Networking Generic Switch Documentation

Release 7.1.2.dev1

OpenStack Foundation

Jun 26, 2024

CONTENTS

CHAPTER

ONE

SUPPORTED DEVICES

The following devices are supported by this plugin:

- Arista EOS
- ArubaOS-CX switches
- Brocade ICX (FastIron)
- Cisco 300-series switches
- Cisco IOS switches
- Cisco NX-OS switches (Nexus)
- Cumulus Linux (via NCLU)
- Dell Force10
- Dell OS10
- Dell PowerConnect
- HPE 5900 Series switches
- Huawei switches
- Juniper Junos OS switches
- OpenVSwitch
- Ruijie switches
- SONiC switches

This Mechanism Driver architecture allows easily to add more devices of any type.

```
OpenStack Neutron v2.0 => ML2 plugin => Generic Mechanism Driver => Device␣
↪plugin
```

These device plugins use Netmiko library, which in turn uses *Paramiko* library to access and configure the switches via SSH protocol.

CHAPTER

TWO

INSTALLATION

This sections describes how to install and configure networking-generic-switch plugin.

At the command line:

```
$ pip install networking-generic-switch
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv networking-generic-switch
$ pip install networking-generic-switch
```

2.1 Enable genericswitch mechanism driver in Neutron

To enable mechanism drivers in the ML2 plug-in, edit the `/etc/neutron/plugins/ml2/ml2_conf.ini` file on the neutron server:

```
[ml2]
  mechanism_drivers = ovs,genericswitch
```

CHAPTER

THREE

CONFIGURATION

In order to use this mechanism driver the Neutron configuration file needs to be created/updated with the appropriate configuration information.

Switch configuration format:

```
[genericswitch:<switch name>]
device_type = <netmiko device type>
ngs_mac_address = <switch mac address>
ip = <IP address of switch>
port = <ssh port>
username = <credential username>
password = <credential password>
key_file = <ssh key file>
secret = <enable secret>

# If set ngs_port_default_vlan to default_vlan, switch's
# interface will restore the default_vlan.
ngs_port_default_vlan = <port default vlan>
```

The `device_type` entry is mandatory. Most other configuration entries are optional, see below.

Note: Switch will be selected by local_link_connection/switch_info or ngs_mac_address. So, you can use the switch MAC address to identify switches if local_link_connection/switch_info is not set.

3.1 Examples

These example device configuration snippets are assumed to be part to a specific file `/etc/neutron/plugins/ml2/ml2_conf_genericswitch.ini`, but they could also be added directly to `/etc/neutron/plugins/ml2/ml2_conf.ini`.

Here is an example for the Cisco 300 series device:

```
[genericswitch:sw-hostname]
device_type = netmiko_cisco_s300
ngs_mac_address = <switch mac address>
username = admin
password = password
ip = <switch mgmt ip address>
```

for the Cisco IOS device:

```
[genericswitch:sw-hostname]
device_type = netmiko_cisco_ios
ngs_mac_address = <switch mac address>
username = admin
password = password
secret = secret
ip = <switch mgmt ip address>
```

for the Cisco NX-OS device:

```
[genericswitch:sw-hostname]
device_type = netmiko_cisco_nxos
ngs_mac_address = <switch mac address>
ip = <switch mgmt ip address>
username = admin
password = password
secret = secret
```

for the Huawei VRPV3 or VRPV5 device:

```
[genericswitch:sw-hostname]
device_type = netmiko_huawei
ngs_mac_address = <switch mac address>
username = admin
password = password
port = 8222
secret = secret
ip = <switch mgmt ip address>
```

for the Huawei VRPV8 device:

```
[genericswitch:sw-hostname]
device_type = netmiko_huawei_vrpv8
ngs_mac_address = <switch mac address>
username = admin
password = password
port = 8222
secret = secret
ip = <switch mgmt ip address>
```

for the Arista EOS device:

```
[genericswitch:arista-hostname]
device_type = netmiko_arista_eos
ngs_mac_address = <switch mac address>
ip = <switch mgmt ip address>
username = admin
key_file = /opt/data/arista_key
```

for the Dell Force10 device:

```
[genericswitch:dell-hostname]
device_type = netmiko_dell_force10
ngs_mac_address = <switch mac address>
ip = <switch mgmt ip address>
username = admin
password = password
secret = secret
```

for the Dell OS10 device:

```
[genericswitch:dell-hostname]
device_type = netmiko_dell_os10
ngs_mac_address = <switch mac address>
ip = <switch mgmt ip address>
username = admin
password = password
secret = secret
```

for the Dell PowerConnect device:

```
[genericswitch:dell-hostname]
device_type = netmiko_dell_powerconnect
ip = <switch mgmt ip address>
username = admin
password = password
secret = secret

# You can set ngs_switchport_mode according to switchmode you have set on
# the switch. The following options are supported: general, access. It
# will default to access mode if left unset. In general mode, the port
# be set to transmit untagged packets.
ngs_switchport_mode = access
```

Dell PowerConnect devices have been seen to have issues with multiple concurrent configuration sessions. See *Synchronization* and *Batching* for details on how to limit the number of concurrent active connections to each device.

for the Brocade FastIron (ICX) device:

```
[genericswitch:hostname-for-fast-iron]
device_type = netmiko_brocade_fastiron
ngs_mac_address = <switch mac address>
ip = <switch mgmt ip address>
username = admin
password = password
```

for the Ruijie device:

```
[genericswitch:sw-hostname]
device_type = netmiko_ruijie
ngs_mac_address = <switch mac address>
```

(continues on next page)

(continued from previous page)

```
username = admin
password = password
secret = secret
ip = <switch mgmt ip address>
```

for the HPE 5900 Series device:

```
[genericswitch:sw-hostname]
device_type = netmiko_hp_comware
username = admin
password = password
ip = <switch mgmt ip address>
```

for the Juniper Junos OS device:

```
[genericswitch:hostname-for-juniper]
device_type = netmiko_juniper
ip = <switch mgmt ip address>
username = admin
password = password
ngs_commit_timeout = <optional commit timeout (seconds)>
ngs_commit_interval = <optional commit interval (seconds)>
```

for a Cumulus Linux device:

```
[genericswitch:hostname-for-cumulus]
device_type = netmiko_cumulus
ip = <switch mgmt_ip address>
username = admin
password = password
secret = secret
ngs_mac_address = <switch mac address>
```

for the Nokia SRL series device:

```
[genericswitch:sw-hostname]
device_type = netmiko_nokia_srl
username = admin
password = password
ip = <switch mgmt ip address>
```

for a Pluribus switch:

```
[genericswitch:sw-hostname]
device_type = netmiko_pluribus
username = admin
password = password
ip = <switch mgmt ip address>
```

for an ArubaOS-CX switch:

```
[genericswitch:aruba-hostname]
device_type = netmiko_aruba_os
username = admin
password = password
ip = <switch mgmt ip address>
```

