OpenStack-Ansible Documentation: os_neutron role

Release 18.1.0.dev751

OpenStack-Ansible Contributors

Mar 27, 2026

CONTENTS

CHAPTER

ONE

CONFIGURING THE NETWORKING SERVICE (NEUTRON) (OPTIONAL)

The OpenStack Networking service (neutron) includes the following services:

Firewall as a Service (FWaaS)
Provides a software-based firewall that filters traffic from the router.

VPN as a Service (VPNaaS)
Provides a method for extending a private network across a public network.

BGP Dynamic Routing service
Provides a means for advertising self-service (private) network prefixes to physical network devices that support BGP.

SR-IOV Support
Provides the ability to provision virtual or physical functions to guest instances using SR-IOV and PCI passthrough. (Requires compatible NICs)

1.1 Firewall service (optional)

The following procedure describes how to modify the `/etc/openstack_deploy/user_variables.yml` file to enable FWaaS.

1.1.1 Deploying FWaaS v2

FWaaS v2 is the next generation Neutron firewall service and will provide a rich set of APIs for securing OpenStack networks. It is still under active development.

Refer to the FWaaS 2.0 API specification for more information on these FWaaS v2 features.

FWaaS v2 requires the use of Open vSwitch. To deploy an environment using Open vSwitch for virtual networking, please refer to the following documentation:

- Scenario - Using Open vSwitch
- Scenario - Using Open vSwitch with DVR

Follow the steps below to deploy FWaaS v2:

Note

FWaaS v1 and v2 cannot be deployed simultaneously.

1. Add the FWaaS v2 plugin to the `neutron_plugin_base` variable in `/etc/openstack_deploy/user_variables.yml`:

```
neutron_plugin_base:
  - router
  - metering
  - firewall_v2
```

   Ensure that `neutron_plugin_base` includes all of the plugins that you want to deploy with neutron in addition to the firewall_v2 plugin.

2. Run the neutron playbook to deploy the FWaaS v2 service plugin

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible os-neutron-install.yml
```

1.2 Virtual private network service - VPNaaS (optional)

The following procedure describes how to modify the `/etc/openstack_deploy/user_variables.yml` file to enable VPNaaS.

1. Override the default list of neutron plugins to include `vpnaas`:

```
neutron_plugin_base:
  - router
  - metering
```

2. `neutron_plugin_base` is as follows:

> **Note**
>
> In the case your `neutron_plugin_type` is `ml2.ovn`, use `ovn-vpnaas` plugin instead

```
neutron_plugin_base:
  - router
  - metering
  - vpnaas
```

3. Override the default list of specific kernel modules in order to include the necessary modules to run ipsec:

```
openstack_host_specific_kernel_modules:
  - { name: "ebtables", pattern: "CONFIG_BRIDGE_NF_EBTABLES=", group:
↪"network_hosts" }
  - { name: "af_key", pattern: "CONFIG_NET_KEY=", group: "network_hosts"␣
↪}
  - { name: "ah4", pattern: "CONFIG_INET_AH=", group: "network_hosts" }
  - { name: "ipcomp", pattern: "CONFIG_INET_IPCOMP=", group: "network_
↪hosts" }
```

4. Execute the openstack hosts setup in order to load the kernel modules at boot and runtime in the network hosts

```
# openstack-ansible openstack-hosts-setup.yml --limit network_hosts\
--tags "openstack_hosts-config"
```

5. Execute the neutron install playbook in order to update the configuration:

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible os-neutron-install.yml
```

6. Execute the horizon install playbook to show the VPNaaS panels:

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible os-horizon-install.yml
```

The VPNaaS default configuration options are changed through the conf override mechanism using the `neutron_neutron_conf_overrides` dict.

You can also define customized configuration files for VPN service with the variable `neutron_vpnaas_custom_config`:

```
neutron_vpnaas_custom_config:
   - src: "/etc/openstack_deploy/strongswan/strongswan.conf.template"
     dest: "{{ neutron_conf_version_dir }}/strongswan.conf.template"
     condition: "{{ ansible_facts['os_family'] | lower == 'debian' }}"
   - src: "/etc/openstack_deploy/strongswan/strongswan.d"
     dest: "/etc/strongswan.d"
     condition: "{{ ansible_facts['os_family'] | lower == 'debian' }}"
   - src: "/etc/openstack_deploy/neutron/ipsec.conf.template"
     dest: "{{ neutron_conf_version_dir }}/ipsec.conf.template"
```

With that `neutron_l3_agent_ini_overrides` should be also defined in user_variables.yml to tell `l3_agent` use the new config file:

Note

Please, use variable `neutron_ovn_vpn_agent_overrides` when `neutron_plugin_type` is set to `ml2.ovn`.

