to the external router, but the router has an ip in that subnet.

test_floatingip_update_to_same_port_id_twice(*expected_status='ACTIVE'*)

test_floatingip_via_router_interface_returns_201()

test_floatingip_via_router_interface_returns_404()

test_floatingip_with_assoc_fails()

test_floatingip_with_invalid_create_port()

test_janitor_clears_orphaned_floatingip_port()

test_janitor_doesnt_delete_if_fixed_in_interim()

test_janitor_updates_port_device_id()

test_network_update_external()

test_network_update_external_failure()

test_nexthop_is_port_ip()

test_no_destination_route()

test_no_nexthop_route()

test_none_destination()

test_none_nexthop()

test_route_clear_routes_with_None()

test_route_update_with_external_route()

test_route_update_with_multi_routes()

test_route_update_with_one_route()

test_route_update_with_route_via_another_tenant_subnet()

test_router_add_and_remove_gateway()

test_router_add_and_remove_gateway_tenant_ctx()

test_router_add_gateway_dup_subnet1_returns_400()

test_router_add_gateway_dup_subnet2_returns_400()

test_router_add_gateway_invalid_network_returns_400()

test_router_add_gateway_multiple_subnets_ipv6()
Ensure external gateway set doesn't add excess IPs on router gw

Setting the gateway of a router to an external network with more than one IPv4 and one IPv6 subnet should only add an address from the first IPv4 subnet, an address from the first IPv6-stateful subnet, and an address from each IPv6-stateless (SLAAC and DHCPv6-stateless) subnet

test_router_add_gateway_net_not_external_returns_400()

test_router_add_gateway_no_subnet()

test_router_add_gateway_no_subnet_forbidden()

test_router_add_gateway_non_existent_network_returns_404()

test_router_add_iface_ipv6_ext_ra_subnet_returns_400()
Test router-interface-add for in-valid ipv6 subnets.

Verify that an appropriate error message is displayed when an IPv6 subnet configured to use an external_router for Router Advertisements (i.e., ipv6_ra_mode is None and ipv6_address_mode is not None) is attempted to associate with a Neutron Router.

test_router_add_interface_bad_values()

test_router_add_interface_by_port_admin_address_out_of_pool()

test_router_add_interface_by_port_cidr_overlapped_with_gateway()

test_router_add_interface_by_port_fails_nested()

test_router_add_interface_by_port_other_tenant_address_in_pool()

test_router_add_interface_by_port_other_tenant_address_out_of_pool()

test_router_add_interface_by_subnet_other_tenant_subnet_returns_400()

test_router_add_interface_cidr_overlapped_with_gateway()

test_router_add_interface_delete_port_after_failure()

test_router_add_interface_dup_port()
This tests that if multiple routers add one port as their interfaces. Only the first router's interface would be added to this port. All the later requests would return exceptions.

test_router_add_interface_dup_subnet1_returns_400()

test_router_add_interface_dup_subnet2_returns_400()

test_router_add_interface_empty_port_and_subnet_ids()

test_router_add_interface_ipv6_port_existing_network_returns_400()
Ensure unique IPv6 router ports per network id.

Adding a router port containing one or more IPv6 subnets with the same network id as an existing router port should fail. This is so there is no ambiguity regarding on which port to add an IPv6 subnet when executing router-interface-add with a subnet and no port.

test_router_add_interface_ipv6_subnet()
Test router-interface-add for valid ipv6 subnets.

Verify the valid use-cases of an IPv6 subnet where we are allowed to associate to the Neutron Router are successful.

test_router_add_interface_ipv6_subnet_without_gateway_ip()

test_router_add_interface_multiple_ipv4_subnet_port_returns_400()
Test adding router port with multiple IPv4 subnets fails.

Multiple IPv4 subnets are not allowed on a single router port. Ensure that adding a port with multiple IPv4 subnets to a router fails.

test_router_add_interface_multiple_ipv4_subnets()
Test router-interface-add for multiple ipv4 subnets.

Verify that adding multiple ipv4 subnets from the same network to a router places them all on different router interfaces.

test_router_add_interface_multiple_ipv6_subnet_port()
A port with multiple IPv6 subnets can be added to a router

Create a port with multiple associated IPv6 subnets and attach it to a router. The action should succeed.

test_router_add_interface_multiple_ipv6_subnets_different_net()
Test router-interface-add for ipv6 subnets on different networks.

Verify that adding multiple ipv6 subnets from different networks to a router places them on different router interfaces.

test_router_add_interface_multiple_ipv6_subnets_same_net()
Test router-interface-add for multiple ipv6 subnets on a network.

Verify that adding multiple ipv6 subnets from the same network to a router places them all on the same router interface.

test_router_add_interface_no_data_returns_400()

test_router_add_interface_overlapped_cidr_returns_400()

test_router_add_interface_port()

test_router_add_interface_port_bad_tenant_returns_404()

test_router_add_interface_port_without_ips()

test_router_add_interface_subnet()

test_router_add_interface_subnet_with_bad_tenant_returns_404()

test_router_add_interface_subnet_with_port_from_other_tenant()

test_router_add_interface_with_both_ids_returns_400()

test_router_clear_gateway_callback_failure_returns_409()

test_router_concurrent_delete_upon_subnet_create()

test_router_create()

test_router_create_call_extensions()

test_router_create_with_gwinfo()

test_router_create_with_gwinfo_ext_ip()

test_router_create_with_gwinfo_ext_ip_non_admin()

test_router_create_with_gwinfo_ext_ip_subnet()

test_router_delete()

test_router_delete_callback()

test_router_delete_denied_for_plugin_managed_router()

test_router_delete_dhcpv6_stateless_subnet_inuse_returns_409()

test_router_delete_ipv6_slaac_subnet_inuse_returns_409()

test_router_delete_race_with_interface_add()

test_router_delete_subnet_inuse_returns_409()

test_router_delete_with_floatingip_existed_returns_409()

test_router_delete_with_port_existed_returns_409()

test_router_interface_add_denied_for_plugin_managed_router()

test_router_interface_in_use_by_route()

test_router_list()

test_router_list_with_pagination()

test_router_list_with_pagination_reverse()

test_router_list_with_parameters()

test_router_list_with_sort()

test_router_remove_interface_callback_failure_returns_409()

test_router_remove_interface_inuse_returns_409()

test_router_remove_interface_nothing_returns_400()

test_router_remove_interface_returns_200()

test_router_remove_interface_with_both_ids_returns_200()

test_router_remove_interface_wrong_port_returns_404()

test_router_remove_interface_wrong_subnet_returns_400()

test_router_remove_ipv6_subnet_from_interface()
Delete a subnet from a router interface

Verify that deleting a subnet with router-interface-delete removes that subnet when there are multiple subnets on the interface and removes the interface when it is the last subnet on the interface.

test_router_show()

test_router_specify_id_backend()

test_router_update()

test_router_update_delete_routes()

test_router_update_denied_for_plugin_managed_router()

test_router_update_gateway()

test_router_update_gateway_add_multiple_prefixes_ipv6()

test_router_update_gateway_to_empty_with_existed_floatingip()

test_router_update_gateway_upon_subnet_create_ipv6()

test_router_update_gateway_upon_subnet_create_max_ips_ipv6()
Create subnet should not cause excess fixed IPs on router gw

If a router gateway port has the maximum of one IPv4 and one IPv6 fixed, create subnet should not add any more IP addresses to the port (unless this is the subnet is a SLAAC/DHCPv6-stateless subnet in which case the addresses are added automatically)

test_router_update_gateway_with_different_external_subnet()

test_router_update_gateway_with_existed_floatingip()

test_router_update_gateway_with_external_ip_used_by_gw()

test_router_update_gateway_with_invalid_external_ip()

test_router_update_gateway_with_invalid_external_subnet()

test_router_update_on_external_port()

test_router_update_with_dup_address()

test_router_update_with_invalid_ip_address()

test_router_update_with_invalid_nexthop_ip()

test_router_update_with_nexthop_is_outside_port_subnet()

test_router_update_with_too_many_routes()

test_routes_update_for_multiple_routers()

test_two_fips_one_port_invalid_return_409()

test_update_port_device_id_to_different_tenants_router()

test_update_subnet_gateway_for_external_net()
Test to make sure notification to routers occurs when the gateway ip address of a subnet of the external network is changed.

class networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.**L3RouterApplianc**

Bases: neutron.tests.unit.extensions.test_l3.L3NatTestCaseMixin, *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3RouterApplianceTestCaseBase*

router_type = 'ASR1k_Neutron_router'

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test_create_floatingip_gbp()

test_is_not_gbp_workflow()

test_update_floatingip_no_gbp()

class networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.**L3RouterApplianc**

Bases: neutron.tests.unit.extensions.test_l3.L3NatTestCaseMixin, *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3RouterApplianceTestCaseBase*

routertype = 'ASR1k_router'

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test_add_router_interface_dns()

test_add_router_interface_pre_and_post_port()

test_add_router_interface_pre_and_post_subnet()

test_create_floating_ip_post()

test_create_floating_ip_post_dns()

test_create_router_pre_and_post()

test_delete_floating_ip_pre_and_post()

test_delete_router_pre_and_post()

test_remove_router_interface_pre_and_post_port()

test_remove_router_interface_pre_and_post_subnet()

test_schedule_router_pre_and_post_commit()

test_unschedule_router_pre_and_post_commit()

test_update_floating_ip_pre_and_post()

test_update_router_pre_and_post()

class networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.**L3RouterApplianc**

Bases: neutron.tests.unit.db.test_db_base_plugin_v2.NeutronDbPluginV2TestCase, *networking_cisco.tests.unit.cisco.l3.test_db_routertype.RoutertypeTestCaseMixin*, *networking_cisco.tests.unit.cisco.device_manager.test_db_device_manager.DeviceManagerTestCaseMixin*, *networking_cisco.tests.unit.cisco.l3.l3_router_test_support.L3RouterTestSupportMixin*, *networking_cisco.tests.unit.cisco.device_manager.device_manager_test_support.DeviceManagerTestSupportMixin*

configure_routertypes = True

mock_cfg_agent_notifiers = True

resource_prefix_map = {'hosting_device_templates': '/dev_mgr', 'hosting_devices': '/

restore_attribute_map()

router_type = None

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*, *create_mgmt_nw=True*, *service_plugins=None*)

tearDown()

class networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.**L3RouterApplianc**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3RouterApplianceNamespaceTestCase*

router_type = 'VM_router'

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test__notify_subnetpool_address_scope_update()

test_associate_to_dhcp_port_fails()

test_create_floating_non_ext_network_returns_400()

test_create_floatingip_invalid_fixed_ip_address_returns_400()

test_create_floatingip_invalid_fixed_ipv6_address_returns_400()

test_create_floatingip_invalid_floating_network_id_returns_400()

test_create_floatingip_invalid_floating_port_id_returns_400()

test_create_floatingip_ipv6_and_ipv4_network_creates_ipv4()

test_create_floatingip_ipv6_only_network_returns_400()

test_create_floatingip_no_ext_gateway_return_404()

test_create_floatingip_no_public_subnet_returns_400()

test_create_floatingip_non_admin_context_agent_notification()

test_create_floatingip_with_assoc(*expected_status='ACTIVE'*)

test_create_floatingip_with_assoc_to_ipv4_and_ipv6_port()

test_create_floatingip_with_assoc_to_ipv6_subnet()

test_create_floatingip_with_duplicated_specific_ip()

test_create_floatingip_with_multisubnet_id()

test_create_floatingip_with_specific_ip()

test_create_floatingip_with_specific_ip_non_admin()

test_create_floatingip_with_specific_ip_out_of_allocation()

test_create_floatingip_with_specific_ip_out_of_subnet()

test_create_floatingip_with_subnet_and_invalid_fip_address()

test_create_floatingip_with_subnet_id_and_fip_address()

test_create_floatingip_with_subnet_id_non_admin()

test_create_floatingip_with_wrong_subnet_id()

test_create_floatingips_native_quotas()

test_create_multiple_floatingips_same_fixed_ip_same_port()
This tests that if multiple API requests arrive to create floating IPs on same external network to same port with one fixed ip, the latter API requests would be blocked at database side.

test_create_non_router_port_device_id_of_other_teants_router_update()

test_create_router_gateway_fails_nested()

test_create_router_gateway_fails_nested_delete_router_failed()

test_create_router_port_with_device_id_of_other_teants_router()

test_create_routers_native_quotas()

test_delete_ext_net_with_disassociated_floating_ips()

test_first_floatingip_associate_notification()

test_floating_ip_direct_port_delete_returns_409()

test_floating_port_status_not_applicable()

test_floatingip_association_on_unowned_router()

test_floatingip_crd_ops()

test_floatingip_create_different_fixed_ip_same_port()
This tests that it is possible to delete a port that has multiple floating ip addresses associated with it (each floating address associated with a unique fixed address).

test_floatingip_delete_router_intf_with_port_id_returns_409()

test_floatingip_delete_router_intf_with_subnet_id_returns_409()

test_floatingip_disassociate_notification()

test_floatingip_list_with_pagination()

test_floatingip_list_with_pagination_reverse()

test_floatingip_list_with_port_id()

test_floatingip_list_with_sort()

test_floatingip_multi_external_one_internal()

test_floatingip_port_delete()

test_floatingip_same_external_and_internal()

test_floatingip_update(*expected_status='ACTIVE'*)

test_floatingip_update_different_fixed_ip_same_port()

test_floatingip_update_different_port_owner_as_admin()

test_floatingip_update_different_router()

test_floatingip_update_invalid_fixed_ip()

test_floatingip_update_same_fixed_ip_same_port()

test_floatingip_update_subnet_gateway_disabled(*expected_status='ACTIVE'*)
Attach a floating IP to an instance

Verify that the floating IP can be associated to a port whose subnet's gateway ip is not connected to the external router, but the router has an ip in that subnet.

test_floatingip_update_to_same_port_id_twice(*expected_status='ACTIVE'*)

test_floatingip_via_router_interface_returns_201()

test_floatingip_via_router_interface_returns_404()

test_floatingip_with_assoc_fails()

test_floatingip_with_invalid_create_port()

test_janitor_clears_orphaned_floatingip_port()

test_janitor_doesnt_delete_if_fixed_in_interim()

test_janitor_updates_port_device_id()

test_network_update_external()

test_network_update_external_failure()

test_nexthop_is_port_ip()

test_no_destination_route()

test_no_nexthop_route()

test_none_destination()

test_none_nexthop()

test_route_clear_routes_with_None()

test_route_update_with_external_route()

test_route_update_with_multi_routes()

test_route_update_with_one_route()

test_route_update_with_route_via_another_tenant_subnet()

test_router_add_and_remove_gateway()

test_router_add_and_remove_gateway_tenant_ctx()

test_router_add_gateway_dup_subnet1_returns_400()

test_router_add_gateway_dup_subnet2_returns_400()

test_router_add_gateway_invalid_network_returns_400()

test_router_add_gateway_multiple_subnets_ipv6()
Ensure external gateway set doesn't add excess IPs on router gw

Setting the gateway of a router to an external network with more than one IPv4 and one IPv6 subnet should only add an address from the first IPv4 subnet, an address from the first IPv6-stateful subnet, and an address from each IPv6-stateless (SLAAC and DHCPv6-stateless) subnet

test_router_add_gateway_net_not_external_returns_400()

test_router_add_gateway_no_subnet()

test_router_add_gateway_no_subnet_forbidden()

test_router_add_gateway_non_existent_network_returns_404()

test_router_add_iface_ipv6_ext_ra_subnet_returns_400()
Test router-interface-add for in-valid ipv6 subnets.

Verify that an appropriate error message is displayed when an IPv6 subnet configured to use an external_router for Router Advertisements (i.e., ipv6_ra_mode is None and ipv6_address_mode is not None) is attempted to associate with a Neutron Router.

test_router_add_interface_bad_values()

test_router_add_interface_by_port_admin_address_out_of_pool()

test_router_add_interface_by_port_cidr_overlapped_with_gateway()

test_router_add_interface_by_port_fails_nested()

test_router_add_interface_by_port_other_tenant_address_in_pool()

test_router_add_interface_by_port_other_tenant_address_out_of_pool()

test_router_add_interface_by_subnet_other_tenant_subnet_returns_400()

test_router_add_interface_cidr_overlapped_with_gateway()

test_router_add_interface_delete_port_after_failure()

test_router_add_interface_dup_port()
This tests that if multiple routers add one port as their interfaces. Only the first router's interface would be added to this port. All the later requests would return exceptions.

test_router_add_interface_dup_subnet1_returns_400()

test_router_add_interface_dup_subnet2_returns_400()

test_router_add_interface_empty_port_and_subnet_ids()

test_router_add_interface_ipv6_port_existing_network_returns_400()
Ensure unique IPv6 router ports per network id.