3.2 General configuration

Additionally the `GenericSwitch` mechanism driver needs to be enabled from the ml2 config file `/etc/neutron/plugins/ml2/ml2_conf.ini`:

```
[ml2]
tenant_network_types = vlan
type_drivers = local,flat,vlan,gre,vxlan
mechanism_drivers = openvswitch,genericswitch
...
```

Physical networks need to be declared in the ML2 config as well, with a range of VLANs that can be allocated to tenant networks. Several physical networks can coexist, possibly with overlapping VLAN ranges: in that case, each switch configuration needs to include its physical network, see *Multiple physical networks*. Example of `/etc/neutron/plugins/ml2/ml2_conf.ini` with two physical networks:

```
[ml2_type_vlan]
network_vlan_ranges = physnet1:700:799,physnet2:600:850
```

For a given physical network, it is possible to specify several disjoint ranges of VLANs by simply repeating the physical network name multiple times:

```
[ml2_type_vlan]
network_vlan_ranges = physnet1:700:720,physnet1:750:760
```

(Re)start `neutron-server` specifying the additional configuration file containing switch configuration:

```
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_genericswitch.ini
```

3.3 Synchronization

Some devices are limited in the number of concurrent SSH sessions that they can support, or do not support concurrent configuration database updates. In these cases it can be useful to use an external service to synchronize access to the managed devices. This synchronization is provided by the Tooz library, which provides support for a number of different backends, including Etcd, ZooKeeper, and others. A connection URL for the backend should be configured as follows:

```
[ngs_coordination]
backend_url = <backend URL>
```

The backend URL format includes the Tooz driver as the scheme, with driver options passed using query string parameters. For example, to use the `etcd3gw` driver with an API version of `v3` and a path to a CA certificate:

```
[ngs_coordination]
backend_url = etcd3+https://etcd.example.com?api_version=v3,ca_cert=/path/to/
↪ca/cert.crt
```

The default behaviour is to limit the number of concurrent active connections to each device to one, but the number may be configured per-device as follows:

```
[genericswitch:device-hostname]
ngs_max_connections = <max connections>
```

When synchronization is used, each Neutron thread executing the networking-generic-switch plugin will attempt to acquire a lock, with a default timeout of 60 seconds before failing. This timeout can be configured as follows (setting it to 0 means no timeout):

```
[ngs_coordination]
...
acquire_timeout = <timeout in seconds>
```

3.4 Batching

For many network devices there is a significant SSH connection overhead which is incurred for each network or port configuration change. In a large scale system with many concurrent changes, this overhead adds up quickly. Since the Antelope release, the Generic Switch driver includes support to batch up switch configuration changes and apply them together using a single SSH connection.

This is implemented using etcd as a queueing system. Commands are added to an input key, then a worker thread processes the available commands for a particular switch device. We pull off the queue using the version at which the keys were added, giving a FIFO style queue. The result of each command set are added to an output key, which the original request thread is watching. Distributed locks are used to serialise the processing of commands for each switch device.

The etcd endpoint is configured using the same `[ngs_coordination] backend_url` option used in *Synchronization*, with the limitation that only `etcd3gw` is supported.

Additionally, each device that will use batched configuration should include the following option:

```
[genericswitch:device-hostname]
ngs_batch_requests = True
```

3.5 Disabling Inactive Ports

By default, switch interfaces remain administratively enabled when not in use, and the access VLAN association is removed. On most devices, this will cause the interface to be a member of the default VLAN, usually VLAN 1. This could be a security issue, with unallocated ports having access to a shared network.

To resolve this issue, it is possible to configure interfaces as administratively down when not in use. This is done on a per-device basis, using the `ngs_disable_inactive_ports` flag:

```
[genericswitch:device-hostname]
ngs_disable_inactive_ports = <optional boolean>
```

This is currently supported by the following devices:

- Juniper Junos OS
- ArubaOS-CX
- Cisco NX-OS

3.6 Network Name Format

By default, when a network is created on a switch, if the switch supports assigning names to VLANs, they are assigned a name of the neutron network UUID. For example:

```
8f60256e4b6343bf873026036606ce5e
```

It is possible to use a different format for the network name using the `ngs_network_name_format` option. This option uses Python string formatting syntax, and accepts the parameters `{network_id}` and `{segmentation_id}`. For example:

```
[genericswitch:device-hostname]
ngs_network_name_format = neutron-{network_id}-{segmentation_id}
```

Some switches have issues assigning VLANs a name that starts with a number, and this configuration option can be used to avoid this.

3.7 Manage VLANs

By default, on network creation VLANs are added to all switches. In a similar way, VLANs are removed when it seems they are no longer required. However, in some cases only a subset of the ports are managed by Neutron. In a similar way, when multiple switches are used, it is very common that the network administrator restricts the VLANs allowed. In these cases, there is little utility in adding and removing vlans on the switches. This process takes time, so not doing this can speed up a number of common operations. A particular case where this can cause problems is when a VLAN used for the switch management interface, or any other port not managed by Neutron, is removed by this Neutron driver.

To stop networking generic switch trying to add or remove VLANs on the switch, administrator are expected to pre-add all enabled VLANs as well as tagging these VLANs on trunk ports. Once those VLANs and trunk ports are preconfigured on the switch, you can use the following configuration to stop networking generic switch adding or removing any VLANs:

```
[genericswitch:device-hostname]
ngs_manage_vlans = False
```

3.8 Saving configuration on devices

By default, all configuration changes are saved on persistent storage of the devices, using model-specific commands. This occurs after each change.

This may be undesirable for performance reasons, or if you have external means of saving configuration on a regular basis. In this case, configuration saving can be disabled:

```
[genericswitch:device-hostname]
ngs_save_configuration = False
```

3.9 Trunk ports

When VLANs are created on the switches, it is common to want to tag these VLANS on one or more trunk ports. To do this, you need to declare a comma-separated list of trunk ports that can be managed by Networking Generic Switch. It will then dynamically tag and untag VLANs on these ports whenever it creates and deletes VLANs. For example:

```
[genericswitch:device-hostname]
ngs_trunk_ports = Ethernet1/48, Port-channel1
```

This is useful when managing several switches in the same physical network, because they are likely to be interconnected with trunk links. Another important use-case is to connect the DHCP agent with a trunk port, because the agent needs access to all active VLANs.

Note that this option is only used if `ngs_manage_vlans = True`.

3.10 Multiple physical networks

It is possible to use Networking Generic Switch to manage several physical networks. The desired physical network is selected by the Neutron API client when it creates the network object.

In this case, you may want to only create VLANs on switches that belong to the requested physical network, especially because VLAN ranges from separate physical networks may overlap. This also improves reconfiguration performance because fewer switches will need to be configured whenever a network is created/deleted.

To this end, each switch can be configured with a list of physical networks it belongs to:

```
[genericswitch:device-hostname]
ngs_physical_networks = physnet1, physnet2
```

Physical network names should match the names defined in the ML2 configuration.

If no physical network is declared in a switch configuration, then VLANs for all physical networks will be created on this switch.

Note that this option is only used if `ngs_manage_vlans = True`.