```
neutron_l3_agent_ini_overrides:
      ipsec:
         enable_detailed_logging: True
      strongswan:
         strongswan_config_template : "{{ neutron_conf_dir }}/strongswan.conf.
↪template"
      openswan:
         ipsec_config_template:  "{{ neutron_conf_dir }}/ipsec.conf.template"
```

1.2.1 VPNaaS Agent for OVN

Since 2024.1 release (Caracal) VPNaaS service does support `ml2.ovn` plugin type.

While configuration of the service is pretty much alike, implementation beneath has significant differences.

First of all, VPNaaS is represented with a standalone agent that is coordinated with help of RabbitMQ. This means, that a new Agent Type `VPN Agent` will appear in `openstack network agent list` output. On a VPN site connection creation, VPN agent will handle a namespace creation on an arbitrary OVN gateway node, inside which ipsec connection will be created

Since OVN L3 Router implementation is not using namespaces, VPN Agent will utilize an extra external IP, since it can not be shared now with the router. Moreover, an extra patch network will be created to connect VPN Agent with L3 agent.

For more details on the implementation please reffer to the VPNaaS OVN Spec

1.3 BGP Dynamic Routing service (optional)

The BGP Dynamic Routing plugin for neutron provides BGP speakers which can advertise OpenStack project network prefixes to external network devices, such as routers. This is especially useful when coupled with the subnet pools feature, which enables neutron to be configured in such a way as to allow users to create self-service segmented IPv6 subnets.

The following procedure describes how to modify the `/etc/openstack_deploy/user_variables.yml` file to enable the BGP Dynamic Routing plugin.

1. Add the BGP plugin to the `neutron_plugin_base` variable in `/etc/openstack_deploy/user_variables.yml`:

   ```
   neutron_plugin_base:
     - ...
     - neutron_dynamic_routing.services.bgp.bgp_plugin.BgpPlugin
   ```

   Ensure that `neutron_plugin_base` includes all of the plugins that you want to deploy with neutron in addition to the BGP plugin.

2. Execute the neutron install playbook in order to update the configuration:

   ```
   # cd /opt/openstack-ansible/playbooks
   # openstack-ansible os-neutron-install.yml
   ```

1.4 OVN BGP Agent (optional)

The OVN BGP Agent exposes VM Floating IPs on provider networks through BGP by leveraging FRRouting.

This intends to provide feature-complete replacement for BGP Dynamic Routing service for environments running OVN as their ml2 plugin.

OVN BGP Agent provides multiple drivers and exposure methods which can be checked on the BGP Supportability Matrix

Note

At the moment of writing only `underlay` exposure method is fully supported by `os_neutron` role.

In order to enable ovn-bgp-agent you need to explicitly configure some variables:

```
neutron_ovn_bgp_enable: true
# This defines an AS to which ovn-bgp-agent will inject an VRF to FRR
neutron_ovn_bgp_config:
   AS: 64513