Adding a router port containing one or more IPv6 subnets with the same network id as an existing router port should fail. This is so there is no ambiguity regarding on which port to add an IPv6 subnet when executing router-interface-add with a subnet and no port.

test_router_add_interface_ipv6_subnet()
Test router-interface-add for valid ipv6 subnets.

Verify the valid use-cases of an IPv6 subnet where we are allowed to associate to the Neutron Router are successful.

test_router_add_interface_ipv6_subnet_without_gateway_ip()

test_router_add_interface_multiple_ipv4_subnet_port_returns_400()
Test adding router port with multiple IPv4 subnets fails.

Multiple IPv4 subnets are not allowed on a single router port. Ensure that adding a port with multiple IPv4 subnets to a router fails.

test_router_add_interface_multiple_ipv4_subnets()
Test router-interface-add for multiple ipv4 subnets.

Verify that adding multiple ipv4 subnets from the same network to a router places them all on different router interfaces.

test_router_add_interface_multiple_ipv6_subnet_port()
A port with multiple IPv6 subnets can be added to a router

Create a port with multiple associated IPv6 subnets and attach it to a router. The action should succeed.

test_router_add_interface_multiple_ipv6_subnets_different_net()
Test router-interface-add for ipv6 subnets on different networks.

Verify that adding multiple ipv6 subnets from different networks to a router places them on different router interfaces.

test_router_add_interface_multiple_ipv6_subnets_same_net()
Test router-interface-add for multiple ipv6 subnets on a network.

Verify that adding multiple ipv6 subnets from the same network to a router places them all on the same router interface.

test_router_add_interface_no_data_returns_400()

test_router_add_interface_overlapped_cidr_returns_400()

test_router_add_interface_port()

test_router_add_interface_port_bad_tenant_returns_404()

test_router_add_interface_port_without_ips()

test_router_add_interface_subnet()

test_router_add_interface_subnet_with_bad_tenant_returns_404()

test_router_add_interface_subnet_with_port_from_other_tenant()

test_router_add_interface_with_both_ids_returns_400()

test_router_clear_gateway_callback_failure_returns_409()

test_router_concurrent_delete_upon_subnet_create()

test_router_create()

test_router_create_call_extensions()

test_router_create_with_gwinfo()

test_router_create_with_gwinfo_ext_ip()

test_router_create_with_gwinfo_ext_ip_non_admin()

test_router_create_with_gwinfo_ext_ip_subnet()

test_router_delete()

test_router_delete_callback()

test_router_delete_denied_for_plugin_managed_router()

test_router_delete_dhcpv6_stateless_subnet_inuse_returns_409()

test_router_delete_ipv6_slaac_subnet_inuse_returns_409()

test_router_delete_race_with_interface_add()

test_router_delete_subnet_inuse_returns_409()

test_router_delete_with_floatingip_existed_returns_409()

test_router_delete_with_port_existed_returns_409()

test_router_interface_add_denied_for_plugin_managed_router()

test_router_interface_in_use_by_route()

test_router_list()

test_router_list_with_pagination()

test_router_list_with_pagination_reverse()

test_router_list_with_parameters()

test_router_list_with_sort()

test_router_remove_interface_callback_failure_returns_409()

test_router_remove_interface_inuse_returns_409()

test_router_remove_interface_nothing_returns_400()

test_router_remove_interface_returns_200()

test_router_remove_interface_with_both_ids_returns_200()

test_router_remove_interface_wrong_port_returns_404()

test_router_remove_interface_wrong_subnet_returns_400()

test_router_remove_ipv6_subnet_from_interface()
Delete a subnet from a router interface

Verify that deleting a subnet with router-interface-delete removes that subnet when there are multiple subnets on the interface and removes the interface when it is the last subnet on the interface.

test_router_show()

test_router_specify_id_backend()

test_router_update()

test_router_update_delete_routes()

test_router_update_denied_for_plugin_managed_router()

test_router_update_gateway()

test_router_update_gateway_add_multiple_prefixes_ipv6()

test_router_update_gateway_to_empty_with_existed_floatingip()

test_router_update_gateway_upon_subnet_create_ipv6()

test_router_update_gateway_upon_subnet_create_max_ips_ipv6()
Create subnet should not cause excess fixed IPs on router gw

If a router gateway port has the maximum of one IPv4 and one IPv6 fixed, create subnet should not add any more IP addresses to the port (unless this is the subnet is a SLAAC/DHCPv6-stateless subnet in which case the addresses are added automatically)

test_router_update_gateway_with_different_external_subnet()

test_router_update_gateway_with_existed_floatingip()

test_router_update_gateway_with_external_ip_used_by_gw()

test_router_update_gateway_with_invalid_external_ip()

test_router_update_gateway_with_invalid_external_subnet()

test_router_update_on_external_port()

test_router_update_with_dup_address()

test_router_update_with_invalid_ip_address()

test_router_update_with_invalid_nexthop_ip()

test_router_update_with_nexthop_is_outside_port_subnet()

test_router_update_with_too_many_routes()

test_routes_update_for_multiple_routers()

test_two_fips_one_port_invalid_return_409()

test_update_port_device_id_to_different_tenants_router()

test_update_subnet_gateway_for_external_net()
Test to make sure notification to routers occurs when the gateway ip address of a subnet of the external network is changed.

class networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.**TestApplianceL3**
Bases: neutron.db.dns_db.DNSDbMixin, *networking_cisco.tests.unit.cisco.l3.l3_router_test_support.TestL3RouterServicePlugin*

cleanup_after_test()
Reset all class variables to their default values. This is needed to avoid tests to pollute subsequent tests.

supported_extension_aliases = ['router', 'standard-attr-description', 'routerhost', 'r

class networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.**TestL3RouterAppl**
Bases: *networking_cisco.tests.unit.cisco.l3.test_db_routertype.L3TestRoutertypeExtensionManager*

get_resources()

class networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.**TestNoL3NatPlugi**
Bases: neutron.tests.unit.extensions.test_l3.TestNoL3NatPlugin, neutron.db.agents_db.AgentDbMixin

NET_TYPE = 'vlan'

get_network_profiles(*context*, *filters=None*, *fields=None*)

get_policy_profiles(*context*, *filters=None*, *fields=None*)

supported_extension_aliases = ['external-net', 'provider']

networking_cisco.tests.unit.cisco.l3.test_l3_router_callbacks module

class networking_cisco.tests.unit.cisco.l3.test_l3_router_callbacks.**TestCfgAgentL3RouterCa**

Bases: neutron.tests.base.BaseTestCase

setUp()

test_cfg_sync_all_hosted_routers_missing_scheduling_fcn()

test_cfg_sync_all_hosted_routers_retries_on_db_errors()

test_cfg_sync_routers_missing_scheduling_fcn()

test_cfg_sync_routers_retries_on_db_errors()

test_update_floatingip_statuses_cfg_ignores_missing_fip()

test_update_floatingip_statuses_cfg_retries_on_db_errors()

test_update_floatingip_statuses_cfg_sets_status_down_if_no_router()

test_update_port_statuses_cfg_retries_on_db_errors()

networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers module

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**HostingDevic**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.L3RoutertypeAwareHostingDeviceSchedulerTestCaseBase*

mock_cfg_agent_notifiers = False

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_backlogged_routers_scheduled_routers_updated_notification()

test_router_add_to_hosting_device_notification()

test_router_remove_from_hosting_device_notification()

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**L3RouterHost**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.L3RoutertypeAwareHostingDeviceSchedulerTestCaseBase*

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_get_candidates_excludes_admin_down()

test_get_candidates_excludes_non_active()

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**L3RouterHost**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.L3RoutertypeAwareHostingDeviceSchedulerTestCaseBase*

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_ha_routers_hosted_on_different_hosting_devices()

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**L3Routertype**

Bases: neutron.tests.unit.scheduler.test_l3_agent_scheduler.L3AgentChanceSchedulerTestCase, *networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.L3RoutertypeAwareL3AgentSchedulerTestCase*

setUp()

test__schedule_router_skips_unschedulable_routers()

test__unbind_router_removes_binding()

test_add_distributed_router_to_l3_agent()

test_add_distributed_router_to_l3_agent_already_scheduled()

test_add_router_to_l3_agent()

test_add_router_to_l3_agent_already_scheduled()

test_add_router_to_l3_agent_dvr_to_snat()

test_add_router_to_l3_agent_mismatch_error_dvr_to_dvr()

test_add_router_to_l3_agent_mismatch_error_dvr_to_legacy()

test_add_router_to_l3_agent_mismatch_error_legacy_to_dvr()

test_bind_absent_router()

test_bind_existing_router()

test_bind_new_router()

test_check_ports_exist_on_l3agent_with_dhcp_enabled_subnets()

test_get_l3_agent_candidates_centralized()

test_get_l3_agent_candidates_dvr()

test_get_l3_agent_candidates_dvr_ha_snat_no_vms()

test_get_l3_agent_candidates_dvr_no_vms()

test_get_l3_agent_candidates_dvr_snat()

test_get_l3_agent_candidates_dvr_snat_no_vms()

test_get_l3_agent_candidates_legacy()

test_get_l3_agents_hosting_routers()

test_get_unscheduled_routers_only_returns_namespace_routers()

test_only_namespace_routers_scheduled_by_l3agent_scheduler()

test_random_scheduling()

test_remove_router_from_l3_agent_in_dvr_mode()

test_remove_router_from_l3_agent_in_dvr_snat_mode()

test_rpc_sync_routers_gets_only_namespace_routers()

test_schedule_dvr_router_without_snatbinding_and_no_gw()

test_schedule_router_distributed()

test_scheduler_auto_schedule_when_agent_added()

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**L3Routertype**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.L3RoutertypeAwareHostingDeviceSchedulerTestCaseBase*

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_already_backlogged_router_not_backlogged()

test_backlogged_router_is_scheduled_if_hosting_device_exists()

test_namespace_router_not_backlogged()

test_new_router_backlogged_and_remains_backlogged_if_no_hosting_device()

test_router_deleted_by_other_process_removed_from_backlog()

test_router_scheduling_aborts_if_other_process_scheduled_router()

test_router_without_auto_schedule_not_backlogged()

test_rpc_sync_routers_ext_gets_no_namespace_routers()

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**L3Routertype**

Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.L3RoutertypeAwareHostingDeviceSchedulerTestCaseBase*

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

test_hosted_router_add_to_different_type_hosting_device()

test_hosted_router_add_to_hosting_device()

test_hosting_device_keep_services_off()

test_hosting_device_keep_services_on()

test_list_active_sync_all_routers_on_all_hosting_devices()

test_list_active_sync_all_routers_on_some_hosting_devices()

test_list_active_sync_routers_on_hosting_devices_cfg_agent_admin_down()

test_list_active_sync_routers_on_hosting_devices_idle_cfg_agent()

test_list_active_sync_routers_on_hosting_devices_no_cfg_agent_on_host()

test_list_active_sync_some_routers_on_all_hosting_devices()

test_list_active_sync_some_routers_on_some_hosting_devices()

test_list_all_routers_on_hosting_devices()

test_list_hosting_devices_hosting_non_existent_router()

test_list_hosting_devices_hosting_router()

test_list_hosting_devices_hosting_unhosted_router()

test_list_routers_by_hosting_device()

test_list_routers_by_hosting_device_with_non_existing_hosting_device()

test_router_add_to_hosting_device()

test_router_add_to_hosting_device_insufficient_slots()

test_router_add_to_hosting_device_insufficient_slots_no_auto()

test_router_add_to_hosting_device_two_times()

test_router_add_to_hosting_device_with_admin_state_down()

test_router_remove_from_hosting_device()

test_router_remove_from_wrong_hosting_device()

test_router_reschedule_from_dead_hosting_device()

test_router_scheduling_policy()

test_router_without_auto_schedule_not_unscheduled_from_dead_hd()

test_unhosted_router_remove_from_hosting_device()

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**L3Routertype**

Bases: neutron.tests.unit.extensions.test_l3.L3NatTestCaseMixin, *networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.RouterHostingDeviceSchedulerTestMixIn*, *networking_cisco.tests.unit.cisco.l3.test_l3_router_appliance_plugin.L3RouterApplianceTestCaseBase*

configure_routertypes = False

router_type = 'ASR1k_Neutron_router'

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*, *use_ini_files=True*)

tearDown()

test_agent_registration_bad_timestamp()

test_agent_registration_invalid_timestamp_allowed()

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**L3Routertype**

Bases: neutron.tests.unit.scheduler.test_l3_agent_scheduler.L3SchedulerTestCaseMixin, *networking_cisco.tests.unit.cisco.l3.test_db_routertype.RoutertypeTestCaseMixin*, *networking_cisco.tests.unit.cisco.device_manager.test_db_device_manager.DeviceManagerTestCaseMixin*, *networking_cisco.tests.unit.cisco.l3.l3_router_test_support.L3RouterTestSupportMixin*, *networking_cisco.tests.unit.cisco.device_manager.device_manager_test_support.DeviceManagerTestSupportMixin*

resource_prefix_map = {'hosting_device_templates': '/dev_mgr', 'hosting_devices': '/

setUp(*core_plugin=None*, *l3_plugin=None*, *dm_plugin=None*, *ext_mgr=None*)

tearDown()

test_add_router_to_l3_agent_dvr_to_snat()

test_check_ports_exist_on_l3agent_with_dhcp_enabled_subnets()

test_get_unscheduled_routers_only_returns_namespace_routers()

test_only_namespace_routers_scheduled_by_l3agent_scheduler()

test_rpc_sync_routers_gets_only_namespace_routers()

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**L3Routertype**

Bases: neutron.tests.unit.scheduler.test_l3_agent_scheduler.L3AgentLeastRoutersSchedulerTestCase, *networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.L3RoutertypeAwareL3AgentSchedulerTestCase*

setUp()

test__schedule_router_skips_unschedulable_routers()

test__unbind_router_removes_binding()

test_add_distributed_router_to_l3_agent()

test_add_distributed_router_to_l3_agent_already_scheduled()

test_add_router_to_l3_agent()

test_add_router_to_l3_agent_already_scheduled()

test_add_router_to_l3_agent_dvr_to_snat()

test_add_router_to_l3_agent_mismatch_error_dvr_to_dvr()

test_add_router_to_l3_agent_mismatch_error_dvr_to_legacy()

test_add_router_to_l3_agent_mismatch_error_legacy_to_dvr()

test_bind_absent_router()

test_bind_existing_router()

test_bind_new_router()

test_check_ports_exist_on_l3agent_with_dhcp_enabled_subnets()

test_get_l3_agent_candidates_centralized()

test_get_l3_agent_candidates_dvr()

test_get_l3_agent_candidates_dvr_ha_snat_no_vms()

test_get_l3_agent_candidates_dvr_no_vms()

test_get_l3_agent_candidates_dvr_snat()

test_get_l3_agent_candidates_dvr_snat_no_vms()

test_get_l3_agent_candidates_legacy()

test_get_l3_agents_hosting_routers()

test_get_unscheduled_routers_only_returns_namespace_routers()

test_only_namespace_routers_scheduled_by_l3agent_scheduler()

test_remove_router_from_l3_agent_in_dvr_mode()

test_remove_router_from_l3_agent_in_dvr_snat_mode()

test_rpc_sync_routers_gets_only_namespace_routers()

test_schedule_dvr_router_without_snatbinding_and_no_gw()

test_schedule_router_distributed()

test_scheduler()

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**RouterHostin**
Bases: neutron.tests.unit.db.test_agentschedulers_db.AgentSchedulerTestMixIn

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**TestHASchedu**
Bases: *networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.TestSchedulingL3RouterApplianceExtensionManager*

get_resources()

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**TestSchedul**
Bases: *networking_cisco.tests.unit.cisco.l3.l3_router_test_support.TestL3RouterServicePlugin*, *networking_cisco.plugins.cisco.db.scheduler.l3_routertype_aware_schedulers_db.L3RouterTypeAwareSchedulerDbMixin*

cleanup_after_test()
Reset all class variables to their default values. This is needed to avoid tests to pollute subsequent tests.