3.11 SSH algorithm configuration

You may need to tune the SSH negotiation process for some devices. Reasons include using a faster key exchange algorithm, disabling an algorithm that has a buggy implementation on the target device, or working around limitations related to FIPS requirements.

The `ngs_ssh_disabled_algorithms` configuration parameter allows to selectively disable algorithms of a given type (key exchange, cipher, MAC, etc). It is based on Paramikos disabled_algorithms setting.

The format is a list of `<type>:<algorithm>` entries to disable. The same type can be repeated several times with different algorithms. Here is an example configuration:

```
[genericswitch:device-hostname]
ngs_ssh_disabled_algorithms = kex:diffie-hellman-group-exchange-sha1,␣
↪ciphers:blowfish-cbc, ciphers:3des-cbc
```

As of Paramiko 2.9.1, the valid types are `ciphers`, `macs`, `keys`, `pubkeys`, `kex`, `gsskex`. However, this might change depending on the version of Paramiko. Check Paramiko source code or documentation to determine the accepted algorithm types.

CHAPTER

FOUR

DEVELOPER QUICK-START

This is a quick walk through to get you started developing code for Networking-generic-switch. This assumes you are already familiar with submitting code reviews to an OpenStack project.

CHAPTER

FIVE

DEPLOYING NETWORKING-GENERIC-SWITCH WITH DEVSTACK

DevStack may be configured to deploy Networking-generic-switch, setup Neutron to use the Networking-generic-switch ML2 driver. It is highly recommended to deploy on an expendable virtual machine and not on your personal work station. Deploying Networking-generic-switch with DevStack requires a machine running Ubuntu 14.04 (or later) or Fedora 20 (or later).

See also:

https://docs.openstack.org/devstack/latest/

Devstack will no longer create the user stack with the desired permissions, but does provide a script to perform the task:

```
git clone https://github.com/openstack-dev/devstack.git devstack
sudo ./devstack/tools/create-stack-user.sh
```

Switch to the stack user and clone DevStack:

```
sudo su - stack
git clone https://github.com/openstack-dev/devstack.git devstack
```

Create devstack/local.conf with minimal settings required to enable Networking-generic-switch. Here is and example of local.conf:

```
[[local|localrc]]
# Set credentials
ADMIN_PASSWORD=secrete
DATABASE_PASSWORD=secrete
RABBIT_PASSWORD=secrete
SERVICE_PASSWORD=secrete
SERVICE_TOKEN=secrete

# Enable minimal required services
ENABLED_SERVICES="dstat,mysql,rabbit,key,q-svc,q-agt,q-dhcp"

# Enable networking-generic-switch plugin
enable_plugin networking-generic-switch https://review.openstack.org/
↪openstack/networking-generic-switch

# Configure Neutron
OVS_PHYSICAL_BRIDGE=brbm
PHYSICAL_NETWORK=mynetwork
```

(continues on next page)

(continued from previous page)

```
Q_PLUGIN=ml2
ENABLE_TENANT_VLANS=True
Q_ML2_TENANT_NETWORK_TYPE=vlan
TENANT_VLAN_RANGE=100:150

# Configure logging
LOGFILE=$HOME/devstack.log
LOGDIR=$HOME/logs
```

Run stack.sh:

```
./stack.sh
```

Source credentials:

```
source ~/devstack/openrc admin admin
```

5.1 Test with OVS

Launch exercise.sh from networking-generic-switch. This script creates port in Neutron/update it with local_link_information and verifies that ovs port has been assigned to correct VLAN:

```
bash ~/networking-generic-switch/devstack/exercise.sh
```

5.2 Test with real hardware:

Add information about hardware switch to Networking-generic-switch config `/etc/neutron/plugins/ml2/ml2_conf_genericswitch.ini` and restart Neutron server:

```
[genericswitch:cisco_switch_1]
device_type = netmiko_cisco_ios
ip = 1.2.3.4
username = cisco
password = cisco
secret = enable_password
```

Get current configuration of the port on the switch, for example for Cisco IOS device:

```
sh running-config int gig 0/12
Building configuration...

Current configuration : 283 bytes
!
interface GigabitEthernet0/12
 switchport mode access
end
```

Run exercise.py to create/update Neutron port. It will print VLAN id to be assigned:

```
$ neutron net-create test
$ python ~/networking-generic-switch/devstack/exercise.py --switch_name cisco_
↪switch_1 --port Gig0/12 --switch_id=06:58:1f:e7:b4:44 --network test
126
```

Verify that VLAN has been changed on the switch port, for example for Cisco IOS device:

```
sh running-config int gig 0/12
Building configuration...

Current configuration : 311 bytes
!
interface GigabitEthernet0/12
 switchport access vlan 126
 switchport mode access
end
```

CHAPTER

SIX

CONTRIBUTING

6.1 Contributing to Networking-generic-switch

If you would like to contribute to the development of GenericSwitch project, you must follow the general OpenStack community procedures documented at:

> https://docs.openstack.org/infra/manual/developers.html#development-workflow

Pull requests submitted through GitHub will be ignored.

6.1.1 Contributor License Agreement

In order to contribute to the GenericSwitch project, you need to have signed OpenStacks contributors agreement.

See also:

- https://docs.openstack.org/infra/manual/developers.html
- https://wiki.openstack.org/CLA

6.1.2 Related Projects

- https://docs.openstack.org/neutron/latest
- https://docs.openstack.org/ironic/latest

6.1.3 Project Hosting Details

Bug tracker https://storyboard.openstack.org/#!/project/956

Code Hosting https://opendev.org/openstack/networking-generic-switch

Code Review https://review.opendev.org/#/q/status:open+project:openstack/networking-generic-switch,n,z

6.1.4 Creating new device plugins

1. Subclass the abstract class `networking_generic_switch.devices.GenericSwitch` and implement all the abstract methods it defines.

   - **Your class must accept a single argument for instantiation -** a dictionary with all fields given in the device config section of the ML2 plugin config. This will be available as `self.config` in the instantiated object.

2. Register your class under `generic_switch.devices` entrypoint.

3. Add your device config to the plugin configuration file (`/etc/neutron/plugins/ml2/ml2_conf_genericswitch.ini` by default). The only required option is `device_type` that must be equal to the entrypoint you have registered your plugin under, as it is used for plugin lookup (see provided `Netmiko`-based plugins for example).