# In this variable we define a base configuration for FRR that will be
# deployed as pre-requisite of ovn-bgp-agent
neutron_frr_bgp_config:
- router bgp 64513
- "bgp router-id {{ ansible_facts['bond0']['ipv4']['address'] }}"
- bgp log-neighbor-changes
- bgp graceful-shutdown
- neighbor uplink peer-group
- neighbor uplink remote-as 64512
- neighbor uplink ebgp-multihop
- neighbor 203.0.113.10 peer-group uplink
- neighbor 203.0.113.11 peer-group uplink
- neighbor 203.0.113.10 description leaf_1
- neighbor 203.0.113.11 description leaf_2
- address-family ipv4 unicast
- "  redistribute connected"
- "  neighbor uplink activate"
- "  neighbor uplink allowas-in origin"
- "  neighbor uplink prefix-list only-host-prefixes out"
- "exit-address-family"
- "ip prefix-list only-default permit 0.0.0.0/0"
- "ip prefix-list only-host-prefixes permit 0.0.0.0/0 ge 32"
- route-map rm-only-default permit 10
- "  match ip address prefix-list only-default"
- "  set src {{ ansible_facts['bond0']['ipv4']['address'] }}"
- ip protocol bgp route-map rm-only-default

# This variable might be useful for ebgp-multihop scenarios
neutron_frr_staticd_routes:
- ip route 203.0.113.10/32 198.51.100.1
- ip route 203.0.113.10/32 198.51.100.1
```

Once all required variables are set, running `openstack-ansible os-neutron-install.yml` should install and configure FRRouting on all of your `neutron_ovn_controller` as well as a new service `neutron-ovn-bgp-agent` will appear.

This service does not use RabbitMQ for communication and listens for events directly on OVN NB/SB databases, so it will not appear on `openstack network agent list` output like one may assume.

1.5 SR-IOV Support (optional)

The following procedure describes how to modify the OpenStack-Ansible configuration to enable Neutron SR-IOV support.

1. Define SR-IOV capable physical host interface for a provider network

   As part of every OpenStack-Ansible installation, all provider networks known to Neutron need to be configured inside the `/etc/openstack_deploy/openstack_user_config.yml` file. For each supported network type (e.g. vlan), the attribute `sriov_host_interfaces` can be defined to map ML2 network names (`net_name` attribute) to one or many physical interfaces. Additionally, the network will need to be assigned to the `neutron_sriov_nic_agent` container group.

   Example configuration:

```
provider_networks
  - network:
    container_bridge: "br-vlan"
    container_type: "veth"
    container_interface: "eth11"
    type: "vlan"
    range: "1000:2000"
    net_name: "physnet1"
    sriov_host_interfaces: "p1p1,p4p1"
    group_binds:
      - neutron_openvswitch_agent
      - neutron_sriov_nic_agent
```

2. Configure Nova

   With SR-IOV, Nova uses PCI passthrough to allocate VFs and PFs to guest instances. Virtual Functions (VFs) represent a slice of a physical NIC, and are passed as virtual NICs to guest instances. Physical Functions (PFs), on the other hand, represent an entire physical interface and are passed through to a single guest.

   To use PCI passthrough in Nova, the `PciPassthroughFilter` filter needs to be added to the conf override `nova_scheduler_default_filters`. Finally, PCI devices available for passthrough need to be allow via the conf override `nova_pci_passthrough_whitelist`.

   Possible options which can be configured:

```
# Single device configuration
nova_pci_passthrough_whitelist: '{ "physical_network":"physnet1", "devname
↪":"p1p1" }'

# Multi device configuration
nova_pci_passthrough_whitelist: '[{"physical_network":"physnet1", "devname
↪":"p1p1"}, {"physical_network":"physnet1", "devname":"p4p1"}]'

# Whitelisting by PCI Device Location
# The example pattern for the bus location '0000:04:*.*' is very wide.␣
↪Make sure that
# no other, unintended devices, are whitelisted (see lspci -nn)
nova_pci_passthrough_whitelist: '{"address":"0000:04:*.*", "physical_
```

(continues on next page)

(continued from previous page)

```
↪network":"physnet1"}'

# Whitelisting by PCI Device Vendor
# The example pattern limits matches to PCI cards with vendor id 8086␣
↪(Intel) and
# product id 10ed (82599 Virtual Function)
nova_pci_passthrough_whitelist: '{"vendor_id":"8086", "product_id":"10ed",
↪ "physical_network":"physnet1"}'