supported_extension_aliases = ['router', 'standard-attr-description', 'routerhost', 'r

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**TestSchedul**
Bases: *networking_cisco.plugins.cisco.db.l3.ha_db.HA_db_mixin*, *networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.TestSchedulingCapableL3RouterServicePlugin*

cleanup_after_test()
Reset all class variables to their default values. This is needed to avoid tests to pollute subsequent tests.

supported_extension_aliases = ['router', 'standard-attr-description', 'routerhost', 'r

class networking_cisco.tests.unit.cisco.l3.test_l3_routertype_aware_schedulers.**TestSchedul**
Bases: *networking_cisco.tests.unit.cisco.l3.test_db_routertype.L3TestRoutertypeExtensionManager*

get_resources()

Module contents

Module contents

networking_cisco.tests.unit.db package

Submodules

networking_cisco.tests.unit.db.test_migrations module

class networking_cisco.tests.unit.db.test_migrations.**TestModelsMigrationsMysql**(**args*, ***kwds*)
Bases: networking_cisco.tests.unit.db.test_migrations._TestModelsMigrationsCisco, *networking_cisco.tests.test_compatibility.MySQLTestCase*

test_branches()

test_forbid_offline_migrations_starting_newton()

test_has_offline_migrations_all_heads_upgraded()

test_has_offline_migrations_pending_contract_scripts()

test_models_sync()

test_upgrade_contract_branch()

test_upgrade_expand_branch()

networking_cisco.tests.unit.db.test_model_base module

class networking_cisco.tests.unit.db.test_model_base.**TestModelBase**(**args*, ***kwds*)
Bases: neutron.tests.unit.testlib_api.SqlTestCase

setUp()

test_get_set_tenant_id_project()

test_get_set_tenant_id_tenant()

test_model_base()

test_project_id_attribute()

test_tenant_id_attribute()

class networking_cisco.tests.unit.db.test_model_base.**TestTable**(***kwargs*)

Bases: sqlalchemy.ext.declarative.api.Base, neutron_lib.db.model_base.HasProject, neutron_lib.db.model_base.HasId, neutron_lib.db.model_base.HasStatusDescription

id

name

project_id

status

status_description

tenant_id

Module contents

networking_cisco.tests.unit.ml2_drivers package

Subpackages

networking_cisco.tests.unit.ml2_drivers.nexus package

Submodules

networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_config module

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_config.**TestCiscoNexusPlugin**

Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_config.TestCiscoNexusPluginConfigBase*

test_config_using_subsection_option()

test_create_device_error()

Test error during create of the Nexus device dictionary.

test_dict_host_port_mapping()

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_config.**TestCiscoNexusPlugin**

Bases: neutron.tests.unit.testlib_api.SqlTestCase

setUp()

networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base module

Basic test Class and elements for testing Cisco Nexus platforms.

Most classes in this file do not contain test cases but instead provide common methods for other classes to utilize. This class provides the basic methods needed to drive a create or delete port request thru to the restapi driver. It verifies the final content of the data base and verifies what data the Drivers sent out. There also exists another 'base' class specifically for Replay testing.

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.**FakeNetworkContex**

Bases: object

Network context for testing purposes only.

current

network_segments

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.**FakePortContext**(

Bases: object

Port context for testing purposes only.

allocate_dynamic_segment(*segment*)

bottom_bound_segment

continue_binding(*segment_id*, *next_segments_to_bind*)

current

network

original

original_bottom_bound_segment

original_top_bound_segment

set_binding(*segment_id*, *vif_type*, *vif_details*, *status=None*)

set_orig_port(*device_id*, *host_name*, *device_owner*, *profile=None*, *vnic_type='normal'*, *dns_name=None*, *netid=None*)

top_bound_segment

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.**FakeUnbindPortCo**

Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.FakePortContext*

Port context used during migration to unbind port.

bottom_bound_segment

original_bottom_bound_segment

original_top_bound_segment

top_bound_segment

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.**TestCiscoNexusBa**

Bases: neutron.tests.unit.testlib_api.SqlTestCase

Feature Base Test Class for Cisco ML2 Nexus driver.

class TestConfigObj(*nexus_ip_addr*, *host_name*, *nexus_port*, *instance_id*, *vlan_id*, *vxlan_id*, *mcast_group*, *device_owner*, *profile*, *dns_name*, *vnic_type*)

Bases: tuple

device_owner
Alias for field number 7

dns_name
Alias for field number 9

host_name
Alias for field number 1

instance_id
Alias for field number 3

mcast_group
Alias for field number 6

nexus_ip_addr
Alias for field number 0

nexus_port
Alias for field number 2

profile
Alias for field number 8

vlan_id
Alias for field number 4

vnic_type
Alias for field number 10

vxlan_id
Alias for field number 5

get_init_side_effect(*action*, *ipaddr=None*, *body=None*, *headers=None*)

get_side_effect(*action*, *ipaddr=None*, *body=None*, *headers=None*)

restapi_mock_init()

setUp()
Sets up mock client, switch, and credentials dictionaries.

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.**TestCiscoNexusBas**
Bases: object

Unit tests driver results for Cisco ML2 Nexus.

get_test_results(*name*)

test_results = {}

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.**TestCiscoNexusPlu**

Bases: neutron.tests.unit.testlib_api.SqlTestCase

setUp()

test__initialize_host_port_mappings()
Verify port-mapping table is configured correctly.

test__initialize_host_port_mappings_with_dict()
Verify port-mapping table is configured correctly.

test_initialize_calls_init_host_mappings(*mock_init_host*)

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.**TestCiscoNexusRep**

Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBase*

Replay Base Test Class for Cisco ML2 Nexus driver.

setUp()
Sets up mock driver, and switch and credentials dictionaries.

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.**TestContext**(**args*, ***kwd*

Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBase*

Verify Context Blocks for Cisco ML2 Nexus driver.

baremetal_profile = {'local_link_information': [{'port_id': 'port-channel:1', 'switch

test_baremetal_format()

test_configs = OrderedDict([('test_bm_vlan_unique1', TestConfigObj(nexus_ip_addr='1.1.

test_normal_vlan_format()

test_normal_vxlan_format()

networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_db module

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_db.**TestCiscoNexusDb**(*a
**

Bases: neutron.tests.unit.testlib_api.SqlTestCase

Unit tests for Cisco mechanism driver's Nexus port binding database.

class NpbObj(*port*, *vlan*, *vni*, *switch*, *instance*, *channel_group*, *is_native_vlan*)

Bases: tuple

channel_group
Alias for field number 5

instance
Alias for field number 4

is_native_vlan
Alias for field number 6

port
Alias for field number 0

switch
Alias for field number 3

vlan
Alias for field number 1

vni
Alias for field number 2

test_nexusbinding_update()
Tests update of vlan IDs for port bindings.

test_nexusportbinding_add_remove()
Tests add and removal of port bindings from the Nexus database.

test_nexusportbinding_get()
Tests get of specific port bindings from the database.

test_nexusportswitchbinding_get()
Tests get of port bindings based on port and switch.

test_nexusportvlanswitchbinding_get()
Tests get of port bindings based on port, vlan, and switch.

test_nexusvlanbinding_get()
Test get of port bindings based on vlan and switch.

test_nexusvmbinding_get()
Test get of port bindings based on vlan and instance.

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_db.**TestCiscoNexusHostM

Bases: neutron.tests.unit.testlib_api.SqlTestCase

Tests for Nexus Mechanism driver Host Mapping database.

test_enet_host_mapping_db()

test_portchannel_host_mapping_db()

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_db.**TestCiscoNexusVpcA**

Bases: neutron.tests.unit.testlib_api.SqlTestCase

Unit tests for Cisco mechanism driver's Nexus vpc alloc database.

setUp()

test_vpcalloc_init()

test_vpcalloc_min_max()

test_vpcalloc_rollback()

test_vpcalloc_test_alloc_collision()

networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events module

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.**TestCiscoNexusI**

Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBase*

Unit tests for Cisco ML2 Nexus baremetal device driver.

baremetal_profile = {'local_link_information': [{'port_id': 'Ethernet 1/10', 'switch_

baremetal_profile_is_native = {'local_link_information': [{'port_id': 'Ethernet 1/10

baremetal_profile_vPC = {'local_link_information': [{'port_id': 'Ethernet 1/10', 'sw

get_init_side_effect(*action*, *ipaddr=None*, *body=None*, *headers=None*)

get_init_side_effect2(*action*, *ipaddr=None*, *body=None*, *headers=None*)

setUp()
Sets up mock driver, and switch and credentials dictionaries.

test_automated_port_channel_creation_deletion()
Basic creation and deletion test of 1 auto port-channel.

test_automated_port_channel_w_user_cfg()
Basic creation and deletion test of 1 auto port-channel.

test_configs = {'test_config_vm': TestConfigObj(nexus_ip_addr='1.1.1.1', host_name='ba

test_create_delete_automated_vpc_and_vm()
Basic creation and deletion test of 2 auto port-channel and vm.

test_create_delete_basic_bm_ethernet_port_and_vm()
Basic creation and deletion test of 1 ethernet port.

test_create_delete_basic_eth_port_is_native()
Basic creation and deletion test of 1 ethernet port.

test_create_delete_basic_port_channel()
Basic creation and deletion test of 1 port-channel.

test_create_delete_learn_vpc_and_vm()
Basic creation and deletion test of 2 learn port-channel and vm.

test_create_delete_switch_ip_not_defined()
Create/delete of 1 ethernet port switchinfo is string.

test_failure_inconsistent_learned_chgrp()
Learning chgrp but different on both eth interfaces.

test_failure_inconsistent_new_chgrp()
Started as newly created chgrp but one if had chgrp configured.

test_vpcids_depleted_failure()
Verifies exception when failed to get vpcid.

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.**TestCiscoNexusI**
Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBaseResults*

Unit tests driver results for Cisco ML2 Nexus.

test_results = {'driver_result_unique_auto_vPC_vm_add1': [['api/mo.json', '1.1.1.1',

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.**TestCiscoNexusI**

Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBase*, *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.TestCiscoNexusDeviceConfig*, *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.TestCiscoNexusDeviceResults*

Unit tests for Cisco ML2 Nexus baremetal VPC Config.

The purpose of this test case is to validate vpc pool initialization. If vpc-pool is configured, it will be compared with what currently exists in the vpc pool data base. Adds and removals of the data base will occur. Removal will not occur if the entry is active.

setUp()

test_vpc_config_db_results_bad_config1()
Config vpc-pool config with garbage. log & no db entries.

test_vpc_config_db_results_bad_config2()
Config vpc-pool config with bad range. log & no db entries.

test_vpc_config_db_results_bad_config3()
Config vpc-pool config with bad digits. log & no db entries.

test_vpc_config_db_results_bad_config_keep_old()
Verify on config error, existing db entries stay intact.

test_vpc_config_db_results_bad_vpc_range()
Config vpc-pool config with bad min/max values.

test_vpc_config_db_results_good_config_all()
Config valid vpc-pool range config. Test Min/Max vpc value.

test_vpc_config_db_results_good_config_not_range()
Config valid vpc-pool not range config.

test_vpc_config_db_results_good_config_range()
Config valid vpc-pool range config.

test_vpc_config_db_results_removal()
Allow user to remove config but only non-active.

test_vpc_config_db_results_with_old_config1()
Config valid vpc-pool compare with pre-existing entries.

test_vpc_config_db_results_with_old_config2()
Config valid vpc-pool compare with pre-existing entries.

test_vpc_config_db_results_with_old_config3()
Config valid vpc-pool compare with pre-existing entries.

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.**TestCiscoNexusI**

Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBase*, *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.TestCiscoNexusDeviceConfig*, *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.TestCiscoNexusDeviceResults*

Unit tests for Cisco ML2 Nexus device driver.

setUp()
Sets up mock driver, and switch and credentials dictionaries.

test_create_delete_dhcp()
Tests creation and deletion of ports with device_owner of dhcp.

test_create_delete_diff_switch_same_host()
Test create/delete of two Ports, diff switch/same host.

test_create_delete_dual()
Tests creation and deletion of dual ports for single server

test_create_delete_duplicate_port_transaction()
Tests creation and deletion same port transaction.

test_create_delete_duplicate_ports()
Tests creation and deletion of two new virtual Ports.

test_create_delete_portchannel()
Tests creation of a port over a portchannel.

test_create_delete_router_gateway()
Tests creation and deletion of ports with device_owner of router_gateway.

test_create_delete_router_ha_intf()
Tests creation and deletion of ports with device_owner of router_ha_interface.

test_create_delete_router_intf()
Tests creation and deletion of ports with device_owner of router_interface.

test_create_delete_same_switch_diff_hosts_diff_vlan()
Test create/delete two Ports, same switch/diff host & vlan.

test_create_delete_same_switch_diff_hosts_same_vlan()
Test create/delete two Ports, same switch & vlan/diff host.

test_nexus_vm_migration()
Verify VM (live) migration.

Simulate the following: Nova informs neutron of live-migration with port-update(new host). This should trigger two update_port_pre/postcommit() calls.

The first one should only change the current host_id and remove the binding resulting in the mechanism drivers receiving:

- PortContext.original['binding:host_id']: previous value
- PortContext.original_top_bound_segment: previous value
- PortContext.current['binding:host_id']: current (new) value
- PortContext.top_bound_segment: None

The second one binds the new host resulting in the mechanism drivers receiving:

- PortContext.original['binding:host_id']: previous value
- PortContext.original_top_bound_segment: None
- PortContext.current['binding:host_id']: previous value
- PortContext.top_bound_segment: new value

test_update_postcommit_port_not_found()

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.**TestCiscoNexusI**

Bases: object

Unit tests Config for Cisco ML2 Nexus device driver.

test_configs = OrderedDict([('test_config1', TestConfigObj(nexus_ip_addr='1.1.1.1', hos

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.**TestCiscoNexusI**

Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBase*, *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.TestCiscoNexusDeviceConfig*, *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.TestCiscoNexusDeviceResults*

Negative Unit tests for Cisco ML2 Nexus device driver.

setUp()
Sets up mock driver, and switch and credentials dictionaries.

test_connect_failure()
Verifies exception handling during driver connect.

test_create_trunk_failure()
Verifies exception during create trunk interface driver.

test_create_vlan_failure()
Verifies exception during edit vlan create driver.

test_delete_trunk_failure()
Verifies exception during delete trunk interface driver.

test_delete_vlan_failure()
Verifies exception during edit vlan delete driver.

test_fail_on_connect_other_exceptions()
Test other errors during connect() sequences are still handled.

test_get_nexus_type_failure()
Verifies exception during get nexus type.

test_nexus_host_not_configured()
Test handling of a host not found in our configuration.

If a host is not found in the cisco configuration the driver should silently ignore (unknown host name is logged) and no database or switch configuration is performed. Exercise against all APIs.

test_nexus_invalid_segment()
Test handling of a non VLAN segment.

Pass a FLAT segment type into the driver. Verify that no exceptions are raised (non-VLAN segments are logged only) and that no database or switch configuration is performed.

test_nexus_missing_fields()
Test handling of a NexusMissingRequiredFields exception.

Test the Cisco NexusMissingRequiredFields exception by using empty device_id value during port creation.

test_nexus_segment_none()
Test handling of segment is None.

Verify that None segments do not throw an exception in _port_action_xxx. None segments passed to the event handlers are logged and are not processed.

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.**TestCiscoNexusI**

Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBase*, *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.TestCiscoNexusDeviceConfig*

Verifies interface vlan allowed none is set when missing.

restapi_mock_init()

setUp()
Sets up mock driver, and switch and credentials dictionaries.

test_verify_initialization()

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.**TestCiscoNexusI**
Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBaseResults*

Unit tests driver results for Cisco ML2 Nexus.

test_results = {'add_port_channel_driver_result': [['api/mo.json', '2.2.2.2', '{"topSy

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events.**TestCiscoNexusI**
Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBaseResults*

Unit tests driver results for Cisco ML2 Nexus.

test_results = {'duplicate_init_port_driver_result1': [['api/mo/sys/intf/phys-[eth1/1

networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events_vxlan module

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events_vxlan.**TestCisco**

Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBase*, *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events_vxlan.TestCiscoNexusVxlanDeviceConfig*

Unit tests for Cisco ML2 VXLAN Nexus device driver.

setUp()
Sets up mock driver, and switch and credentials dictionaries.

test_create_nve_member_failure()
Verifies exception during create nve member driver.

test_delete_nve_member_failure()
Verifies exception during delete nve member driver.

test_disable_vxlan_feature_failure()
Verifies exception during disable VXLAN driver.

test_enable_vxlan_feature_failure()
Verifies exception during enable VXLAN driver.

test_nexus_missing_vxlan_fields()
Test handling of a VXLAN NexusMissingRequiredFields exception.