CHAPTER

SEVEN

NETWORKING_GENERIC_SWITCH

7.1 networking_generic_switch package

7.1.1 Subpackages

networking_generic_switch.devices package

Subpackages

networking_generic_switch.devices.netmiko_devices package

Submodules

networking_generic_switch.devices.netmiko_devices.arista module

class networking_generic_switch.devices.netmiko_devices.arista.**AristaEos**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

ADD_NETWORK = ('vlan {segmentation_id}', 'name {network_name}')

DELETE_NETWORK = ('no vlan {segmentation_id}',)

DELETE_PORT = ('interface {port}', 'no switchport access vlan {segmentation_id}', 'no switchport mode trunk', 'switchport trunk allowed vlan none')

PLUG_PORT_TO_NETWORK = ('interface {port}', 'switchport mode access', 'switchport access vlan {segmentation_id}')

networking_generic_switch.devices.netmiko_devices.aruba module

class networking_generic_switch.devices.netmiko_devices.aruba.**ArubaOSCX**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

Built for ArubaOS-CX

ADD_NETWORK = ('vlan {segmentation_id}', 'name {network_name}')

ADD_NETWORK_TO_TRUNK = ('interface {port}', 'no routing', 'vlan trunk allowed {segmentation_id}')

DELETE_NETWORK = ('no vlan {segmentation_id}',)

DELETE_PORT = ('interface {port}', 'no vlan access {segmentation_id}')

DISABLE_PORT = ('interface {port}', 'shutdown')

ENABLE_PORT = ('interface {port}', 'no shutdown')

PLUG_PORT_TO_NETWORK = ('interface {port}', 'no routing', 'vlan access {segmentation_id}')

REMOVE_NETWORK_FROM_TRUNK = ('interface {port}', 'no vlan trunk allowed {segmentation_id}')

networking_generic_switch.devices.netmiko_devices.brocade module

class networking_generic_switch.devices.netmiko_devices.brocade.**BrocadeFastIron**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

ADD_NETWORK = ('vlan {segmentation_id} by port', 'name {network_name}')

DELETE_NETWORK = ('no vlan {segmentation_id}',)

DELETE_PORT = ('vlan {segmentation_id} by port', 'no untagged ether {port}')

PLUG_PORT_TO_NETWORK = ('vlan {segmentation_id} by port', 'untagged ether {port}')

QUERY_PORT = ('show interfaces ether {port} | include VLAN',)

clean_port_vlan_if_necessary(*port*)

get_wrong_vlan(*port*)

plug_port_to_network(*port*, *segmentation_id*)

networking_generic_switch.devices.netmiko_devices.cisco module

class networking_generic_switch.devices.netmiko_devices.cisco.**CiscoIos**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

ADD_NETWORK = ('vlan {segmentation_id}', 'name {network_name}')

DELETE_NETWORK = ('no vlan {segmentation_id}',)

DELETE_PORT = ('interface {port}', 'no switchport access vlan {segmentation_id}', 'no switchport mode trunk', 'switchport trunk allowed vlan none')

PLUG_PORT_TO_NETWORK = ('interface {port}', 'switchport mode access', 'switchport access vlan {segmentation_id}')

class networking_generic_switch.devices.netmiko_devices.cisco.**CiscoNxOS**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

Netmiko device driver for Cisco Nexus switches running NX-OS.

ADD_NETWORK = ('vlan {segmentation_id}', 'name {network_name}', 'exit')

ADD_NETWORK_TO_TRUNK = ('interface {port}', 'switchport mode trunk', 'switchport trunk allowed vlan add {segmentation_id}', 'exit')

DELETE_NETWORK = ('no vlan {segmentation_id}',)

DELETE_PORT = ('interface {port}', 'no switchport access vlan', 'exit')

DISABLE_PORT = ('interface {port}', 'shutdown', 'exit')

ENABLE_PORT = ('interface {port}', 'no shutdown', 'exit')

PLUG_PORT_TO_NETWORK = ('interface {port}', 'switchport mode access', 'switchport access vlan {segmentation_id}', 'exit')

REMOVE_NETWORK_FROM_TRUNK = ('interface {port}', 'switchport trunk allowed vlan remove {segmentation_id}', 'exit')

networking_generic_switch.devices.netmiko_devices.cisco300 module

class networking_generic_switch.devices.netmiko_devices.cisco300.**Cisco300**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

ADD_NETWORK = ('vlan {segmentation_id}',)

DELETE_NETWORK = ('no vlan {segmentation_id}',)

DELETE_PORT = ('interface {port}', 'no switchport access vlan', 'switchport trunk allowed vlan remove all')

PLUG_PORT_TO_NETWORK = ('interface {port}', 'switchport mode access', 'switchport access vlan {segmentation_id}')

networking_generic_switch.devices.netmiko_devices.cumulus module

class networking_generic_switch.devices.netmiko_devices.cumulus.**Cumulus**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

Built for Cumulus 4.x

Note for this switch you want config like this, where secret is the password needed for sudo su:

[genericswitch:<hostname>] device_type = netmiko_cumulus ip = <ip> username = <username> password = <password> secret = <password for sudo> ngs_physical_networks = physnet1 ngs_max_connections = 1 ngs_port_default_vlan = 123 ngs_disable_inactive_ports = False

ADD_NETWORK = ['net add vlan {segmentation_id}']

DELETE_NETWORK = ['net del vlan {segmentation_id}']

DELETE_PORT = ['net del interface {port} bridge access {segmentation_id}']

DISABLE_BOND = ['net add bond {bond} link down']

DISABLE_PORT = ['net add interface {port} link down']

ENABLE_BOND = ['net del bond {bond} link down']

ENABLE_PORT = ['net del interface {port} link down']

ERROR_MSG_PATTERNS = [re.compile('ERROR: Command not found.'), re.compile('command not found'), re.compile('is not a physical interface on this switch')]

Sequence of error message patterns.

Sequence of re.RegexObject objects representing patterns to check for in device output that indicate a failure to apply configuration.

NETMIKO_DEVICE_TYPE = 'linux'

PLUG_BOND_TO_NETWORK = ['net add bond {bond} bridge access {segmentation_id}']

PLUG_PORT_TO_NETWORK = ['net add interface {port} bridge access {segmentation_id}']

SAVE_CONFIGURATION = ['net commit']

UNPLUG_BOND_FROM_NETWORK = ['net del bond {bond} bridge access {segmentation_id}']

networking_generic_switch.devices.netmiko_devices.dell module

class networking_generic_switch.devices.netmiko_devices.dell.**DellNos**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

Netmiko device driver for Dell Force10 (OS9) switches.

ADD_NETWORK = ('interface vlan {segmentation_id}', 'description {network_name}', 'exit')

ADD_NETWORK_TO_TRUNK = ('interface vlan {segmentation_id}', 'tagged {port}', 'exit')

DELETE_NETWORK = ('no interface vlan {segmentation_id}', 'exit')

DELETE_PORT = ('interface vlan {segmentation_id}', 'no untagged {port}', 'exit')

PLUG_PORT_TO_NETWORK = ('interface vlan {segmentation_id}', 'untagged {port}', 'exit')

REMOVE_NETWORK_FROM_TRUNK = ('interface vlan {segmentation_id}', 'no tagged {port}', 'exit')

class networking_generic_switch.devices.netmiko_devices.dell.**DellOS10**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

Netmiko device driver for Dell PowerSwitch switches.

ADD_NETWORK = ('interface vlan {segmentation_id}', 'description {network_name}', 'exit')

ADD_NETWORK_TO_TRUNK = ('interface {port}', 'switchport mode trunk', 'switchport trunk allowed vlan {segmentation_id}', 'exit')

DELETE_NETWORK = ('no interface vlan {segmentation_id}', 'exit')

DELETE_PORT = ('interface {port}', 'no switchport access vlan', 'exit')

DISABLE_PORT = ('interface {port}', 'shutdown', 'exit')

ENABLE_PORT = ('interface {port}', 'no shutdown', 'exit')

ERROR_MSG_PATTERNS = ()

Sequence of error message patterns.