# Additionally, devices can be matched by their type, VF or PF, using the␣
↪dev_type parameter
# and type-VF or type-PF options
nova_pci_passthrough_whitelist: '{"vendor_id":"8086", "product_id":"10ed",
↪ "dev_type":"type-VF", physical_network":"physnet1"}'
```

It is recommended to use whitelisting by either the Linux device name (devname attribute) or by the PCI vendor and product id combination (`vendor_id` and `product_id` attributes)

3. Enable the SR-IOV ML2 plugin

   The conf override `neutron_plugin_type` variable defines the core ML2 plugin, and only one plugin can be defined at any given time. The conf override `neutron_plugin_types` variable can contain a list of additional ML2 plugins to load. Make sure that only compatible ML2 plugins are loaded at all times. The SR-IOV ML2 plugin is known to work with openvswitch (`ml2.ovs`) ML2 plugin.

```
neutron_plugin_types:
  - ml2.sriov
```

4. Execute the Neutron install playbook in order to update the configuration:

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible os-neutron-install.yml
# openstack-ansible os-nova-install.yml
```

5. Check Neutron SR-IOV agent state

   After the playbooks have finished configuring Neutron and Nova, the new Neutron Agent state can be verified with:

```
# neutron agent-list --agent_type 'NIC Switch agent'
+--------------------------------------+------------------+-----------+---
↪----+----------------+-------------------------+
| id                                   | agent_type       | host      |␣
↪alive | admin_state_up | binary                  |
+--------------------------------------+------------------+-----------+---
↪----+----------------+-------------------------+
| 3012ff0e-de35-447b-aff6-fdb55b04c518 | NIC Switch agent | compute01 | :-
↪)   | True           | neutron-sriov-nic-agent |
| bb0c0385-394d-4e72-8bfe-26fd020df639 | NIC Switch agent | compute02 | :-
↪)   | True           | neutron-sriov-nic-agent |
```

(continues on next page)

(continued from previous page)

```
+-------------------------------------+-----------------+----------+---
↪----+---------------+-------------------------+
```

Deployers can make changes to the SR-IOV nic agent default configuration options via the `neutron_sriov_nic_agent_ini_overrides` dict. Review the documentation on the conf override mechanism for more details.

CHAPTER

TWO

DEFAULT SCENARIO - OPEN VIRTUAL NETWORK (OVN)

2.1 Overview

By default, OpenStack-Ansible provisions Open Virtual Network (OVN) mechanism driver (ML2/OVN) instead of ML2/LXB or ML2/OVS since 2023.1 (Antelope) release. This driver offers the possibility of deploying virtual networks and routers using OVN with Open vSwitch, which replaces the agent-based models used by the legacy ML2/LXB and ML2/OVS architectures. This document outlines how to set it up in your environment.

2.1.1 Recommended Reading

Since this is an extension of the basic Open vSwitch scenario, it is worth reading that scenario to get some background. It is also recommended to be familiar with OVN and networking-ovn projects and their configuration.

- Scenario: Open vSwitch
- OVN Architecture Docs
- OpenStack Integration with OVN
- OVN OpenStack Tutorial

2.1.2 Prerequisites

- Open vSwitch >= 2.17.0

2.2 Inventory Architecture

While OVN itself supports many different configurations, Neutron and `networking-ovn` leverage specific functionality to provide virtual routing capabilities to an OpenStack-based Cloud.

OpenStack-Ansible separates OVN-related services and functions into three groups:

- neutron_ovn_northd
- neutron_ovn_controller
- neutron_ovn_gateway

The `neutron_ovn_northd` group is used to specify which host(s) will contain the OVN northd daemon responsible for translating the high-level OVN configuration into logical configuration consumable by daemons such as `ovn-controller`. In addition, these nodes host the OVN Northbound and OVN Southbound databases the `ovsdb-server` services. Members of this group are typically the controller nodes hosting the Neutron APIs (`neutron-server`).

The `neutron_ovn_controller` group is used to specify which host(s) will run the local `ovn-controller` daemon for OVN, which registers the local chassis and VIFs to the OVN Southbound database and converts logical flows to physical flows on the local hypervisor node. Members of this group are typically the compute nodes hosting virtual machine instances (`nova-compute`).