Test the Cisco NexusMissingRequiredFields exception by using empty VNI and mcast address values during port update event.

test_nexus_vxlan_bind_port()
Test VXLAN bind_port method processing.

Verify the bind_port method allocates the VLAN segment correctly.

test_nexus_vxlan_bind_port_no_dynamic_segment()
Test VXLAN bind_port processing.

Verify that the continue_binding() method is not called when the vlan dynamic segment wasn't allocated.

test_nexus_vxlan_bind_port_no_physnet()
Test VXLAN bind_port error processing.

Verify that continue_binding() method is not called when no 'physnet' key is present in the nexus switch dictionary.

test_nexus_vxlan_one_network()
Test processing for creating one VXLAN segment.

test_nexus_vxlan_one_network_two_hosts()
Tests creation and deletion of two new virtual Ports.

test_nexus_vxlan_two_network()
Test processing for creating one VXLAN segment.

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events_vxlan.**TestCisco**
Bases: object

Config Data for Cisco ML2 VXLAN Nexus device driver.

test_configs = OrderedDict([('test_vxlan_config1', TestConfigObj(nexus_ip_addr='1.1.1.

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_events_vxlan.**TestCisco**
Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBaseResults*

Unit tests driver results for Cisco ML2 Nexus.

test_results = {'add_port_driver_result2': [['api/mo/sys/epId-1/nws/vni-70001.json',

networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_replay module

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_replay.**TestCiscoNexusR**

Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusReplayBase*

Unit tests for Replay of Cisco ML2 Nexus data.

baremetal_profile = {'local_link_information': [{'port_id': 'Ethernet 1/10', 'switch_

baremetal_profile2 = {'local_link_information': [{'port_id': 'Ethernet 1/20', 'switch

baremetal_profile_vPC = {'local_link_information': [{'port_id': 'Ethernet 1/10', 'sw

get_init_side_effect(*action*, *ipaddr=None*, *body=None*, *headers=None*)

get_init_side_effect2(*action*, *ipaddr=None*, *body=None*, *headers=None*)

setUp()
Sets up mock driver, and switch and credentials dictionaries.

test_configs = OrderedDict([('test_config_vm', TestConfigObj(nexus_ip_addr='1.1.1.1',

test_replay_automated_port_channel_w_user_cfg()
Basic replay of auto-port-channel creation with user config.

test_replay_automated_vPC_ports_and_vm()
Provides replay data and result data for unique ports.

test_replay_unique_ethernet_port_and_vm()
Provides replay data and result data for unique ports.

test_replay_unique_ethernet_ports()
Provides replay data and result data for unique ports.

test_replay_unique_vPC_ports()
Provides replay data and result data for unique ports.

test_replay_unique_vPC_ports_and_vm()
Provides replay data and result data for unique ports.

test_replay_unique_vPC_ports_chg_to_enet()
Persist with learned channel group even if it was removed.

test_replay_unique_vPC_ports_chg_vPC_nbr()
Persist with learned channel group even if it changed.

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_replay.**TestCiscoNexusR**
Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBaseResults*

Unit tests driver results for Cisco ML2 Nexus.

test_results = {'driver_result_unique_vPC470_add2': [['api/mo.json', '1.1.1.1', '{"to

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_replay.**TestCiscoNexusR**

Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusReplayBase*

Unit tests for Replay of Cisco ML2 Nexus data.

setUp()
Sets up mock driver, and switch and credentials dictionaries.

test_configs = OrderedDict([('test_replay_dual', TestConfigObj(nexus_ip_addr='4.4.4.4'

test_replay_create_fails_if_single_switch_down()
Verifies port create fails if switch down.

test_replay_create_vlan_failure_during_replay()
Verifies exception during create vlan while replaying.

test_replay_delete_success_if_switch_down()
Verifies port delete success if switch down.

test_replay_duplicate_ports()
Provides replay data and result data for duplicate ports.

test_replay_duplicate_vlan()
Provides replay data and result data for duplicate vlans.

test_replay_get_nexus_type_failure()
Verifies exception during get nexus_type while replaying.

test_replay_get_nexus_type_failure_two_switches()
Verifies exception during driver get nexus type.

test_replay_new_port_success_if_one_switch_up()
Verifies create port successful if one multi-switch up.

test_replay_no_retry_failure_handling()
Tests to check replay 'no retry' failure handling.

1) Verify config_failure is incremented upon failure during replay config and verify create_vlan transactions are seen. 2) Verify contact_failure is incremented upon failure during get_nexus_type transaction. 3) Verify receipt of new transaction does not reset failure statistics. 4) Verify config&contact_failure is reset when replay is successful.

test_replay_port_success_if_one_switch_restored()
Verifies port restored after one of multi-switch restored.

test_replay_unique_ports()
Provides replay data and result data for unique ports.

test_replay_update_fails_if_single_switch_down()
Verifies port update fails if switch down.

test_replay_vlan_batch_failure_during_replay()
Verifies handling of batch vlan during replay.

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_replay.**TestCiscoNexusR**
Bases: *networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_base.TestCiscoNexusBaseResults*

Unit tests driver results for Cisco ML2 Nexus.

test_results = {'switch_up_result_add': [['api/mo.json', '4.4.4.4', '{"topSystem": {

networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_restapi_client module

Basic Test Class to verify REST API Client code "nexus_restapi_client.py"

class networking_cisco.tests.unit.ml2_drivers.nexus.test_cisco_nexus_restapi_client.**TestCis**

Bases: neutron.tests.unit.testlib_api.SqlTestCase

Unit tests for Cisco REST API client.

json()

json_cli()

json_err()

setUp()

test_bad_json_with_get_nexus_type()

test_verify_for_cli_no_cert()

test_verify_for_cli_with_local_cert()

test_verify_no_certificate()

test_verify_with_local_certificate()

test_verify_with_nonlocal_certificate()

networking_cisco.tests.unit.ml2_drivers.nexus.test_provider_network module

class networking_cisco.tests.unit.ml2_drivers.nexus.test_provider_network.**TestCiscoNexusPro**

Bases: neutron.tests.unit.testlib_api.SqlTestCase

Test the provider network code added to the cisco nexus MD.

setUp()

test_create_network()

test_create_network_false_provider()

test_create_network_no_provider()

test_delete_network()

test_delete_network_no_id()

test_port_action_vlan_no_provider()

test_port_action_vlan_provider()

class networking_cisco.tests.unit.ml2_drivers.nexus.test_provider_network.**TestCiscoNexusPro**

Bases: neutron.tests.base.BaseTestCase

Test the provider network configuration used by the cisco nexus MD.

setUp()

test_pnet_configure_create()

test_pnet_configure_create_and_trunk()

test_pnet_configure_not_providernet()

test_pnet_configure_trunk()

test_pnet_delete_create()

test_pnet_delete_create_and_trunk()

test_pnet_delete_not_providernet()

test_pnet_delete_trunk()

test_pnet_replay_not_providernet()

test_pnet_replay_providernet_create()

test_pnet_replay_providernet_create_and_trunk()

test_pnet_replay_providernet_trunk()

class networking_cisco.tests.unit.ml2_drivers.nexus.test_provider_network.**TestCiscoNexusPro**

Bases: neutron.tests.base.BaseTestCase

Test the provider network extension class used by the cisco nexus MD.

setUp()

test_create_network_no_vlan()

test_create_network_none_vlan()

test_create_network_vlan()

test_extension_alias()

networking_cisco.tests.unit.ml2_drivers.nexus.test_trunk module

class networking_cisco.tests.unit.ml2_drivers.nexus.test_trunk.**TestNexusTrunkHandler**(**args*, ***kwds*)

Bases: neutron.tests.unit.db.test_db_base_plugin_v2.NeutronDbPluginV2TestCase

setUp()

test_is_trunk_parentport()

test_is_trunk_parentport_no_trunk()

test_is_trunk_subport()

test_is_trunk_subport_baremetal()

test_is_trunk_subport_baremetal_no_subport()

test_is_trunk_subport_baremetal_vm_port()

test_is_trunk_subport_invalid_deviceowner()

test_update_subports_baremetal()

class networking_cisco.tests.unit.ml2_drivers.nexus.test_trunk.**TestSubPort**

Bases: object

port_id = 'fake_port_id'

segmentation_id = 101

segmentation_type = 'vlan'

trunk_id = 'fake_trunk_id'

class networking_cisco.tests.unit.ml2_drivers.nexus.test_trunk.**TestTrunk**

Bases: object

admin_state_up = 'test_admin_state'

id = 'fake_trunk_id'

name = 'test_trunk_name'

port_id = 'fake_port_id'

status = 'ACTIVE'

sub_ports = [{'port_id': 'fake_port_id', 'segmentation_id': 101, 'segmentation_type'

tenant_id = 'test_tenant_id'

update = <Mock id='139693349710816'>

networking_cisco.tests.unit.ml2_drivers.nexus.test_type_nexus_vxlan module

class networking_cisco.tests.unit.ml2_drivers.nexus.test_type_nexus_vxlan.**NexusVxlanTypeTes**

Bases: neutron.tests.unit.testlib_api.SqlTestCase

setUp()

test_allocate_shared_mcast_group()

test_allocate_tenant_segment()

test_invalid_vni_ranges()

test_reserve_provider_segment_full_specs()

test_reserve_provider_segment_partial_specs()

vni_in_range(*vni*)

Module contents

networking_cisco.tests.unit.ml2_drivers.ucsm package

Submodules

networking_cisco.tests.unit.ml2_drivers.ucsm.test_cisco_ucsm_common module

class networking_cisco.tests.unit.ml2_drivers.ucsm.test_cisco_ucsm_common.**ConfigMixin**

Bases: object

Mock config for UCSM driver.

mocked_parser = None

set_up_mocks()

class networking_cisco.tests.unit.ml2_drivers.ucsm.test_cisco_ucsm_common.**UCSMConfigTestCas**

Bases: *networking_cisco.tests.base.TestCase*

setUp()

test_add_sp_template_config_for_host()

test_oslo_config_configuration_loading()

test_support_pci_devices_bad_format()

test_update_sp_template_config()

networking_cisco.tests.unit.ml2_drivers.ucsm.test_cisco_ucsm_driver module

class networking_cisco.tests.unit.ml2_drivers.ucsm.test_cisco_ucsm_driver.**FakeNetworkContex**

Bases: neutron_lib.plugins.ml2.api.NetworkContext

Network context for testing purposes only.

current

network_segments

original

class networking_cisco.tests.unit.ml2_drivers.ucsm.test_cisco_ucsm_driver.**FakePortContext**(

Bases: object

Port context for testing purposes only.

bottom_bound_segment

current

network

original

segment

set_binding(*segment_id*, *vif_type*, *vif_details*, *status=None*)

class networking_cisco.tests.unit.ml2_drivers.ucsm.test_cisco_ucsm_driver.**FakeServer**(*server*)
Bases: object

class networking_cisco.tests.unit.ml2_drivers.ucsm.test_cisco_ucsm_driver.**FakeServiceProfi**
Bases: object

Fake Service Profile class for testing only.

next()

class networking_cisco.tests.unit.ml2_drivers.ucsm.test_cisco_ucsm_driver.**FakeUcsmHandle**(*po*

Bases: object

Ucsm connection handle for testing purposes only.

commit()

logout()

query_classid(*class_id*)

query_dn(*dn*)

remove_mo(*p_profile*)

class networking_cisco.tests.unit.ml2_drivers.ucsm.test_cisco_ucsm_driver.**TestCiscoUcsmMech**

Bases: neutron.tests.unit.testlib_api.SqlTestCase, *networking_cisco.tests.unit.ml2_drivers.ucsm.test_cisco_ucsm_common.ConfigMixin*

Unit tests for Cisco ML2 UCS Manager MD.

setUp()
Sets up mock Ucs Sdk.

test_add_sp_template_config_to_db()
Verifies the SP template dict has been created properly.

test_bind_port_active()
Verifies bind_port sets the port status as active.

test_check_segment_vlan()
Verifies VLAN network segments are supported.

test_check_segment_vxlan()
Verifies VXLAN network segments are not supported.

test_delete_network_precommit_no_segments(*mock_delete_vlan_entry*, *mock_delete_vnic*, *mock_delete_sp_template*)

test_delete_network_precommit_vlan_segment(*mock_delete_vlan_entry*, *mock_delete_vnic*, *mock_delete_sp_template*)

test_generic_create_profile()
Test to verify duplicate creation exceptions.

This is a generic test to mimic the behavior of any UCS Manager driver function that creates a profile on the UCS Manager. The first time the profile is created, the create succeeds if all parameters are correct. If we attempt to create it any number of times after that, UCS Manager throws an exception. This test code mimics that behavior by using counter to keep track of how many times 'update_serviceprofile' is being called. counter == 0 -> Simulates invalid input, so raise an exception. counter == 1 -> Simulates valid inputs and 1st creation request. counter > 1 -> Simulates duplicate creation request and results in UCS Manager throwing a duplicate creation request.

test_get_physnet()

test_get_ucsm_ip_for_host_failure()
Tests that case where UCSM does not control this host.

test_get_ucsm_ip_for_host_success()
Verfies that ucsm_ip to Service Profile mapping is successful.

test_host_id_to_hostname()
Verifies extraction of hostname from host-id from Nova.

test_learn_sp_and_template_for_host_error()
Tests case where learning config from UCSM gives diff host.'

test_learn_sp_and_template_for_host_exp()
Tests case where reading config from UCSM generates exception.

test_learn_sp_and_template_for_host_success()
Tests case where learning config from UCSM gives correct host.'

test_normal_vnic_type()
Verifies NORMAL vnic type is not supported.

test_parse_virtio_eth_ports()
Verifies eth_port_list contains a fully-formed path.

test_port_profile_delete_on_ucsm()
Verifies that the PP delete retry logic.

test_port_profile_delete_table_add()
Verifies that add and get of 1 PP to delete table works.

test_port_supported_deviceowner()
Verifies detection of supported set of device owners for ports.

test_port_supported_status()
Verifies detection of supported status values for ports.

test_port_unsupported_deviceowner()
Verifies detection of unsupported device owners for ports.

test_port_unsupported_status()
Verifies detection of unsupported status values for ports.

test_pp_delete_table_add_multiple()
Verifies that add and get of multiple PPs to delete table works.

test_remove_non_existent_port_profile_from_table()
Verifies that removing previously deleted PP works.

test_remove_port_profile_from_table()
Verifies that removing entry from PP delete table works.

test_sriov_update_port_precommit()
Verifies MD does not create Port Profiles for SR-IOV ports.

test_sriov_vnic_type_and_vendor_info()
Verifies SR-IOV port and MACVTAP vnic_type are supported.

test_ucs_manager_disconnect_fail()
Verifies UCS Manager driver is called with correct parameters.

test_unsupported_vnic_type_and_vendor_info()
Verifies unsupported pci vendor is rejected.

test_update_port_postcommit_direct()
Verifies UCS Manager driver is called with correct parameters.

test_update_port_postcommit_failure()
Verifies duplicate Port Profiles are not being created.

test_update_port_postcommit_macvtap()
Verifies UCS Manager driver is called with correct parameters.

test_update_port_postcommit_normal()
Verifies UCS Manager driver is called with correct parameters.

test_update_port_postcommit_success()
Verifies duplicate Port Profiles are not being created.

test_update_port_postcommit_vnic_template()
Verifies UCSM driver works correcly with VNIC Templates.

test_validate_vm_fex_port_bad()
Verifies unsupported pci vendor is not VM-FEX capable.

test_validate_vm_fex_port_cisco()
Verifies port's pci vendor info makes it VM-FEX capable.

test_validate_vm_fex_port_sriov()
Verifies valid SR-IOV port is not VM-FEX capable.

test_virtio_update_port_precommit()
Verifies MD adds VNIC Template to DB for Neutron virtio ports.

test_vmfex_update_port_precommit()
Verifies MD saves relevant info for VM-FEX ports into DB.