Sequence of re.RegexObject objects representing patterns to check for in device output that indicate a failure to apply configuration.

PLUG_PORT_TO_NETWORK = ('interface {port}', 'switchport mode access', 'switchport access vlan {segmentation_id}', 'exit')

REMOVE_NETWORK_FROM_TRUNK = ('interface {port}', 'no switchport trunk allowed vlan {segmentation_id}', 'exit')

class networking_generic_switch.devices.netmiko_devices.dell.**DellPowerConnect**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

Netmiko device driver for Dell PowerConnect switches.

ADD_NETWORK = ('vlan database', 'vlan {segmentation_id}', 'exit')

ADD_NETWORK_TO_TRUNK = ('interface {port}', 'switchport general allowed vlan add {segmentation_id} tagged', 'exit')

DELETE_NETWORK = ('vlan database', 'no vlan {segmentation_id}', 'exit')

DELETE_PORT = ('interface {port}', 'switchport access vlan none', 'exit')

DELETE_PORT_GENERAL = ('interface {port}', 'switchport general allowed vlan remove {segmentation_id}', 'no switchport general pvid', 'exit')

ERROR_MSG_PATTERNS = (re.compile('\\% Incomplete command'), re.compile('VLAN was not created by user'), re.compile('Configuration Database locked by another application \\- try later'))

Sequence of error message patterns.

Sequence of re.RegexObject objects representing patterns to check for in device output that indicate a failure to apply configuration.

PLUG_PORT_TO_NETWORK = ('interface {port}', 'switchport access vlan {segmentation_id}', 'exit')

PLUG_PORT_TO_NETWORK_GENERAL = ('interface {port}', 'switchport general allowed vlan add {segmentation_id} untagged', 'switchport general pvid {segmentation_id}', 'exit')

REMOVE_NETWORK_FROM_TRUNK = ('interface {port}', 'switchport general allowed vlan remove {segmentation_id}', 'exit')

networking_generic_switch.devices.netmiko_devices.fake module

class networking_generic_switch.devices.netmiko_devices.fake.**Fake**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

Netmiko device driver for Fake switches.

ADD_NETWORK = ('add network {segmentation_id}',)

ADD_NETWORK_TO_TRUNK = ('add network {segmentation_id} to trunk {port}',)

DELETE_NETWORK = ('delete network {segmentation_id}',)

DELETE_PORT = ('delete port {port}',)

DISABLE_PORT = ('disable {port}',)

ENABLE_PORT = ('enable {port}',)

ERROR_MSG_PATTERNS = ()

Sequence of error message patterns.

Sequence of re.RegexObject objects representing patterns to check for in device output that indicate a failure to apply configuration.

NETMIKO_DEVICE_TYPE = 'linux'

PLUG_PORT_TO_NETWORK = ('plug port {port} to network {segmentation_id}',)

REMOVE_NETWORK_FROM_TRUNK = ('remove network {segmentation_id} from trunk {port}',)

send_commands_to_device(*cmd_set*)

networking_generic_switch.devices.netmiko_devices.hpe module

class networking_generic_switch.devices.netmiko_devices.hpe.**HpeComware**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

ADD_NETWORK = ('vlan {segmentation_id}',)

DELETE_NETWORK = ('undo vlan {segmentation_id}',)

DELETE_PORT = ('interface {port}', 'undo port access vlan')

PLUG_PORT_TO_NETWORK = ('interface {port}', 'port link-type access', 'port access vlan {segmentation_id}')

networking_generic_switch.devices.netmiko_devices.huawei module

class networking_generic_switch.devices.netmiko_devices.huawei.**Huawei**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

For Huawei Network Operating System VRP V3 and V5.

ADD_NETWORK = ('vlan {segmentation_id}',)

DELETE_NETWORK = ('undo vlan {segmentation_id}',)

DELETE_PORT = ('interface {port}', 'undo port default vlan {segmentation_id}')

PLUG_PORT_TO_NETWORK = ('interface {port}', 'port link-type access', 'port default vlan {segmentation_id}')

networking_generic_switch.devices.netmiko_devices.huawei_vrpv8 module

class networking_generic_switch.devices.netmiko_devices.huawei_vrpv8.**Huawei**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

For Huawei Next-Generation Network Operating System VRP V8.

ADD_NETWORK = ('vlan {segmentation_id}', 'commit')

DELETE_NETWORK = ('undo vlan {segmentation_id}', 'commit')

DELETE_PORT = ('interface {port}', 'undo port default vlan {segmentation_id}', 'commit')

PLUG_PORT_TO_NETWORK = ('interface {port}', 'port link-type access', 'port default vlan {segmentation_id}', 'commit')

networking_generic_switch.devices.netmiko_devices.juniper module

class networking_generic_switch.devices.netmiko_devices.juniper.**Juniper**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

ADD_NETWORK = ('set vlans {network_name} vlan-id {segmentation_id}',)

ADD_NETWORK_TO_TRUNK = ('set interfaces {port} unit 0 family ethernet-switching vlan members {segmentation_id}',)

DELETE_NETWORK = ('delete vlans {network_name}',)

```
DELETE_PORT = ('delete interfaces {port} unit 0 family ethernet-switching
vlan members',)

DISABLE_PORT = ('set interfaces {port} disable',)

ENABLE_PORT = ('delete interfaces {port} disable',)

PLUG_PORT_TO_NETWORK = ('delete interfaces {port} unit 0 family
ethernet-switching vlan members', 'set interfaces {port} unit 0 family
ethernet-switching vlan members {segmentation_id}')

REMOVE_NETWORK_FROM_TRUNK = ('delete interfaces {port} unit 0 family
ethernet-switching vlan members {segmentation_id}',)
```

save_configuration(*net_connect*)

Save the devices configuration.

Parameters **net_connect** a netmiko connection object.

Raises GenericSwitchNetmikoConfigError if saving the configuration fails.

send_config_set(*net_connect*, *cmd_set*)

Send a set of configuration lines to the device.

Parameters

- **net_connect** a netmiko connection object.
- **cmd_set** a list of configuration lines to send.