The `neutron_ovn_gateway` group is used to specify which hosts are eligible to act as an OVN Gateway Chassis, which is a node running `ovn-controller` that is capable of providing external (north/south) connectivity to the tenant traffic. This is essentially a node(s) capable of hosting the logical router performing SNAT and DNAT (Floating IP) translations. East/West traffic flow is not limited to a gateway chassis and is performed between the OVN chassis nodes themselves.

When planning out your architecture, it is important to determine early if you want to centralize OVN gateway chassis functions to a subset of nodes or across all compute nodes. Centralizing north/south

routing to a set of dedicated network or gateway nodes is reminiscent of the legacy network node model. Enabling all compute nodes as gateway chassis will narrow the failure domain and potential bottlenecks at the cost of ensuring the compute nodes can connect to the provider networks.

The following section will describe how to configure your inventory to meet certain deployment scenarios.

2.3 Deployment Scenarios

OpenStack-Ansible supports the following common deployment scenarios:

- Collapsed Network/Gateway Nodes
- Collapsed Compute/Gateway Nodes
- Standalone Gateway Nodes

In an OpenStack-Ansible deployment, infrastructure hosts are intended to run OVN northd-related services, while compute hosts are intended to run OVN controller-related services.

In `openstack_user_config.yml`, specify the hosts or aliases that will run the `ovn-northd` service(s), like so:

```
network-northd_hosts:
  infra01:
    ip: 172.25.1.11
  infra02:
    ip: 172.25.1.12
  infra03:
    ip: 172.25.1.13
```

Alternatively, an alias can be specified, like so:

```
network-northd_hosts: *infrastructure_hosts
```

It is up to the deployer to dictate which nodes are considered OVN Gateway Chassis nodes by using the `network-gateway_hosts` inventory group in `openstack_user_config.yml`.

In `openstack_user_config.yml`, specify the hosts or aliases that will run the `ovn-controller` service and act as an OVN Gateway Chassis, like so:

```
network-gateway_hosts:
  network-node1:
    ip: 172.25.1.21
  network-node2:
    ip: 172.25.1.22
  network-node3:
    ip: 172.25.1.23
```

Existing inventory aliases can also be used. In the following example, members of the `infrastructure_hosts` group are also network hosts and will serve as OVN Gateway Chassis nodes:

```
network-gateway_hosts: *infrastructure_hosts
```

In the following example, members of the `compute_hosts` group running the `ovn-controller` service will also serve as OVN Gateway Chassis nodes:

```
network-gateway_hosts: *compute_hosts
```

Lastly, specific hosts can also be targeted:

```
network-gateway_hosts:
  compute5:
    ip: 172.25.1.55
  compute10:
    ip: 172.25.1.60
  compute15:
    ip: 172.25.1.65
  compute20:
    ip: 172.25.1.70
  compute25:
    ip: 172.25.1.75
```

2.3.1 OpenStack-Ansible user variables

To deploy OpenStack-Ansible using the ML2/OVN mechanism driver, set the following user variables in the"/etc/openstack_deploy/user_variables.yml" file:

```
neutron_plugin_type: ml2.ovn

neutron_plugin_base:
  - ovn-router

neutron_ml2_drivers_type: "vlan,local,geneve,flat"
```

The overrides are instructing Ansible to deploy the OVN mechanism driver and associated OVN components. This is done by setting `neutron_plugin_type` to `ml2.ovn`.

The `neutron_plugin_base` override enables Neutron to use OVN for routing functions.

The `neutron_ml2_drivers_type` override provides support for all type drivers supported by OVN.

Provider network overrides can be specified on a global or per-host basis, and the following format can be used in `user_variables.yml` or per-host in `openstack_user_config.yml` or host vars.