test_vmfex_vnic_type_and_vendor_info()
Verifies VM-FEX port is recognized as a supported vendor.

test_vnic_template_db_methods()
Verifies VNIC Template DB methods.

networking_cisco.tests.unit.ml2_drivers.ucsm.test_ucsm_ssl module

class networking_cisco.tests.unit.ml2_drivers.ucsm.test_ucsm_ssl.**TestCiscoUcsmSSL**(**args*, ***kwds*)

Bases: neutron.tests.base.BaseTestCase

Unit tests for SSL overrides.

test_SSLContext_verify_false()

test_SSLContext_verify_true()

test_wrap_socket_verify_false()

test_wrap_socket_verify_false_cert_reqs_true()

test_wrap_socket_verify_true()

test_wrap_socket_verify_true_cert_reqs_false()

class networking_cisco.tests.unit.ml2_drivers.ucsm.test_ucsm_ssl.**TestUcsmsdkPatch**(**args*, ***kwds*)

Bases: neutron.tests.base.BaseTestCase

Unit tests for Cisco ML2 UCS Manager SSL override for ucsmsdk.

test_ucsmsdk_default_behaviour_of_ssl_cert_checking(*mocked_socket*)

test_ucsmsdk_ssl_monkey_patch(*mock_create_host*)

Module contents

Module contents

networking_cisco.tests.unit.neutronclient package

Submodules

networking_cisco.tests.unit.neutronclient.test_cli20_hostingdevice module

class networking_cisco.tests.unit.neutronclient.test_cli20_hostingdevice.**CLITestV20HostingD**

Bases: neutronclient.tests.unit.test_cli20.CLITestV20Base

setUp()

test_create_hosting_device()
Create hosting device.

test_create_hosting_device_admin_down()

test_create_hosting_device_auto_delete()

test_create_hosting_device_cfg_agent()

test_create_hosting_device_creds()

test_create_hosting_device_description()

test_create_hosting_device_device_id()

test_create_hosting_device_full()

test_create_hosting_device_id()
Create hosting device: –id this_id “Device 1” “Template 1”.

test_create_hosting_device_mgmt_ip()

test_create_hosting_device_mgmt_port()

test_create_hosting_device_proto_port()

test_create_hosting_device_tenant()
Create hosting device: –tenant_id tenantid “Device 1” “Template 1”.

test_create_hosting_device_tenant_bound()

test_delete_hosting_device()
Delete hosting device: myid.

test_ext_cmd_help_doc_with_extension_name()

test_ext_cmd_loaded()

test_get_hosting_device_config()
Get config of hosting device: myid.

test_list_hosting_devices_detail()
list hosting devices: -D.

test_list_hosting_devices_limit()
list hosting devices: -P.

test_list_hosting_devices_sort()
list hosting devices: –sort-key name –sort-key id –sort-key asc –sort-key desc

test_show_hosting_device()
Show hosting device: myid.

test_update_hosting_device_admin_state()

test_update_hosting_device_auto_delete()

test_update_hosting_device_creds()

test_update_hosting_device_description()

test_update_hosting_device_device_id()

test_update_hosting_device_exception()
Update hosting device: myid.

test_update_hosting_device_mgmt_ip()

test_update_hosting_device_name()
Update hosting device: myid –name myname.

test_update_hosting_device_proto_port()

test_update_hosting_device_tenant_bound()

networking_cisco.tests.unit.neutronclient.test_cli20_hostingdevicescheduler module

class networking_cisco.tests.unit.neutronclient.test_cli20_hostingdevicescheduler.**CLITestV**

Bases: neutronclient.tests.unit.test_cli20.CLITestV20Base

test_associate_hosting_device_with_cfg_agent()

test_disassociate_hosting_device_with_cfg_agent()

test_list_cfg_agents_handling_hosting_device()

test_list_hosting_devices_handled_by_cfg_agent()

networking_cisco.tests.unit.neutronclient.test_cli20_hostingdevicetemplate module

class networking_cisco.tests.unit.neutronclient.test_cli20_hostingdevicetemplate.**CLITestV2**

Bases: neutronclient.tests.unit.test_cli20.CLITestV20Base

setUp()

test_create_hosting_device_template()
Create hosting device template.

test_create_hosting_device_template_boot_time()

test_create_hosting_device_template_conf_mech()

test_create_hosting_device_template_creds()

test_create_hosting_device_template_desired_slots_free()

test_create_hosting_device_template_device_driver()

test_create_hosting_device_template_disabled()

test_create_hosting_device_template_flavor()

test_create_hosting_device_template_full()

test_create_hosting_device_template_id()
Create hosting device template: –id this_id “Device 1” “Template 1”.

test_create_hosting_device_template_image()

test_create_hosting_device_template_plugging_driver()

test_create_hosting_device_template_proto_port()

test_create_hosting_device_template_service_types()

test_create_hosting_device_template_slot_capacity()

test_create_hosting_device_template_tenant()
Create hosting device template: –tenant_id tenantid “Device 1” “Template 1”.

test_create_hosting_device_template_tenant_bound()

test_delete_hosting_device_template()
Delete hosting device template: myid.

test_ext_cmd_help_doc_with_extension_name()

test_ext_cmd_loaded()

test_list_hosting_device_templates_detail()
list hosting device templates: -D.

test_list_hosting_device_templates_limit()
list hosting device templates: -P.

test_list_hosting_device_templates_sort()
list hosting device templates: –sort-key name –sort-key id –sort-key asc –sort-key desc

test_show_hosting_device_template()
Show hosting device template: myid.

test_update_hosting_device_template_boot_time()

test_update_hosting_device_template_conf_mech()

test_update_hosting_device_template_creds()

test_update_hosting_device_template_disabled()

test_update_hosting_device_template_flavor()

test_update_hosting_device_template_full()

test_update_hosting_device_template_image()

test_update_hosting_device_template_proto_port()

test_update_hosting_device_template_service_types()

test_update_hosting_device_template_tenant_bound()

networking_cisco.tests.unit.neutronclient.test_cli20_networkprofile module

class networking_cisco.tests.unit.neutronclient.test_cli20_networkprofile.**CLITestV20Networ**

Bases: neutronclient.tests.unit.test_cli20.CLITestV20Base

setUp()

test_create_networkprofile()
Create networkprofile: myid.

test_create_networkprofile_overlay()
Create networkprofile: myid.

test_delete_networkprofile()
Delete networkprofile: myid.

test_ext_cmd_loaded()

test_list_networkprofile_detail()
List networkprofile: -D.

test_list_networkprofile_fields()
List networkprofile: –fields a –fields b – –fields c d.

test_list_networkprofile_known_option_after_unknown()
List networkprofile: – –tags a b –request-format xml.

test_list_networkprofile_overlay_detail()
List networkprofile: -D.

test_show_networkprofile()
Show networkprofile: –fields id –fields name myid.

networking_cisco.tests.unit.neutronclient.test_cli20_policyprofile module

class networking_cisco.tests.unit.neutronclient.test_cli20_policyprofile.**CLITestV20PolicyPr**

Bases: neutronclient.tests.unit.test_cli20.CLITestV20Base

setUp()

test_ext_cmd_loaded()

test_list_policyprofile_detail()
List policyprofile: -D.

test_list_policyprofile_fields()
List policyprofile: –fields a –fields b – –fields c d.

test_list_policyprofile_known_option_after_unknown()
List policyprofile: – –tags a b –request-format xml.

test_show_policyprofile()
Show policyprofile: –fields id –fields name myid.

networking_cisco.tests.unit.neutronclient.test_cli20_routerscheduler module

class networking_cisco.tests.unit.neutronclient.test_cli20_routerscheduler.**CLITestV20L3Rout**

Bases: neutronclient.tests.unit.test_cli20.CLITestV20Base

test_add_router_to_hosting_device()

test_list_hosting_devices_hosting_router()

test_list_routers_on_hosting_device()

test_remove_router_from_hosting_device()

networking_cisco.tests.unit.neutronclient.test_cli20_routertype module

class networking_cisco.tests.unit.neutronclient.test_cli20_routertype.**CLITestV20RouterType**

Bases: neutronclient.tests.unit.test_cli20.CLITestV20Base

setUp()

test_create_router_type()
Create router type.

test_create_router_type_description()

test_create_router_type_full()

test_create_router_type_ha()

test_create_router_type_id()
Create router type: –id this_id myname.

test_create_router_type_name()

test_create_router_type_slots()

test_create_router_type_tenant()
Create router type: –tenant_id tenantid myname.

test_create_router_type_unshared()

test_delete_router_type()
Delete routertype: myid.

test_ext_cmd_help_doc_with_extension_name()

test_ext_cmd_loaded()

test_list_router_types_detail()
list routers: -D.

test_list_router_types_limit()
list routertypes: -P.

test_list_router_types_sort()
list routertypes: –sort-key name –sort-key id –sort-key asc –sort-key desc

test_show_router_type()
Show routertype: myid.

test_update_router_type_description()

test_update_router_type_exception()
Update routertype: myid.

test_update_router_type_full()

test_update_router_type_ha()

test_update_router_type_name()
Update routertype: myid –name myname.

test_update_router_type_sharing()

test_update_router_type_slots()

Module contents

networking_cisco.tests.unit.saf package

Subpackages

networking_cisco.tests.unit.saf.agent package

Subpackages

networking_cisco.tests.unit.saf.agent.topo_disc package

Submodules

networking_cisco.tests.unit.saf.agent.topo_disc.test_pub_lldp_api module

class networking_cisco.tests.unit.saf.agent.topo_disc.test_pub_lldp_api.**LldpApiTest**(**args*, ***kwds*)

Bases: neutron.tests.base.BaseTestCase

A test suite to exercise the public LldpApi class.

setUp()
Setup for LldpApiTest.

test_common_tlv_format_no_tlv_case()
Test _check_common_tlv_format when specific TLV is not present.

test_common_tlv_format_no_tlv_data_case()
Test _check_common_tlv_format when TLV data is not present.

test_common_tlv_format_no_tlv_data_pattern_case()
Test _check_common_tlv_format when TLV pattern is not present.

test_common_tlv_format_none_case()
Test _check_common_tlv_format when TLV data is None.

test_enable_lldp_invalid_case()
Test for enable_lldp function for neither NCB or NB DMAC case.

test_enable_lldp_nb()
Test for enable_lldp function for NB DMAC case.

test_enable_lldp_ncb_correct_reply()
Test for enable_lldp function for correct return value.

test_enable_lldp_ncb_incorrect_reply()
Test for enable_lldp function for incorrect return value.

test_get_lldp_tlv_nb()
Test for get_lldp_tlv for nb DMAC case.

test_get_lldp_tlv_ncb()
Test for get_lldp_tlv for ncb DMAC case.

test_remote_chassis_id_mac()
Test the get_remote_chassis_id_mac function.

test_remote_evb_cfgd()
Test the case when remove EVB TLV is present.

False case is tested in the test_common.. cases.

test_remote_evb_mode()
Test the get_remote_evb_mode function.

test_remote_evb_mode_incorrect()
Test the get_remote_evb_mode function for incorrect TLV.

test_remote_mgmt_addr()
Test the get_remote_mgmt_addr function.

test_remote_port()
Test the get_remote_port function.

test_remote_port_id_local()
Test the get_remote_port_id_local function.

test_remote_port_id_mac()
Test the get_remote_port_id_mac function.

test_remote_system_desc()
Test the get_remote_system_desc function.

test_remote_system_name()
Test the get_remote_system_name function.

networking_cisco.tests.unit.saf.agent.topo_disc.test_topo_disc module

class networking_cisco.tests.unit.saf.agent.topo_disc.test_topo_disc.**TopoDiscTest**(**args*, ***kwds*)

Bases: neutron.tests.base.BaseTestCase

A test suite to exercise the TopoDisc Class.

setUp()
Setup Routine.

test_cfg_intf()
Test the cfg_intf function.

test_cfg_lldp_interface()
Test the test_cfg_lldp_interface function.

test_cfg_lldp_interface_error()
Test the test_cfg_lldp_interface function when it returns False.

test_cfg_lldp_interface_list()
Test the cfg_lldp_interface_list function.

test_cmp_store_tlv_params_all_false()
Test the test_cmp_store_tlv_params for all false case.

This test the case when all TLV compare functions returns False.

test_cmp_store_tlv_params_all_true()
Test the test_cmp_store_tlv_params for all True case.

This test the case when all TLV compare functions returns True.

test_cmp_store_tlv_params_one_true()
Test the test_cmp_store_tlv_params for one True case.

This test the case when all TLV compare functions returns True.

test_init_cfg_interfaces()
Test the _init_cfg_interfaces function.

test_period_task_reset_case_all_false()
Test the periodic task when callback is not invoked.

When all conditions for calling callback returns False.

test_period_task_reset_case_bond_intf_change_true()
Test the periodic task when bond_intf_change returns True.

Also, the callback invoked case is verified for a return value of False.

test_period_task_reset_case_cmp_true()
Test the periodic task when cmp_store_tlv_params returns True.

Also, the callback invoked case is verified for a return value of True.

test_period_task_reset_case_get_db_retry_true()
Test the periodic task when get_db_retry_status returns True.

Also, the callback invoked case is verified for a return value of False.

test_period_task_reset_case_lldp_status_false()
Test the periodic task when LLDP status is False.

test_period_task_reset_case_topo_disc_cnt_exceed_threshold()
Test the periodic task when send failure cnt exceeds threshold.

Also, the callback invoked case is verified for a return value of True.

test_remote_chassis_id_mac_uneq_store_equal()
Test remote_chassis_id_mac_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_chassis_id_mac_uneq_store_unequal()
Test remote_chassis_id_mac_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_evb_cfgd_uneq_store_equal()
Test remote_evb_cfgd_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_evb_cfgd_uneq_store_unequal()
Test remote_evb_cfgd_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_evb_mode_uneq_store_equal()
Test remote_evb_mode_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_evb_mode_uneq_store_unequal()
Test remote_evb_mode_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_mgmt_addr_uneq_store_equal()
Test remote_mgmt_addr_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_mgmt_addr_uneq_store_unequal()
Test remote_mgmt_addr_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_port_id_mac_uneq_store_equal()
Test remote_port_id_mac_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_port_id_mac_uneq_store_unequal()
Test remote_port_id_mac_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_port_uneq_store_equal()
Test remote_port_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_port_uneq_store_unequal()
Test remote_port_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_system_desc_uneq_store_equal()
Test remote_system_desc_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_system_desc_uneq_store_unequal()
Test remote_system_desc_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_system_name_uneq_store_equal()
Test remote_system_name_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

test_remote_system_name_uneq_store_unequal()
Test remote_system_name_uneq_store for same value case.

This tests the case when the value stored is the same as the value passed.

Module contents

networking_cisco.tests.unit.saf.agent.vdp package

Submodules

networking_cisco.tests.unit.saf.agent.vdp.test_dfa_vdp_mgr module

class networking_cisco.tests.unit.saf.agent.vdp.test_dfa_vdp_mgr.**DfaVdpMgrTest**(**args*, ***kwds*)

Bases: neutron.tests.base.BaseTestCase

A test suite to exercise the DfaVdpMgr Class.

setUp()
Setup routine

test_dfa_uplink_restart_invalid_veth()
Test for DFA uplink restart case with invalid veth.

test_dfa_uplink_restart_no_uplink()
Test for DFA uplink restart case with no uplink.

test_dfa_uplink_restart_valid_uplink_veth()
Test for DFA uplink restart case with valid uplink/veth.

test_process_bulk_vm_event_down()
Test Bulk Process event for down status.

test_process_bulk_vm_event_up()
Test Bulk Process event for up status.

test_process_static_uplink_down()
Test routine for static uplink, down case.

test_process_static_uplink_new()
Test routine for static uplink, first time call.

test_process_static_uplink_normal()
Test routine for static uplink, normal case.

test_process_up_uplink_event_lldp_fail()
Test routine when a uplink is detected and lldp is down.

test_process_uplink_event_case1()
Top routine that calls the uplink down case

test_process_uplink_event_case2()
Top routine that calls the uplink detect case

test_process_vm_down_event_uplink_not_rcvd()
Test for VM down event, when uplink event is not received.

test_process_vm_event_fail()
Top routine that calls process VM event fail case

test_process_vm_event_succ()
Top routine that calls process VM event success case

test_process_vm_event_uplink_not_rcvd()
Test for VM event process, when uplink event is not received.

test_topo_disc_cb()
Test the topology discovery CB function.

test_vdp_uplink_proc_down_threshold_exceed()
Test VDP uplink proc for down case, when threshold exceed.

test_vdp_uplink_proc_down_threshold_not_exceed()
Test VDP uplink proc for down case, when threshold not exceed.

test_vdp_uplink_proc_new_uplink()
Test VDP uplink proc for a new uplink case.

test_vdp_uplink_proc_none()
Test VDP uplink proc for none case.

test_vdp_uplink_proc_normal()
Test VDP uplink proc for normal case.

test_vdp_uplink_proc_normal_bond_intf()
Test VDP uplink proc for normal case, for bond interface.

test_vdp_uplink_proc_normal_bulk_vm_not_rcvd()
Test VDP uplink proc for normal case, for bulk VM.