Returns The output of the configuration commands.

networking_generic_switch.devices.netmiko_devices.mellanox_mlnxos module

class networking_generic_switch.devices.netmiko_devices.mellanox_mlnxos.**MellanoxMlnxOS**(*device*

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

```
ADD_NETWORK = ('vlan {segmentation_id}', 'name {network_id}')

DELETE_NETWORK = ('no vlan {segmentation_id}',)

DELETE_PORT = ('interface ethernet {port}', 'no switchport access vlan',
'no switchport mode')

PLUG_PORT_TO_NETWORK = ('interface ethernet {port}', 'switchport mode
access', 'switchport access vlan {segmentation_id}')
```

networking_generic_switch.devices.netmiko_devices.nokia module

class networking_generic_switch.devices.netmiko_devices.nokia.**NokiaSRL**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

ADD_NETWORK = ('set tunnel-interface vxlan0 vxlan-interface {segmentation_id} type bridged', 'set tunnel-interface vxlan0 vxlan-interface {segmentation_id} ingress vni {segmentation_id}', 'set tunnel-interface vxlan0 vxlan-interface {segmentation_id} egress source-ip use-system-ipv4-address', 'set network-instance mac-vrf-{segmentation_id} type mac-vrf', 'set network-instance mac-vrf-{segmentation_id} description OS-Network-ID-{network_name}', 'set network-instance mac-vrf-{segmentation_id} vxlan-interface vxlan0.{segmentation_id}', 'set network-instance mac-vrf-{segmentation_id} protocols bgp-evpn bgp-instance 1 vxlan-interface vxlan0.{segmentation_id}', 'set network-instance mac-vrf-{segmentation_id} protocols bgp-evpn bgp-instance 1 evi {segmentation_id}', 'set network-instance mac-vrf-{segmentation_id} protocols bgp-evpn bgp-instance 1 ecmp 8', 'set network-instance mac-vrf-{segmentation_id} protocols bgp-vpn bgp-instance 1 route-target export-rt target:1:{segmentation_id}', 'set network-instance mac-vrf-{segmentation_id} protocols bgp-vpn bgp-instance 1 route-target import-rt target:1:{segmentation_id}')

DELETE_NETWORK = ('delete network-instance mac-vrf-{segmentation_id}', 'delete tunnel-interface vxlan0 vxlan-interface {segmentation_id}')

DELETE_PORT = ('delete network-instance mac-vrf-{segmentation_id} interface {port}.{segmentation_id}', 'delete interface {port} subinterface {segmentation_id}')

PLUG_PORT_TO_NETWORK = ('set interface {port} subinterface {segmentation_id} type bridged', 'set network-instance mac-vrf-{segmentation_id} interface {port}.{segmentation_id}')

commit(*net_connect*) → str

Try to commit the Nokia SRL configuration.

Parameters **net_connect** a netmiko connection object.

send_commands_to_device(*cmd_set*)

networking_generic_switch.devices.netmiko_devices.ovs module

class networking_generic_switch.devices.netmiko_devices.ovs.**OvsLinux**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

DELETE_PORT = ('ovs-vsctl clear port {port} tag', 'ovs-vsctl clear port {port} trunks', 'ovs-vsctl clear port {port} vlan_mode')

PLUG_PORT_TO_NETWORK = ('ovs-vsctl set port {port} vlan_mode=access', 'ovs-vsctl set port {port} tag={segmentation_id}')

networking_generic_switch.devices.netmiko_devices.pluribus module

class networking_generic_switch.devices.netmiko_devices.pluribus.**Pluribus**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

ADD_NETWORK = ('vlan-create id {segmentation_id} scope fabric ports none description {network_name} auto-vxlan',)

DELETE_NETWORK = ('vlan-delete id {segmentation_id}',)

DELETE_PORT = ('vlan-port-remove vlan-range all ports {port}',)

PLUG_PORT_TO_NETWORK = ('vlan-port-remove vlan-range all ports {port}', 'port-vlan-add port {port} untagged-vlan {segmentation_id}')

networking_generic_switch.devices.netmiko_devices.ruijie module

class networking_generic_switch.devices.netmiko_devices.ruijie.**Ruijie**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

ADD_NETWORK = ('vlan {segmentation_id}', 'name {network_name}')

DELETE_NETWORK = ('no vlan {segmentation_id}',)

DELETE_PORT = ('interface {port}', 'no switchport access vlan {segmentation_id}', 'no switchport mode trunk', 'switchport trunk allowed vlan none')

PLUG_PORT_TO_NETWORK = ('interface {port}', 'switchport mode access', 'switchport access vlan {segmentation_id}')

networking_generic_switch.devices.netmiko_devices.sonic module

class networking_generic_switch.devices.netmiko_devices.sonic.**Sonic**(*device_cfg*)

Bases: *networking_generic_switch.devices.netmiko_devices.NetmikoSwitch*

Built for SONiC 3.x

Note for this switch you want config like this, where secret is the password needed for sudo su:

[genericswitch:<hostname>] device_type = netmiko_sonic ip = <ip> username = <username> password = <password> secret = <password for sudo> ngs_physical_networks = physnet1 ngs_max_connections = 1 ngs_port_default_vlan = 123 ngs_disable_inactive_ports = False

ADD_NETWORK = ['config vlan add {segmentation_id}']

ADD_NETWORK_TO_TRUNK = ['config vlan member add {segmentation_id} {port}']

DELETE_NETWORK = ['config vlan del {segmentation_id}']

DELETE_PORT = ['config vlan member del {segmentation_id} {port}']

ERROR_MSG_PATTERNS = [re.compile("VLAN[0-9]+ doesn\\'t exist"), re.compile('Invalid Vlan Id , Valid Range : 1 to 4094'), re.compile('Interface name is invalid!!'), re.compile('No such command')]

Sequence of error message patterns.

Sequence of re.RegexObject objects representing patterns to check for in device output that indicate a failure to apply configuration.

NETMIKO_DEVICE_TYPE = 'linux'

PLUG_PORT_TO_NETWORK = ['config vlan member add -u {segmentation_id} {port}']

REMOVE_NETWORK_FROM_TRUNK = ['config vlan member del {segmentation_id} {port}']

SAVE_CONFIGURATION = ['config save -y']

send_config_set(*net_connect*, *cmd_set*)

Send a set of configuration lines to the device.

Parameters

- **net_connect** a netmiko connection object.
- **cmd_set** a list of configuration lines to send.

Returns The output of the configuration commands.

Module contents

class networking_generic_switch.devices.netmiko_devices.**NetmikoSwitch**(*device_cfg*)

Bases: *networking_generic_switch.devices.GenericSwitchDevice*

ADD_NETWORK = None

ADD_NETWORK_TO_TRUNK = None

DELETE_NETWORK = None

DELETE_PORT = None

DISABLE_BOND = None

DISABLE_PORT = None

ENABLE_BOND = None

ENABLE_PORT = None

ERROR_MSG_PATTERNS = ()

Sequence of error message patterns.

Sequence of re.RegexObject objects representing patterns to check for in device output that indicate a failure to apply configuration.

NETMIKO_DEVICE_TYPE = None

PLUG_BOND_TO_NETWORK = None

PLUG_PORT_TO_NETWORK = None

REMOVE_NETWORK_FROM_TRUNK = None

SAVE_CONFIGURATION = None

UNPLUG_BOND_FROM_NETWORK = None

add_network(*segmentation_id*, *network_id*)

check_output(*output*, *operation*)

Check the output from the device following an operation.

Drivers should implement this method to handle output from devices and perform any checks necessary to validate that the configuration was applied successfully.

Parameters

- **output** Output from the device.
- **operation** Operation being attempted. One of add network, delete network, plug port, unplug port.