Note

When `network_interface_mappings` are defined, the playbooks will attempt to connect the mapped interface to the respective OVS bridge. Omitting `network_interface_mappings` will require the operator to connect the interface to the bridge manually using the `ovs-vsctl add-port` command.

```
# When configuring Neutron to support geneve tenant networks and
# vlan provider networks the configuration may resemble the following:
neutron_provider_networks:
  network_types: "geneve"
  network_geneve_ranges: "1:1000"
  network_vlan_ranges: "public"
```

(continues on next page)

(continued from previous page)

```
  network_mappings: "public:br-publicnet"
  network_interface_mappings: "br-publicnet:bond1"

# When configuring Neutron to support only vlan tenant networks and
# vlan provider networks the configuration may resemble the following:
neutron_provider_networks:
  network_types: "vlan"
  network_vlan_ranges: "public:203:203,467:500"
  network_mappings: "public:br-publicnet"
  network_interface_mappings: "br-publicnet:bond1"

# When configuring Neutron to support multiple vlan provider networks
# the configuration may resemble the following:
neutron_provider_networks:
  network_types: "vlan"
  network_vlan_ranges: "public:203:203,467:500,private:101:200,301:400"
  network_mappings: "public:br-publicnet,private:br-privatenet"
  network_interface_mappings: "br-publicnet:bond1,br-privatenet:bond2"
```

(Optional) DVR or Distributed L3 routing

DVR will be used for floating IPs if the ovn / enable_distributed_floating_ip flag is configured to True in the neutron server configuration.

Create a group var file for neutron server `/etc/openstack_deploy/group_vars/neutron_server.yml`. It has to include:

```
# DVR/Distributed L3 routing support
neutron_ovn_distributed_fip: True
```

2.4 Useful Open Virtual Network (OVN) Commands

The following commands can be used to provide useful information about the state of Open vSwitch networking and configurations.

Note

Commands towards OVN Southbound and Northbound databases are expected to be run from `neutron_ovn_northd` hosts. OpenStack-Ansible places an openrc file named */root/ovnctl.rc* on these hosts. Once you `source` that file, required environment variables will be set to connect to the database. Alternatively, you can use `--no-leader-only` flag to connect to the local database only instead of the leader one (which is default).

Additional commands can be found in upstream OVN documentation and other resources listed on this page.

2.4.1 Check state of NB/SB DB clusters

The following ad-hoc command can be executed to find the current state and the leader of the NB/SB database:

```
ansible neutron_ovn_northd -m command -a "ovs-appctl -t /var/run/ovn/ovnnb_db.
↪ctl cluster/status OVN_Northbound"
ansible neutron_ovn_northd -m command -a "ovs-appctl -t /var/run/ovn/ovnsb_db.
↪ctl cluster/status OVN_Southbound"
```

2.4.2 Checking local state of Open vSwitch

The `ovs-vsctl list open_vswitch` command provides information about the `open_vswitch` table in the local Open vSwitch database and can be run from any network or compute host:

```
root@mnaio-controller1:~# ovs-vsctl list open_vswitch
_uuid               : 7f96baf2-d75e-4a99-bb19-ca7138fc14c2
bridges             : []
cur_cfg             : 1
datapath_types      : [netdev, system]
datapaths           : {}
db_version          : "8.3.0"
dpdk_initialized    : false
dpdk_version        : none
external_ids        : {hostname=mnaio-controller1, rundir="/var/run/
↪openvswitch", system-id="a67926f2-9543-419a-903d-23e2aa308368"}
iface_types         : [bareudp, erspan, geneve, gre, gtpu, internal,␣
↪ip6erspan, ip6gre, lisp, patch, stt, system, tap, vxlan]
manager_options     : []
next_cfg            : 1
other_config        : {}
ovs_version         : "2.17.2"
ssl                 : []
statistics          : {}
system_type         : ubuntu
system_version      : "20.04"
```

If you want to check only for only a specific field from the ovs-vsctl output, like applied interface mappings, you can select it in the following way:

```
root@mnaio-controller1:~# ovs-vsctl get open . external_ids:ovn-bridge-
↪mappings
"vlan:br-provider"
```

You can also get information about the agent UUID which will be stated in `openstack network agent list` output via similar command:

```
root@mnaio-controller1:~# ovs-vsctl get open . external_ids:system-id
"a67926f2-9543-419a-903d-23e2aa308368"
```