This is for the case when bulk VM notificationis not received.

test_vdp_uplink_proc_normal_static()
Test VDP uplink proc for normal case for static uplink.

test_vdp_vlan_change_cb()
Function to test the VDP VLAN change Callback.

test_vdp_vm_event_dict()
Test for VM event, when input is a VM dict.

test_vdp_vm_event_list()
Test for VM event, when input is a VM dict list.

networking_cisco.tests.unit.saf.agent.vdp.test_lldpad module

class networking_cisco.tests.unit.saf.agent.vdp.test_lldpad.**LldpadDriverTest**(**args*, ***kwds*)

Bases: neutron.tests.base.BaseTestCase

A test suite to exercise the Lldpad Driver.

fill_default_vsi_params()
Mock VSI Params

setUp()
Setup for the test scripts

test_crosscheck_query_mismatch_mac()
Test MAC mismatch in query.

test_crosscheck_query_mismatch_vsiid()
Test VSI ID mismatch in reply.

test_crosscheck_reply_mismatch_mac()
Test MAC mismatch in reply.

test_crosscheck_reply_mismatch_vsiid()
Test VSI ID mismatch in reply.

test_enable_evb()
Top level routine for EVB cfg test

test_enable_lldp()
Tests the routine the enables LLDP cfg

test_filter_query_validity()
Test for filter query validity. Positive case.

test_filter_query_validity_incorrect_filter()
Test for filter query, when there's no filter.

test_filter_query_validity_multiple_filter()
Test for filter query, when there are multiple filters.

test_filter_reply_validity()
Test for filter reply validity. Positive case.

test_filter_reply_validity_incorrect_filter()
Test for filter reply, when there's no filter.

test_filter_reply_validity_multiple_filter()
Test for filter reply, when there are multiple filters.

test_hints_exception()
Test for incorrectly formatted hints.

test_incorrect_hints()
Test for case when there's no hints in reply.

test_init()
Place hlder for init

test_mode_reply_deassoc()
Test for mode reply when deassoc is sent.

test_mode_reply_invalid()
Test for mode reply validity. Invalid case.

test_multiple_hints()
Test for case, when there are multiple hints in query response.

test_nonzero_hints()
Test for non-zero hints.

test_valid_hints()
Test for valid hints case.

test_vdp_failure_reason_invalid()
Test for case that parses the failure reason for invalid case.

test_vdp_failure_reason_invalid_null()
Test for case that parses the failure reason for null case.

test_vdp_failure_reason_valid()
Test for case that parses the failure reason for valid case.

test_vdp_port_down()
Tests the case when a VM goes down

test_vdp_port_up_new_nwk()
Tests the case when a VM comes for a new network

test_vdp_port_up_new_nwk_after_restart()
Tests the case when a VM comes for a new network after restart

test_vdp_port_up_new_nwk_invalid_vlan()
Tests the case when an invalid VLAN is rteturned for a VM that comes up for a new network

test_vdp_port_up_old_nwk()
Tests the case when a VM comes for an existing network

test_vdp_refresh_handler()
Test for VDP refresh handler, with one VDI.

test_vdp_refresh_handler_cb_thresh_exceed()
Test for refresh handler, when callback threshold has exceeded.

test_vdp_refresh_handler_modf_reason()
Test for VDP refresh handler, when fail reason is changed.

test_vdp_refresh_handler_modf_vlan()
Test for VDP refresh handler, when VLAN from VDP is changed.

test_vlan_query_exception()
Test for incorrectly formatted reply in vlan query function.

test_vlan_query_incorrect_filter()
Test for incorrect filter in vlan query function.

test_vlan_query_vsiid_fail()
Test for incorrect vsiid in vlan query function.

test_vlan_reply_invalid()
Test for invalid vlan reply.

networking_cisco.tests.unit.saf.agent.vdp.test_ovs_vdp module

class networking_cisco.tests.unit.saf.agent.vdp.test_ovs_vdp.**OvsVdpTest**(**args*, ***kwds*)

Bases: oslotest.base.BaseTestCase

A test suite to exercise the OvsVdp Class.

setUp()
Setup routine

test_flow_check_handler_both_flows_missing()
Testing the flow check handler for both bridges.

Flow is missing for both bridges.

test_flow_check_handler_ext_flows_missing()
Testing the flow check handler for external bridge.

Flow is missing for external bridge.

test_flow_check_handler_integ_flows_missing()
Testing the flow check handler for integ bridge.

Flow is missing for integration bridge.

test_flow_check_handler_no_flows_missing()
Testing the flow check handler for both bridges.

No flows are missing in both the bridges.

test_populate_cache()
Test the populate cache function.

test_process_init()
Wrapper for the init routine test

test_setup_lldpad_ports()
Test for setup lldpad ports.

test_vdp_port_event()
Routine the calls the other new port and existing port test routines

test_vdp_port_event_down()
Routine the calls the port down test

test_vdp_port_event_down_mismatched_vlans()
Test the case for a vnic port down for a network with mismatch vlan.

This is to test the case when there are more than one vNic for a network with mismatched VLAN's. Stale Flows should be removed and new flows should be added.

test_vdp_port_event_down_no_valid_vlan()
Test the case for a vnic port down for a network with no valid vlan.

This is to test the case when there are more than one vNic for a network with no valid VLAN. Flows should be removed.

test_vdp_port_event_down_valid_vlan()
Test the case for a vnic port down for a network.

This is to test the case when there are more than one vNic for a network with a valid VLAN. Flows should not be removed.

test_vdp_vlan_change_multiple_vnics_norem()
Testing the VDP VLAN change for multiple vnic's.

This is for the case when there are multiple vNics for the same network and for one vNic a VDP VLAN of 0, is returned. So flow should not be deleted.

test_vdp_vlan_change_rem()
Testing the VDP VLAN change for a remove flow case.

test_vdp_vlan_change_rem_add()
Testing the VDP VLAN change for a remove/add flow case.

Module contents

Module contents

networking_cisco.tests.unit.saf.server package

Subpackages

networking_cisco.tests.unit.saf.server.services package

Subpackages

networking_cisco.tests.unit.saf.server.services.firewall package

Subpackages

networking_cisco.tests.unit.saf.server.services.firewall.native package

Subpackages

networking_cisco.tests.unit.saf.server.services.firewall.native.drivers package

Submodules

networking_cisco.tests.unit.saf.server.services.firewall.native.drivers.test_native module

class networking_cisco.tests.unit.saf.server.services.firewall.native.drivers.test_native.**I**
Bases: object

Fake class

classmethod imitate(**others*)

classmethod set_return(*class_name*, *fn_name*, *return_val*)

class networking_cisco.tests.unit.saf.server.services.firewall.native.drivers.test_native.**N**

Bases: neutron.tests.base.BaseTestCase

A test suite to exercise the Native Firewall Driver.

setUp()
Setup for the test scripts

test_create_fw()
Create FW Test

test_delete_fw()
Delete FW Test

test_native_fw_init()
Wrapper for the init

test_nwk_create_notif()
Nwk Create Notif

test_nwk_delete_notif()
Create Notif

test_program_default_gw()
Test for programming the default GW.

test_program_default_gw_fail()
Test for programming the default gw, failure case.

test_program_next_hop()
Test for programming the next hop for out network.

test_program_next_hop_fail()
Test for programming the next hop for out network, failure case.

test_send_in_router_port_msg_down()

test_send_in_router_port_msg_fail_down()

test_send_in_router_port_msg_fail_up()

test_send_in_router_port_msg_up()

test_send_out_router_port_msg_down()

test_send_out_router_port_msg_fail_down()

test_send_out_router_port_msg_fail_up()

test_send_out_router_port_msg_up()

test_update_dcnm_part_static_route()
Test for updating the DCNM partition's static route.

test_update_dcnm_part_static_route_fail()
Test for updating the DCNM partition's static route.

networking_cisco.tests.unit.saf.server.services.firewall.native.drivers.test_phy_asa module

class networking_cisco.tests.unit.saf.server.services.firewall.native.drivers.test_phy_asa
Bases: object

Fake class

classmethod imitate(**others*)

classmethod set_return(*class_name*, *fn_name*, *return_val*)

class networking_cisco.tests.unit.saf.server.services.firewall.native.drivers.test_phy_asa

Bases: neutron.tests.base.BaseTestCase

A test suite to exercise the Phy ASA Driver.

setUp()
Setup for the test scripts

test_build_acl_ip_none()

test_build_acl_port_dst()

test_build_acl_port_not_enabled_dst()

test_build_acl_port_range_dst()

test_build_acl_port_src()

test_build_acl_valid_ip()

test_create_fw()
Create FW Test

test_delete_fw()
Delete FW Test

test_get_ip_address_null()

test_get_ip_address_valid()

test_modify_fw()
Modify FW Test

test_phy_asa_init()
Wrapper for the init

Module contents

Submodules

networking_cisco.tests.unit.saf.server.services.firewall.native.test_fabric_setup_base module

class networking_cisco.tests.unit.saf.server.services.firewall.native.test_fabric_setup_bas

Bases: neutron.tests.base.BaseTestCase

A test suite to exercise the Fabric setup Base.

setUp()
Setup for the test scripts

test_clear_dcnm_in_part()
Clear DCNM IN partition service node address Test.

test_clear_dcnm_out_part()
Clear DCNM OUT partition service node address Test.

test_create_dcnm_in_nwk()
Create IN Network Test.

This function relies on the state information filled by previous functions. So, rather than starting fresh, we shall populate the FW DB. This is equivalent to restarting the enabler server and it reading the DB and populating the local cache

test_create_dcnm_in_part_update()
DCNM Update IN Partition Test.

test_create_dcnm_out_nwk()
Create OUT Network Test.

This function relies on the state information filled by previous functions. So, rather than starting fresh, we shall populate the FW DB. This is equivalent to restarting the enabler server and it reading the DB and populating the local cache.

test_create_dcnm_out_part()
DCNM Out Part Create Test.

test_create_dcnm_out_part_update()
DCNM Update OUT Partition Test.

test_create_in_nwk()
Create IN Network.

test_create_in_nwk_fail()
Create IN Network Fail.

The Openstack create network helper function is mocked to return a failure.

test_create_os_dummy_rtr()
Create Dummy Router Test.

test_create_os_dummy_rtr_fail()
Create Dummy Router Fail Test.

The Openstack add interface to router helper function is mocked to return a failure.

test_create_os_dummy_rtr_virt()
Create Dummy Router Virt Test.

test_create_out_nwk()
Create Out Network Test.

test_create_out_nwk_fail()
Create OUT Network Fail.

The Openstack create network helper function is mocked to return a failure.

test_delete_dcnm_in_nwk()
Delete DCNM In Network Test.

test_delete_dcnm_out_nwk()
Delete DCNM Out Network Test.

test_delete_dcnm_out_part()
Delete DCNM OUT partition address Test.

test_delete_in_nwk()
Delete IN Network Test.

test_delete_in_nwk_fail()
Delete IN Network Failure Test.

test_delete_os_dummy_rtr()
Delete Dummy Router Test.

test_delete_os_dummy_rtr_fail()
Delete Dummy Router Fail Test.

The Openstack delete interface to router helper function is mocked to return a failure.

test_delete_out_nwk()
Delete OUT Network Test.

test_delete_out_nwk_fail()
Delete OUT Network Failure Test.

test_fabric_base_init()
Wrapper for the init

class networking_cisco.tests.unit.saf.server.services.firewall.native.test_fabric_setup_bas
Bases: object

Fake class.

classmethod imitate(**others*)

classmethod set_return(*class_name*, *fn_name*, *return_val*)

networking_cisco.tests.unit.saf.server.services.firewall.native.test_fw_mgr module

class networking_cisco.tests.unit.saf.server.services.firewall.native.test_fw_mgr.**FakeClass**
Bases: object

Fake class.

classmethod imitate(**others*)

classmethod set_return(*class_name*, *fn_name*, *return_val*)

class networking_cisco.tests.unit.saf.server.services.firewall.native.test_fw_mgr.**FwMgrTest**

Bases: neutron.tests.base.BaseTestCase

A test suite to exercise the FW Mgr.

setUp()
Setup for the test scripts.

test_fw_create()
Test FW create.

test_fw_create_device_error()
Test FW create.

test_fw_create_fabric_error()
Test FW create.

test_fw_delete()
Test FW delete.

test_fw_delete_dev_error()
Test FW delete.

test_fw_delete_fab_error()
Test FW delete.

test_fw_mgr_init()
Wrapper for the init.

test_fw_policy_create()
Test FW policy create.

test_fw_rule_create()
Test FW rule create.

Module contents

Module contents

Module contents

Submodules

networking_cisco.tests.unit.saf.server.test_cisco_dfa_rest module

class networking_cisco.tests.unit.saf.server.test_cisco_dfa_rest.**TestCiscoDFAClient**(**args*, ***kwds*)

Bases: oslotest.base.BaseTestCase

Test cases for DFARESTClient.

setUp()

test_create_network()
Test create network.

test_create_project()
Test create project.

test_delete_network()
Test delete network.

test_delete_project()
Test delete tenant.

test_get_segmentid_range()
Test get segment ID range.

test_http_verify_protocol()
Test login test using http.

test_https_verify_protocol()
Test login test using https.

test_set_segmentid_range()
Test set segment ID range.

test_update_segmentid_range()
Test set segment ID range.

class networking_cisco.tests.unit.saf.server.test_cisco_dfa_rest.**TestNetwork**

Bases: object

config_profile = 'defaultL2ConfigProfile'

name = 'cisco_test_network'

segmentation_id = 123456

networking_cisco.tests.unit.saf.server.test_dfa_server module

class networking_cisco.tests.unit.saf.server.test_dfa_server.**FakeClass**

Bases: object

Fake class

classmethod imitate(**others*)

class networking_cisco.tests.unit.saf.server.test_dfa_server.**FakeProject**(*proj_id*, *name*, *dci_id*, *desc*)

Bases: object

Fake Project class.

class networking_cisco.tests.unit.saf.server.test_dfa_server.**TestDFAServer**(**args*, ***kwds*)

Bases: oslotest.base.BaseTestCase

Test cases for DFA Server class.

setUp()

test_add_dhcp_port()
Test add dhcp port

test_add_lbaas_port()

test_correct_dhcp_ports()
Test case for port delete event.

test_dcnm_network_create_event()
Test case for DCNM network create event.

test_dcnm_network_delete_event()
Test case for DCNM network delete event.

test_delete_vm_funciton()
Test case for port delete event.

test_is_mand_arg_present_false()
Test the is_mand_arg_present function for False case.

test_is_mand_arg_present_true()
Test the is_mand_arg_present function for True case.

test_network_create_func()
Test case for network create event.

test_network_delete_event()
Test case for network delete event.

test_port_create_event()
Test case for port create event.

test_port_update_event()
Test case for port update event.

test_project_create_func()
Test case for project create event.

test_project_delete_event()
Test case for project delete event.

test_project_update_event()
Test case for project update event.

test_save_topo_disc_params_exist_mand()
Test the save_topo_disc_params function for exist, mandatory case.

This is for the case when config is already present in the DB and mandatory TLV's are present in the new config. This is the uppdate case.

test_save_topo_disc_params_exist_nomand()
Test the save_topo_disc_* function for exist, non-mandatory case.

This is for the case when config is already present in the DB and mandatory TLV's are not present in the new config. This is the delete case.

test_save_topo_disc_params_none_nonexist_nonmand()
Test the save_topo_disc_* func for none, non-exist, non-mand case.