Raises GenericSwitchNetmikoConfigError if the driver detects that an error has occurred.

del_network(*segmentation_id*, *network_id*)

delete_port(*port*, *segmentation_id*)

plug_bond_to_network(*bond*, *segmentation_id*)

plug_port_to_network(*port*, *segmentation_id*)

save_configuration(*net_connect*)

Try to save the devices configuration.

Parameters **net_connect** a netmiko connection object.

send_commands_to_device(*cmd_set*)

send_config_set(*net_connect*, *cmd_set*)

Send a set of configuration lines to the device.

Parameters

- **net_connect** a netmiko connection object.
- **cmd_set** a list of configuration lines to send.

Returns The output of the configuration commands.

unplug_bond_from_network(*bond*, *segmentation_id*)

networking_generic_switch.devices.netmiko_devices.**check_output**(*operation*)

Returns a decorator that checks the output of an operation.

Parameters **operation** Operation being attempted. One of add network, delete network, plug port, unplug port.

Submodules

networking_generic_switch.devices.utils module

networking_generic_switch.devices.utils.**get_hostname**()

Helper to allow isolation of CONF.host and plugin loading.

networking_generic_switch.devices.utils.**get_switch_device**(*switches*, *switch_info=None*, *ngs_mac_address=None*)

Return switch device by specified identifier.

Returns switch device from switches array that matched with any of passed identifiers. ngs_mac_address takes precedence over switch_info, if didnt match any address based on mac fallback to switch_info.

Parameters

- **switch_info** hostname of the switch or any other switch identifier.
- **ngs_mac_address** Normalized mac address of the switch.

Returns switch device matches by specified identifier or None.

networking_generic_switch.devices.utils.**sanitise_config**(*config*)

Return a sanitised configuration of a switch device.

Parameters **config** a configuration dict to sanitise.

Returns a copy of the configuration, with sensitive fields removed.

Module contents

class networking_generic_switch.devices.**GenericSwitchDevice**(*device_cfg*)

Bases: object

abstract add_network(*segmentation_id*, *network_id*)

abstract del_network(*segmentation_id*, *network_id*)

abstract delete_port(*port_id*, *segmentation_id*)

plug_bond_to_network(*bond_id*, *segmentation_id*)

abstract plug_port_to_network(*port_id*, *segmentation_id*)

unplug_bond_from_network(*bond_id*, *segmentation_id*)

networking_generic_switch.devices.**device_manager**(*device_cfg*)

7.1.2 Submodules

7.1.3 networking_generic_switch.batching module

exception networking_generic_switch.batching.**ShutdownTimeout**

Bases: Exception

Exception raised when shutdown timeout is exceeded.

class networking_generic_switch.batching.**SwitchBatch**(*switch_name*, *etcd_url=None*, *switch_queue=None*)

Bases: object

do_batch(*device*, *cmd_set*, *timeout=300*)

Batch up switch configuration commands to reduce overheads.

We collect together the iterables in the cmd_set, and execute them toegether in a single larger batch.

Parameters

- **device** a NetmikoSwitch device object
- **cmd_set** an iterable of commands

Returns output string generated by this command set

class networking_generic_switch.batching.**SwitchQueue**(*switch_name*, *etcd_client*)

Bases: object

EXEC_LOCK = '/ngs/batch/%s/execute_lock'

INPUT_ITEM_KEY = '/ngs/batch/%s/input/%s'

INPUT_PREFIX = '/ngs/batch/%s/input/'

RESULT_ITEM_KEY = '/ngs/batch/%s/output/%s'

acquire_worker_lock(*item*, *acquire_timeout=300*, *lock_ttl=120*, *wait=None*)

Wait for lock needed to call record_result.

This blocks until the work queue is empty of the switch lock is acquired. If we timeout waiting for the lock we raise an exception.

add_batch(*cmds*)

Clients add batch, given key events.

Each batch is given an uuid that is used to generate both and input and result key in etcd.

First we watch for any results, second we write the input in a location that the caller of get_batches will be looking.

No locks are required when calling this function to send work to the workers, and start waiting for results.

Parameters **cmds** an iterable of commands

Returns a SwitchQueueItem object

get_batches(*item=None*)

Return a list of the event dicts written in wait for result.

This is called both with or without getting a lock to get the latest list of work that has send to the per switch queue in etcd.

Parameters **item** Optional SwitchQueueItem object. If provided, only batches added up to and including this item are returned.

record_result(*batch*)

Record the result from executing given command set.

We assume that a lock is held before getting a fresh list of batches, executing them, and then calling this record results function, before finally dropping the lock.

wait_for_result(*item*, *timeout*)

Wait for the result of a command batch.

Parameters

- **item** SwitchQueueItem object returned by add_batch
- **timeout** wait timeout in seconds

Returns output string generated by this command set

Raises Exception if waiting times out or the command batch was unsuccessful

class networking_generic_switch.batching.**SwitchQueueItem**(*uuid*, *create_revision*)

Bases: object

An item in the queue.

7.1.4 networking_generic_switch.config module

networking_generic_switch.config.**get_devices**()

Parse supplied config files and fetch defined supported devices.

7.1.5 networking_generic_switch.exceptions module

exception networking_generic_switch.exceptions.**GenericSwitchBatchError**(***kwargs*)

Bases: *networking_generic_switch.exceptions.GenericSwitchException*

message = 'Batching error: %(device)s, error: %(error)s'

exception networking_generic_switch.exceptions.**GenericSwitchConfigException**(***kwargs*)

Bases: neutron_lib.exceptions.NeutronException

message = '%(option)s must be one of: %(allowed_options)s'

exception networking_generic_switch.exceptions.**GenericSwitchEntrypointLoadError**(***kwargs*)

Bases: *networking_generic_switch.exceptions.GenericSwitchException*

message = 'Failed to load entrypoint %(ep)s: %(err)s'

exception networking_generic_switch.exceptions.**GenericSwitchException**(***kwargs*)

Bases: neutron_lib.exceptions.NeutronException

message = '%(method)s failed.'

exception networking_generic_switch.exceptions.**GenericSwitchNetmikoConfigError**(***kwargs*)

Bases: *networking_generic_switch.exceptions.GenericSwitchException*

message = 'Netmiko configuration error: %(config)s, error: %(error)s'

exception networking_generic_switch.exceptions.**GenericSwitchNetmikoConnectError**(***kwargs*)

Bases: *networking_generic_switch.exceptions.GenericSwitchException*

message = 'Netmiko connection error: %(config)s, error: %(error)s'

exception networking_generic_switch.exceptions.**GenericSwitchNetmikoMethodError**(***kwargs*)

Bases: *networking_generic_switch.exceptions.GenericSwitchException*

message = 'Can not parse arguments: commands %(cmds)s, args %(args)s'

exception networking_generic_switch.exceptions.**GenericSwitchNetmikoNotSupported**(***kwargs*)

Bases: *networking_generic_switch.exceptions.GenericSwitchException*

message = 'Netmiko does not support device type %(device_type)s'

exception networking_generic_switch.exceptions.**GenericSwitchNetworkNameFormatInvalid**(***kwarg*

Bases: *networking_generic_switch.exceptions.GenericSwitchException*

message = "Invalid value for 'ngs_network_name_format': %(name_format)s.
Valid format options include 'network_id' and 'segmentation_id'"

7.1.6 networking_generic_switch.generic_switch_mech module

class networking_generic_switch.generic_switch_mech.**GenericSwitchDriver**

Bases: neutron_lib.plugins.ml2.api.MechanismDriver

bind_port(*context*)

Attempt to bind a port.

Parameters **context** PortContext instance describing the port

This method is called outside any transaction to attempt to establish a port binding using this mechanism driver. Bindings may be created at each of multiple levels of a hierarchical network, and are established from the top level downward. At each level, the mechanism driver determines whether it can bind to any of the network segments in the context.segments_to_bind property, based on the value of the context.host property, any relevant port or network attributes, and its own knowledge of the network topology. At the top level, context.segments_to_bind contains the static segments of the ports network. At each lower level of binding, it contains static or dynamic segments supplied by the driver that bound at the level above. If the driver is able to complete the binding of the port to any segment in context.segments_to_bind, it must call context.set_binding with the binding details. If it can partially bind the port, it must call context.continue_binding with the network segments to be used to bind at the next lower level.

If the binding results are committed after bind_port returns, they will be seen by all mechanism drivers as update_port_precommit and update_port_postcommit calls. But if some other thread or process concurrently binds or updates the port, these binding results will not be committed, and update_port_precommit and update_port_postcommit will not be called on the mechanism drivers with these results. Because binding results can be discarded rather than committed, drivers should avoid making persistent state changes in bind_port, or else must ensure that such state changes are eventually cleaned up.

Implementing this method explicitly declares the mechanism driver as having the intention to bind ports. This is inspected by the QoS service to identify the available QoS rules you can use with ports.

create_network_postcommit(*context*)

Create a network.

Parameters **context** NetworkContext instance describing the new network.

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_network_precommit(*context*)

Allocate resources for a new network.

Parameters **context** NetworkContext instance describing the new network.

Create a new network, 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.

create_port_postcommit(*context*)

Create a port.

Parameters **context** PortContext instance describing the port.

Called after the transaction completes. Call can block, though will block the entire process so care should be taken to not drastically affect performance. Raising an exception will result in the deletion of the resource.

create_port_precommit(*context*)

Allocate resources for a new port.

Parameters **context** PortContext instance describing the port.

Create a new port, 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.

create_subnet_postcommit(*context*)

Create a subnet.

Parameters **context** SubnetContext instance describing the new subnet.

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_subnet_precommit(*context*)

Allocate resources for a new subnet.

Parameters **context** SubnetContext instance describing the new subnet.

rt = context.current device_id = port[device_id] device_owner = port[device_owner] Create a new subnet, 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_network_postcommit(*context*)

Delete a network.

Parameters **context** NetworkContext instance describing the current state of the network, 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_network_precommit(*context*)

Delete resources for a network.

Parameters **context** NetworkContext instance describing the current state of the network, prior to the call to delete it.

Delete network resources previously allocated by this mechanism driver for a network. Called inside transaction context on session. Runtime errors are not expected, but raising an exception will result in rollback of the transaction.

delete_port_postcommit(*context*)

Delete a port.

Parameters **context** PortContext instance describing the current state of the port, prior to the call to delete it.

Called after the transaction completes. 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_port_precommit(*context*)

Delete resources of a port.

Parameters **context** PortContext instance describing the current state of the port, prior to the call to delete it.

Called inside transaction context on session. Runtime errors are not expected, but raising an exception will result in rollback of the transaction.

delete_subnet_postcommit(*context*)

Delete a subnet.

Parameters **context** SubnetContext instance describing the current state of the subnet, 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_subnet_precommit(*context*)

Delete resources for a subnet.

Parameters **context** SubnetContext instance describing the current state of the subnet, prior to the call to delete it.

Delete subnet resources previously allocated by this mechanism driver for a subnet. Called inside transaction context on session. Runtime errors are not expected, but raising an exception will result in rollback of the transaction.

initialize()

Perform driver initialization.

Called after all drivers have been loaded and the database has been initialized. No abstract methods defined below will be called prior to this method being called.

update_network_postcommit(*context*)

Update a network.

Parameters **context** NetworkContext instance describing the new state of the network, as well as the original state prior to the update_network 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_network_postcommit is called for all changes to the network state. It is up to the mechanism driver to ignore state or state changes that it does not know or care about.

update_network_precommit(*context*)

Update resources of a network.

Parameters **context** NetworkContext instance describing the new state of the network, as well as the original state prior to the update_network call.

Update values of a network, updating the associated resources in the database. Called inside transaction context on session. Raising an exception will result in rollback of the transaction.

update_network_precommit is called for all changes to the network state. It is up to the mechanism driver to ignore state or state changes that it does not know or care about.

update_port_postcommit(*context*)

Update a port.

Parameters **context** PortContext instance describing the new state of the port, as well as the original state prior to the update_port call.

Called after the transaction completes. Call can block, though will block the entire process so care should be taken to not drastically affect performance. Raising an exception will result in the deletion of the resource.

update_port_postcommit is called for all changes to the port state. It is up to the mechanism driver to ignore state or state changes that it does not know or care about.

update_port_precommit(*context*)

Update resources of a port.

Parameters **context** PortContext instance describing the new state of the port, as well as the original state prior to the update_port call.

Called inside transaction context on session to complete a port update as defined by this mechanism driver. Raising an exception will result in rollback of the transaction.

update_port_precommit is called for all changes to the port state. It is up to the mechanism driver to ignore state or state changes that it does not know or care about.

update_subnet_postcommit(*context*)

Update a subnet.

Parameters **context** SubnetContext instance describing the new state of the subnet, as well as the original state prior to the update_subnet 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_subnet_postcommit is called for all changes to the subnet state. It is up to the mechanism driver to ignore state or state changes that it does not know or care about.

update_subnet_precommit(*context*)

Update resources of a subnet.

Parameters **context** SubnetContext instance describing the new state of the subnet, as well as the original state prior to the update_subnet call.

Update values of a subnet, updating the associated resources in the database. Called inside transaction context on session. Raising an exception will result in rollback of the transaction.

update_subnet_precommit is called for all changes to the subnet state. It is up to the mechanism driver to ignore state or state changes that it does not know or care about.

7.1.7 networking_generic_switch.locking module

class networking_generic_switch.locking.**PoolLock**(*coordinator*, *locks_pool_size=1*, *locks_prefix='ngs-'*, *timeout=0*)

Bases: object

Tooz lock wrapper for pools of locks

If tooz coordinator is provided, it will attempt to grab any lock from a predefined set of names, with configurable set size (lock pool), and keep attempting for until given timeout is reached.

7.1.8 Module contents

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

INDEX

A

B

C

D

E

N

O

P

Q

R

S

U

W