This is for the case when config is not present in the DB and mandatory TLV's are not present in the new config. The output returned is None. This is the no-op case.

test_save_topo_disc_params_nonexist_mand()
Test the save_topo_disc_* function for non-exist, mandatory case.

This is for the case when config is not present in the DB and mandatory TLV's are present in the new config. This is the add case.

test_save_topo_disc_params_nonexist_nonmand()
Test the save_topo_disc_* function for non-exist, non-mand case.

This is for the case when config is not present in the DB and mandatory TLV's are not present in the new config. This is the no-op case.

test_send_vm_info()
Test send_send_vm_info

test_subnet_create_event()
Test case for subnet create event.

test_update_project_info_cache()
Test case for update project info.

Module contents

Module contents

networking_cisco.tests.unit.services package

Subpackages

networking_cisco.tests.unit.services.trunk package

Submodules

networking_cisco.tests.unit.services.trunk.test_nexus_trunk module

class networking_cisco.tests.unit.services.trunk.test_nexus_trunk.**TestNexusTrunkDriver**(**args*, ***kw*

Bases: neutron.tests.unit.testlib_api.SqlTestCase

setUp()

test_is_loaded()

class networking_cisco.tests.unit.services.trunk.test_nexus_trunk.**TestNexusTrunkHandler**(*ar
**k

Bases: neutron.tests.unit.db.test_db_base_plugin_v2.NeutronDbPluginV2TestCase

setUp()

test_subport_postcommit_baremetal_after_create()

test_subport_postcommit_baremetal_after_delete()

test_subport_postcommit_baremetal_unsupported_event()

test_subport_postcommit_vm()

test_subport_postcommit_vm_no_hostid()

test_subport_postcommit_vm_unsupported_event()

test_trunk_update_postcommit_baremetal_active_state()

test_trunk_update_postcommit_baremetal_down_state()

test_trunk_update_postcommit_vm()

Module contents

Module contents

Module contents

Submodules

networking_cisco.tests.base module

class networking_cisco.tests.base.**TestCase**(**args*, ***kwds*)

Bases: oslotest.base.BaseTestCase

Test case base class for all unit tests.

networking_cisco.tests.base.**load_config_file**(*string*)

networking_cisco.tests.test_compatibility module

class networking_cisco.tests.test_compatibility.**MySQLTestCase**(**args*, ***kwds*)

Bases: neutron.tests.unit.testlib_api.MySQLTestCaseMixin, neutron.tests.unit.testlib_api.SqlTestCaseLight

networking_cisco.tests.test_networking_cisco module

test_networking_cisco

Tests for *networking_cisco* module.

class networking_cisco.tests.test_networking_cisco.**TestNetworking_cisco**(**args*, ***kwds*)

Bases: *networking_cisco.tests.base.TestCase*

test_something()

Module contents

Module contents

3.5 Reference Guides

3.5.1 ASR1000 L3 Router Service Plugin Overview and Architecture

The ASR1k L3 router service plugin (L3P) represents each Neutron router as a virtual routing and forwarding table (VRF) to ensure isolation. Each neutron router port maps to a VLAN sub-interface in the ASR1k. These sub-interfaces can either reside on ethernet interfaces or port-channel interfaces.

When a neutron router is attached to a subnet on an internal network the corresponding VLAN sub-interface that is created is placed in the VRF of the router. Hence the VRF effectively monopolizes that sub-interface. This is the reason why a particular neutron network can only be attached to a single neutron router.

The gateway port of a neutron router is handled differently as external networks typically need to be shared among many tenant routers. For this reason VLAN sub-interfaces for external networks are placed in the *global* VRF. The default gateway inside the VRF of the tenant routers point to the IP address of the *upstream* router on the external network *via* the VLAN sub-interface for the external network in the global VRF.

For the dynamic source NAT that is normally performed by tenant neutron routers with gateway set, the IP address of the tenant router's *gateway port* is used. If a floating ip has been created and associated with a neutron port on an internal network/subnet, the static source NAT for that floating ip takes precedence of the dynamic source NAT.

3.5.2 Nexus Mechanism Driver Overview and Architecture

Introduction

The ML2 Nexus driver adds and removes trunk vlans on the Cisco Nexus switch for both ethernet interfaces and port-channel interfaces. It also supports configuration of VXLAN Overlay, baremetal deployments (VLAN configurations only), and performs configuration replay on switch recovery.

This user guide describes the activities the Nexus driver performs to create and remove VLANS on receipt of OpenStack ML2 baremetal and non-baremetal port events.

VLAN Creation

When VMs are created or when subnets are created with dhcp is enabled, port events are received by the Nexus driver. If there are switch credentials defined by the administrator for the event, then the Nexus driver will process the event.

The basic Nexus configuration actions taken by the Nexus driver are as follows:

1. Configure the provided VLAN on the Nexus device,
2. Configure the interface as **switchport mode trunk** and **switchport trunk allowed vlan none** if no trunk vlans have been configured manually by the user.

3. Add a trunk vlan onto the specified interface using the interface CLI **switchport trunk allowed vlan add <vlanid>**.
4. For baremetal port events, if the port is the *parent* port to an OpenStack trunking configuration or not a participant of an OpenStack trunking configuration, the VLAN is also configured as *native* using the CLI **switchport trunk native vlan <vlanid>**.

Both normal VM port events and baremetal port events are supported by the Nexus Driver for VLAN creation. They can co-exist at the same time.

In the case of non-baremetal port events, the Nexus driver uses the host name from the port event to identify a switch and interface(s) to configure. The vlan used to configure the interface also comes from the port event. The administrator configures the host to interface mapping as well as switch credentials in the ML2 Nexus Driver switch configuration section of the neutron config file. (ref: section header `ml2_mech_cisco_nexus` of config file shown in the *Nexus Mechanism Driver Administration Guide*.)

In the case of baremetal port events, the switch and interface mapping are contained in the event itself. The Nexus driver learns the host to interface mapping by using dns name as the host name. If dns is not enabled, then the host name defaults to `reserved_port`. Even though the administrator does not configure this mapping, the switch credentials must be configured for baremetal events. This configuration aides in determining whether the baremetal event is for the Nexus Driver.

If there are multiple ethernet interfaces defined in the baremetal event, this implies it is a port-channel. When the Nexus driver detects multiple interfaces, it next determines whether the interfaces are already configured as members of a port-channel. If not, it creates a new port-channel interface and adds the ethernet interfaces as members. In more detail, it will do the following:

1. Allocate a port-channel id from `vpc_pool` configured by administrator ref: `vpc_pool` variable in admin and config guides).
2. Create the port-channel which includes **switchport mode trunk**, **switchport trunk allowed vlan none**, and **vpc-id x**.
3. Apply either user customized port-channel config provided by administrator OR the default config **spanning-tree port type edge trunk** and **no lacp suspend-individual** (refer to `intfcfg_portchannel` variable in the Nexus *administration* and *configuration* guides).
4. Apply **channel-group <vpcid> force mode-active** to the ethernet interfaces to make each interface a member of the port-channel.

Regardless whether the port-channel is learned or created, the trunk vlans are applied to the port-channel and inherited by ethernet interfaces.

VLAN Removal

When a VM is removed or a subnet is removed and dhcp is enabled, a delete port-event is received by the Nexus driver. If the port exists in the Nexus driver's port data base, the driver will remove it from the data base as well as remove the trunk vlan on the Nexus device.

To remove the trunk vlan from interface on the Nexus switch, it sends **switchport trunk allowed vlan remove <vlanid>** and possibly **no switchport trunk native vlan <vlanid>** if it was sent during vlan creation. The driver then checks if the vlan is used on any other interfaces. If not, it will remove the vlan from the Nexus switch as well by issuing **no vlan <vlanid>**.

If a port-channel was previously created for baremetal port events as described in *VLAN Creation* and if there are no more port-bindings referencing the created port-channel, the Nexus Driver will do as follows:

- The ethernet interfaces will be removed as members to the port-channel by issuing :command:' no channel-group ' on each participating Nexus Switch interface,

- The port-channel will be completely removed from the Nexus Switch(s) by issuing **`no port-channel <id>`** on each participating switch,
- And the port-channel/vpc id released back into the Nexus driver vpc-id pool.

VXLAN Overlay Creation

VXLAN Overlay creation does similar basic vlan trunk config as described in the *VLAN Creation* section. Prior to doing vlan trunk config, the VLAN is mapped to a VXLAN Network Identifier (VNI) and applied to the NVE (network virtualization edge) interface. Specifically, the steps done for the user is as follows:

- Create a one-to-one mapping by creating a mulitcast IP address and associating it with a VXLAN Network ID. Apply this configuration to the NVE interface:

```
int nve1
    member vni <vni-id> mcast-group <mcast-addr>
```

- Associate the VNI segment to the VLAN segment. The configuration applied is as follows:

```
vlan <vlanid>
  vn-segment <vni-id>
```

Configuration VXLAN VNI ranges and multicast groups is done beneath the section header `ml2_type_nexus_vxlan` of the configuration file. See the *Nexus Mechanism Driver Administration Guide* for more details.

VXLAN Overlay Removal

VXLAN Overlay removal does vlan trunk removal as described in *VLAN Removal* section. Additionally, it removes the vni member from the nve interface as well as vlan segment if there are no other ports referencing it.

Configuration Replay

If the Nexus MD discovers the Nexus switch is no longer reachable, all known configuration for this switch is replayed once communication is restored. The order of the events are performed differently than described in *VLAN Creation* for efficiency reasons. This order is as follows:

1. All known interfaces are initialized with **`switchport mode trunk`** and **`switchport trunk allowed vlan none`** if there are no trunking vlans already configured.
2. For VXLAN, set **`member vni <vni-id> mcast-group <mcast-addr>`** beneath the nve interface.
3. For each interface, a lists of VLANS are sent to the Nexus switch as a single request using the configuration **`switchport trunk allowed vlan add <multiple-vlanids>`**.
4. Following this, batches of vlans made active. For VXLAN, this will also include the **`vn-segment <vni>`** configuration.

3.5.3 UCSM Mechanism Driver Overview and Architecture

Introduction

The ML2 UCSM driver translates neutron ML2 port and network configuration into configuration for Fabric Interconnects and vNICs on the UCS Servers. This driver supports configuration of neutron virtual and SR-IOV ports on VLAN networks. It communicates with the UCSM via the Cisco UCS Python SDK (version 0.8.2).

Overview

The UCSM driver runs as part of the neutron server process on the OpenStack controller nodes and has no component running on any of the compute nodes. In an OpenStack cloud consisting of a mixiture of servers from different vendors, enabling the UCSM driver will result in configuration of only the UCS servers controlled by UCS Managers. The other servers will not be affected. Once the UCSM driver is initialized with its configuration, any changes to the configuration will not take effect until the neutron server process is restarted.

Neutron virtual port support

The UCSM driver configures UCS servers and Fabric Interconnects (FIs) in response to a neutron virtual port request during VM creation and deletion. The UCSM driver when enabled, configures the FI's outgoing trunk port to carry VLAN traffic for that VM. It also configures the server's ethernet ports to allow the VM's VLAN traffic by manipulating the server's Service Profile (or Template). The driver is also responsible for removing this configuration during VM port deletion.

SR-IOV port support

The UCSM driver supports configuration of SR-IOV ports created on Cisco and and Intel NICs available on UCS Servers. VMs with SR-IOV ports enjoy greater network performance because these ports bypass the virtual switch on the compute host's kernel and send packets directly to the Fabric Interconnect.

When the SR-IOV ports being configured are on Cisco NICs, the driver creates Port Profiles (PPs) on the UCS Manager and associates it with the next available VF. The driver is also responsible for deleting the PPs when all VMs using it no longer exist.

UCS Manager Template configuration support

The UCSM driver supports configuration of Service Profile Templates and vNIC Templates on the UCS Manager. These UCS Manager constructs allow the driver to simultaneously modify configuration on several UCS Servers.

Service Profile Templates can be used on the UCS Manager to configure several UCS Servers with identical configuration. When UCS Servers in the OpenStack setup are controlled by Service Profile Templates, this information needs to be provided to the UCSM driver so that it can directly update the Service Profile Template.

Similarly, vNIC Templates are UCS Manager constructs that can be used to configure several vNICs on different UCS Servers with similar configuration. This can be used to manage neutron provider network configuration by effectively using a vNIC Template to configure all vNICs that are connected to the same physical network. When this information is provided to the UCSM driver, it will modify the vNIC Templates with the VLAN configuraton of the corresponding neutron provider network.

3.5.4 Full Release Notes for networking-cisco

7.0.0

Prelude

In this release, clean-up of Nexus configuration was performed by removing obsolete features. For users of tripleo and puppet-neutron repos, you must make sure you are using the correct versions of the repos that include the changes from bugs https://bugs.launchpad.net/tripleo/+bug/1793381 and https://bugs.launchpad.net/puppet-neutron/+bug/1793379.

New Features

- Adds support for the Rocky release of OpenStack

Upgrade Notes

- Nexus/UCSM: Deprecate old ML2 documentation file

  In older versions of networking-cisco, the file `etc/neutron/plugins/ml2/ml2_conf_cisco.ini` contains all ML2 configuration documentation for Nexus and UCSM. This has been replaced with a much improved set of documentation which can be found at http://networking-cisco.readthedocs.io. For details of changes, refer to https://bugs.launchpad.net/networking-cisco/+bug/1780445

- Nexus: Verify https certificates by default

  See the Security release notes for more details.

- Nexus: Remove deprecated format of host interface mapping config

  The host to interface mapping configuration was deprecated and replaced in release 6.1.0 (see previous release notes). The deprecated configuration is now removed so the replacement configuration must take its place. The following demonstrates the old versus new configuration.

  `Old:` *hostname_abc=ethernet:1/19,1/20*

  `New:` *host_ports_mapping=hostname_abc:[ethernet:1/19,1/20]*

  Refer to https://bugs.launchpad.net/networking-cisco/+bug/1771672 for implementation details.

- Nexus: Remove deprecated intfcfg.portchannel configuration

  The intfcfg.portchannel configuration was deprecated and replaced in release 5.4.0 (see previous release notes). The deprecated configuration is now removed so the replacement configuration intfcfg_portchannel must take its place.

- Nexus: Remove Nexus ncclient configuration driver

  The `ncclient` Driver used to configure Nexus was deprecated in the `Cisco 5.5.0` release. It is replaced by the current default RESTAPI Driver. The ncclient driver is now removed along with related configuration options which include `ssh_port`, `persistent_switch_config`, `never_cache_ssh_connection`, `host_key_checks`, and `nexus_driver`. For the RESTAPI driver to work, the :command:'feature nxapi' must be preconfigured on the Nexus device and the correct version of NX-OS must be installed. Refer to *Nexus Installation Guide*. for more prerequisite details. For implementation details, refer to https://bugs.launchpad.net/networking-cisco/+bug/1758042

Deprecation Notes

- Nexus/UCSM: Deprecate old ML2 documentation file

  The documentation file `etc/neutron/plugins/ml2/ml2_conf_cisco.ini` has been replaced with http://networking-cisco.readthedocs.io. Reference to this new location is placed in `ml2_conf_cisco.ini`. This file will be removed entirely in a later release and is no longer copied during setup operations. For details of changes, refer to https://bugs.launchpad.net/networking-cisco/+bug/1780445

Security Issues

- Nexus: Verify https certificates by default

  In release 5.4.0, the ability to verify https certificates was implemented. The default was originally set to False for insecure https connections to allow time to acquire certificates. In this release, the default is changed to True so certificates will be verified for secured connections.

  See release notes for release 5.4.0 for details on this feature and https://bugs.launchpad.net/networking-cisco/+bug/1778275 for implementation details.

Bug Fixes

- Fixes a bug in the networking-cisco migrations when run against MariaDB which prevent the *subnet_id* field being added as a primary key due to it previously being added as a foreign key. See: https://bugs.launchpad.net/networking-cisco/+bug/1791121

6.1.0

New Features

- Adds support for the OpenStack Neutron queens release.

Deprecation Notes

- Nexus: Host Mapping Configuration being replaced

  The host mapping configuration beneath the header *ml2_mech_cisco_nexus* currently does not have a static config option name. This can lead to some issues where even typographical errors can be interpreted as a host mapping config. The config option *host_ports_mapping* has been introduced to resolve this shortcoming. The following demonstrates the before and after config change.

  `Before:` *hostname_abc=ethernet:1/19*, `After:` *host_ports_mapping=hostname_abc:[ethernet:1/19]*

Bug Fixes

- Nexus: Do not raise exception during update_postcommit when port not found

  Occasionally spurious updates are seen in parallel with deletes for same vlan. In this window an update can be received after the port binding is removed. This change reports a warning message instead of raising an exception keeping it consistent with other ML2 driver behavior. This circumstance will more likely be seen when there are multiple neutron threads and controllers.

- Nexus: Neutron trunking feature not supported in Newton

  Introduced a fix to prevent an error from being generated when using openstack newton or below branches with baremetal configurations. The error message seen is "TypeError: get_object() got an unexpected keyword" "argument 'port_id'". For implementation details, refer to https://review.openstack.org/#/c/542877.

6.0.0

Upgrade Notes

- Cisco UCSM: The ucsmsdk is now the default replacing the UcsSdk

  The ucsmsdk is now the default package for interacting with UCSM. Use of the now deprecated UcsSdk will still work if the ucsmsdk is not installed. However, all new features will be developed using the ucsmsdk so users are encouraged to upgrade.

- All code for CSR1kv-based routing has been removed from networking-cisco. The code was removed in commit 917480566afa2b40dc382bc4f535d173bad7736d.
- All Nexus 1000v driver code has been removed from networking-cisco, and all n1kv related tables have been dropped. The code was removed in commit 0730ec9e6b76b3c1e75082e9dd1af55c9faeb34c
- NCS: Remove support for Network Control System (NCS). The code was removed in commit 31e4880299d04ceb399aa38097fc5f2b26e30ab1

Deprecation Notes

- Cisco UCSM: Use of the UcsSdk has been deprecated for removal

  Use of of the UcsSdk will be removed in a future release. It has been replaced by the ucsmsdk. While use of the UcsSdk will continue to work until its complete removal, no new features will be added so users are encouraged to upgrade.

5.5.2

Bug Fixes

- Nexus: DBDuplicateEntry error from interface mapping db table with multi-controllers

  When there are multiple controllers running, they could simultaneously attempt to initialize the Nexus host interface mapping db table using the user's static host mapping configuration. This could result in a *DBDuplicateEntry* exception. This type of error is seen with static user configured hosts but not ironic learned hosts. Refer to *DBDuplicate Entry - Failed Insert into cisco_ml2_nexus_host_interface_mapping* for error message details and corrective action. For implementation details, refer to https://bugs.launchpad.net/networking-cisco/+bug/1743573.

5.5.0

New Features

- Cisco UCSM: vNIC and Service Profile Template support for Single UCSM

  This feature allows a cloud admin to take advantage of the Service Profile Template and vNIC Template configuration options available on the UCS Manager. The UCSM driver can be provided with the Service Profile or the Service Profile Template configuration, and it will handle the configuration of the UCS Servers accordingly. The vNIC Templates can be used to configure several vNIC on different UCS Servers, all connected to the same physical network. The driver will handle configuration of the appropriate vNIC Template with the VLAN configuration associated with the corresponding neutron provider network.

5.4.0

Security Issues

- Nexus: https certification now supported by RESTAPI Client

  The Nexus RESTAPI Client now sends requests using https instead of http which results in communication with the Nexus to be encrypted. Certificate verification can also be performed. A new configuration option 'https_verify' controls this latter capability. When set to False, the communication path is insecure making it vulnerable to man-in-the-middle attacks. Initially, the default for 'https_verify' is set to False but will change to True in the 'Cisco 6.0.0' release. If a certificate is already available and configured on the Nexus device, it is highly recommended to set this options to True in the neutron start-up configuration file.

  For testing or lab purposes, a temporary local certificate can be generated and the certificate filename can be provided in the configuration option 'https_local_certificate'. This depends on the Nexus device being configured with the local key and certificate file.

  Both configuration options are available for every Nexus switch configured. Refer to the *Nexus Configuration Reference* for more details on these options as well as https://bugs.launchpad.net/networking-cisco/+bug/1735295

- Nexus: Obfuscate password

  In log output, obfuscate Nexus Switch password provided in Neutron Start-up configuration.

- Cisco UCSM: Add config to control SSL certificate checks

  This feature allows a cloud admin to disable SSL certificate checking when the UCSM driver connects to the UCS Managers provided in its configuration. SSL certificate checking is ON by default and setting the `ucsm_https_verify` configuration parameter to `False` turns it OFF. Turning it OFF makes the connection insecure and vulnerable to man-in-the-middle attacks.

Bug Fixes

- Nexus: DBDuplicateEntry error seen with Nexus interface mapping database

  Introduced a fix to resolve the issue when the same port-channel is configured for multiple hosts beneath the same switch, a *DBDuplicateEntry* error is seen. This type of configuration is seen with static configurations only and not ironic. Refer to *DBDuplicate Entry - Failed Insert into cisco_ml2_nexus_host_interface_mapping* for sample config, more error message details, and corrective action. For implementation details, refer to https://bugs.launchpad.net/networking-cisco/+bug/1735540.

- Nexus: Remove '.' from configuration variable names

  When testing new configuration variables with puppet and tripleo, the use of dot '.' in configuration variable name fails. There is only one such variable which is *intfcfg.portchannel*. It is replaced with *intfcfg_portchannel*.

Other Notes

- Nexus: RESTAPI Client Scaling Improvement

  To improve performance, the same cookie will be used in requests until it expires and the Nexus device returns a status_code of 403. When this is detected, an attempt to refresh the cookie occurs and upon successful receipt of a new cookie the request that originally failed will be resent. For more details, refer to https://bugs.launchpad.net/networking-cisco/+bug/1735295

5.3.0

New Features

- Nexus: Improved port-channel support with baremetal events

  When there are multiple ethernet interfaces in the baremetal's neutron port event, the Nexus driver determines whether the interfaces are already configured as members of a port-channel. If not, it creates a new port-channel interface and adds the ethernet interfaces as members. In either case, trunk vlans are applied to the port-channel. For this to be successful, a new configuration variable 'vpc_pool' must be defined with a pool of vpc ids for each switch. This must be defined beneath the section header [ml2_mech_cisco_nexus:<switch-ip-address>]. A vpc id common between participating switches will be selected. To get more details on defining this variable, refer to networking-cisco repo, file: etc/neutron/plugins/ml2/ml2_conf_cisco.ini For implementation details on automated port-channel creation, refer to https://bugs.launchpad.net/networking-cisco/+bug/1707286 and https://bugs.launchpad.net/networking-cisco/+bug/1691822 and https://bugs.launchpad.net/networking-cisco/+bug/1705294

- Nexus: User customizable port-channels for baremetal interfaces

  When the Nexus driver creates port-channels for baremetal events, an additional capability was provided to allow the user to custom configure port-channels that are created. This is done by way of the config variable 'intfcfg.portchannel' beneath each switch's section header [ml2_mech_cisco_nexus:<switch-ip-address>]. Nexus CLI commands are defined in this variable unique for each switch and sent while creating the port-channel. For details, refer to https://bugs.launchpad.net/networking-cisco/+bug/1706965

- Nexus: Neutron Trunk Support

  The Nexus mechanism driver support of the neutron trunk feature (https://docs.openstack.org/ocata/networking-guide/config-trunking.html) is to create and trunk on the Nexus switch the trunk subport's network VLAN(s) configured under the neutron trunk parent port.

- Nexus: Provider Network Limited Operations

  The Openstack administrator may want to control how the neutron port events program the Nexus switch for provider networks. Two configuration variables have been introduced to suppress vlan creation and the vlan trunk port setting on the Nexus switch for provider networks. These variables, 'provider_vlan_auto_create' and 'provider_vlan_auto_trunk', are defined under the [ml2_cisco] section header.

- Cisco UCSM: Auto detection of Compute hosts

  This feature allows a cloud admin to expand the size of the Openstack cloud dynamically by adding more compute hosts to an existing UCS Manager. The cloud admin can now add the hostname of this new compute host to the "Name" field of its Service Profile on the UCSM. Then when a VM is scheduled on this compute host, the Cisco UCSM ML2 mechanism driver goes through all the Service Profiles of all the UCSMs known to it to figure out the UCSM and the Service Profile associated with that host. After learning the UCSM and Service Profile, the mechanism driver saves this information for future operations. Note that this method cannot be used to add more Controller nodes to the cloud.

Upgrade Notes

- Nexus: Add host to switch/interface mapping database table

  A new database table for host to interface mapping is added for baremetal deployments. The administrator must perform a database migration to incorporate this upgrade. The new database table name is 'cisco_ml2_nexus_host_interface_mapping'. For more details, refer to https://bugs.launchpad.net/networking-cisco/+bug/1691194

- Nexus: Set RESTAPI driver as default replacing ncclient driver

  The Nexus 9K handles the RESTAPI events more efficiently and without session limitations. It is now the default and will be the only choice in 'Cisco 7.0.0' release. This may require the administrator to upgrade the Nexus operating system. If necessary, use 'nexus_driver=ncclient' to temporarily go back to original default driver; however, some enhancements may not be available when using this driver. For details, refer to https://bugs.launchpad.net/networking-cisco/+bug/1705036

- Nexus: Set default for Configuration Replay to enabled

  Configuration replay is now enabled by default by setting the variable 'switch_heartbeat_time' to 30 seconds (defined under the [ml2_cisco] section header). If the administrator does not want this feature enabled, set this variable to 0. When enabled, the nexus driver checks each switch for connectivity and will restore the configuration known to the driver if a switch recovers from failure. For details, refer to https://bugs.launchpad.net/networking-cisco/+bug/1712090

- Nexus: New vpc id allocation database table

  To implement the vpc id pool for automated port-channel creation with baremetal deployments, a new database table was created so a database migration is needed to incorporate the new vpc id table. The new database table name is 'cisco_ml2_nexus_vpc_alloc'. For more details, refer to https://bugs.launchpad.net/networking-cisco/+bug/1707286 and https://bugs.launchpad.net/networking-cisco/+bug/1691822 and https://bugs.launchpad.net/networking-cisco/+bug/1705294

Deprecation Notes

- Nexus: The ncclient/ssh config driver has been deprecated for removal

  Use of ncclient/ssh_driver will be removed in the 'Cisco 7.0.0' release. It will be replaced by the RESTAPI Driver. Some configuration options are also deprecated for removal since they relate only to the ncclient driver. These include 'persistent_switch_config', 'never_cache_ssh_connection', 'host_key_checks', and 'nexus_driver'. For details, refer to https://bugs.launchpad.net/networking-cisco/+bug/1705036

Bug Fixes

- Nexus: Eliminate warning message for 'MultiConfigParser' from Nexus ML2 Plugin

  The 'MultiConfigParser' class is deprecated as seen by warnings in the neutron log file. Refer to https://bugs.launchpad.net/networking-cisco/+bug/1712853 for details.

- ASR1k: Fix greenpool.py traceback in Ocata

  The ASR1k plugin was wrapping neutron and plugin DB operations in common transactions that was generating a lot of strange tracebacks in the neutron server logs. This commit removes the transaction wrapper to make the operations more independent of each other, eliminating the tracebacks entirely.

- Nexus: Eliminate warning message for 'neutron.db.api.get_session'

  The 'neutron.db.api.get_session' API is deprecated as seen by warnings in the neutron log file so it is replaced. For details, refer to https://bugs.launchpad.net/networking-cisco/+bug/1713498

Other Notes

- Nexus: Remove unused configuration variables

The configuration variables 'svi_round_robin', 'provider_vlan_name_prefix', and 'vlan_name_prefix' are no longer used by the nexus driver and can be removed. For further details, refer to https://bugs.launchpad.net/networking-cisco/+bug/1712118

Python Module Index

n

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

UcsmDbModel (class in networking_cisco.ml2_drivers.ucsm.ucsm_db), 159
UcsmDisconnectFailed, 158
UCSTemplate (class in networking_cisco.ml2_drivers.ucsm.config), 157
unassign_hosting_device_from_cfg_agent() (networking_cisco.plugins.cisco.db.scheduler.cfg_agentschedulers_db.CfgAgentSchedulerDbMixin method), 205
unassign_hosting_device_from_cfg_agent() (networking_cisco.plugins.cisco.extensions.ciscocfgagentscheduler.CfgAgentSchedulerPluginBase method), 216
uncfg_intf() (networking_cisco.apps.saf.agent.topo_disc.topo_disc.TopoDisc method), 92
UnexpectedError, 192
unprovision_vdp_overlay_networks() (networking_cisco.apps.saf.agent.vdp.ovs_vdp.OVSNeutronVdp method), 102
unregister() (networking_cisco.plugins.cisco.cfg_agent.service_helpers.service_helper.ServiceHelperBase method), 183
unschedule_router() (networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.L3RouterHostingDeviceBaseScheduler method), 233
unschedule_router() (networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.L3RouterHostingDeviceLongestRunningScheduler method), 234
unschedule_router() (networking_cisco.plugins.cisco.l3.schedulers.l3_router_hosting_device_scheduler.L3RouterHostingDeviceRandomScheduler method), 234
unschedule_router() (networking_cisco.plugins.cisco.l3.schedulers.noop_l3_router_hosting_device_scheduler.NoopL3RouterHostingDeviceScheduler method), 234
unschedule_router_postcommit() (networking_cisco.plugins.cisco.l3.drivers.asr1k.asr1k_routertype_driver.ASR1kL3RouterDriver method), 224
unschedule_router_postcommit() (networking_cisco.plugins.cisco.l3.drivers.L3RouterBaseDriver method), 229
unschedule_router_postcommit() (networking_cisco.plugins.cisco.l3.drivers.noop_routertype_driver.NoopL3RouterDriver method), 225
unschedule_router_precommit() (networking_cisco.plugins.cisco.l3.drivers.asr1k.asr1k_routertype_driver.ASR1kL3RouterDriver method), 224
unschedule_router_precommit() (networking_cisco.plugins.cisco.l3.drivers.L3RouterBaseDriver method), 229
unschedule_router_precommit() (networking_cisco.plugins.cisco.l3.drivers.noop_routertype_driver.NoopL3RouterDriver method), 225
update (networking_cisco.tests.unit.ml2_drivers.nexus.test_trunk.TestTrunk attribute), 318
update() (networking_cisco.plugins.cisco.cfg_agent.service_helpers.service_helper.ServiceHelperBase method), 183
update() (networking_cisco.plugins.cisco.cpnr.model.ClientEntry
method), 196
update() (networking_cisco.plugins.cisco.cpnr.model.ForwardZone method), 196
update() (networking_cisco.plugins.cisco.cpnr.model.Host method), 196
update() (networking_cisco.plugins.cisco.cpnr.model.Network method), 196
update() (networking_cisco.plugins.cisco.cpnr.model.ReverseZone method), 197
update() (networking_cisco.plugins.cisco.cpnr.model.Scope method), 197
update() (networking_cisco.plugins.cisco.cpnr.model.View method), 197
update() (networking_cisco.plugins.cisco.cpnr.model.Vpn method), 197
update() (networking_cisco.services.trunk.trunkstubs.TrunkObject class method), 237
update_agent_configurations() (networking_cisco.apps.saf.db.dfa_db_models.DfaDBMixin method), 110
update_agent_db() (networking_cisco.apps.saf.db.dfa_db_models.DfaDBMixin method), 110
update_agent_status() (networking_cisco.apps.saf.server.dfa_server.DfaServer method), 139
update_ccm_host() (networking_cisco.plugins.cisco.cpnr.cpnr_client.CpnrClient method), 191
update_ccm_reverse_zone() (networking_cisco.plugins.cisco.cpnr.cpnr_client.CpnrClient method), 191
update_ccm_zone() (networking_cisco.plugins.cisco.cpnr.cpnr_client.CpnrClient method), 191
update_client_class() (networking_cisco.plugins.cisco.cpnr.cpnr_client.CpnrClient method), 191
update_client_entry() (networking_cisco.plugins.cisco.cpnr.cpnr_client.CpnrClient method), 191
update_dcnm_in_part() (networking_cisco.apps.saf.server.services.firewall.native.fabric_setup_bas
method), 125
update_dcnm_net_info() (networking_cisco.apps.saf.server.services.firewall.native.fabric_setup_bas
method), 125
update_dcnm_out_part() (networking_cisco.apps.saf.server.services.firewall.native.fabric_setup_bas
method), 125
update_dcnm_partition_static_route() (networking_cisco.apps.saf.server.services.firewall.native.drivers.native.Na
method), 119
update_device() (network-

V

W