2.4.3 Get information from SouthBound database

Chassis details

The `ovn-sbctl show` command provides information related to southbound connections. If used outside the ovn_northd container, specify the connection details:

```
root@mnaio-controller1:~# ovn-sbctl show
Chassis "5335c34d-9233-47bd-92f1-fc7503270783"
    hostname: mnaio-compute1
    Encap geneve
        ip: "172.25.1.31"
        options: {csum="true"}
    Encap vxlan
        ip: "172.25.1.31"
        options: {csum="true"}
    Port_Binding "852530b5-1247-4ec2-9c39-8ae0752d2144"
Chassis "ff66288c-5a7c-41fb-ba54-6c781f95a81e"
    hostname: mnaio-compute2
    Encap vxlan
        ip: "172.25.1.32"
        options: {csum="true"}
    Encap geneve
        ip: "172.25.1.32"
        options: {csum="true"}
Chassis "cb6761f4-c14c-41f8-9654-16f3fc7cc7e6"
    hostname: mnaio-compute3
    Encap geneve
        ip: "172.25.1.33"
        options: {csum="true"}
    Encap vxlan
        ip: "172.25.1.33"
        options: {csum="true"}
    Port_Binding cr-lrp-022933b6-fb12-4f40-897f-745761f03186
```

You can get specific information about a chassis by providing either its *name*, where *name* is UUID of the agent (*external_ids:system-id* from the ovs-vsctl output), for example:

```
root@mnaio-controller1:~# ovn-sbctl list Chassis ff66288c-5a7c-41fb-ba54-
↪6c781f95a81e
_uuid               : b0b6ebec-1c64-417a-adb7-d383632a4c5e
encaps              : [a3ba78c3-df14-4144-81e0-e6379541bc89]
external_ids        : {}
hostname            : mnaio-compute2
name                : "ff66288c-5a7c-41fb-ba54-6c781f95a81e"
nb_cfg              : 0
other_config        : {ct-no-masked-label="true", datapath-type=system, fdb-
↪timestamp="true", iface-types="afxdp,afxdp-nonpmd,bareudp,erspan,geneve,gre,
↪gtpu,internal,ip6erspan,ip6gre,lisp,patch,srv6,stt,system,tap,vxlan", is-
↪interconn="false", mac-binding-timestamp="true", ovn-bridge-mappings=
↪"vlan:br-provider", ovn-chassis-mac-mappings="", ovn-cms-options="", ovn-ct-
↪lb-related="true", ovn-enable-lflow-cache="true", ovn-limit-lflow-cache="",␣
```

(continues on next page)

(continued from previous page)

```
↪ovn-memlimit-lflow-cache-kb="", ovn-monitor-all="false", ovn-trim-limit-
↪lflow-cache="", ovn-trim-timeout-ms="", ovn-trim-wmark-perc-lflow-cache="",␣
↪port-up-notif="true"}
transport_zones     : []
vtep_logical_switches: []
```

As you might see, `other_config` row also contains bridge-mapping, which can be fetched from the table similarly to the ovs-vsctl way:

```
root@mnaio-controller1:~# ovn-sbctl get Chassis ff66288c-5a7c-41fb-ba54-
↪6c781f95a81e other_config:ovn-bridge-mappings
"vlan:br-provider"
```

2.4.4 Get information from NorthBound database

Checking networks and ports

The `ovn-nbctl show` command provides information about networks, ports, and other objects known to OVN and demonstrates connectivity between the northbound database and neutron-server.

```
root@mnaio-controller1:~# ovn-nbctl show
switch 03dc4558-f83e-4531-b854-156292f1dbad (neutron-a6e65821-93e2-4521-9e31-
↪37c35d52d953) (aka project-tenant-network)
    port 852530b5-1247-4ec2-9c39-8ae0752d2144
        addresses: ["fa:16:3e:d2:af:bf 10.3.3.49"]
    port 624de478-7e75-472f-b867-e6f514790a81
        addresses: ["fa:16:3e:bf:c0:c3 10.3.3.3", "unknown"]
    port 1cca8ef3-d3c9-4307-a779-13348db5e647
        addresses: ["fa:16:3e:4a:67:ed 10.3.3.4", "unknown"]
    port 05e20b32-2933-414a-ba31-eac683d09ac2
        addresses: ["fa:16:3e:bd:5d:e8 10.3.3.5", "unknown"]
    port 5a2e35cb-178b-443b-9f15-4c6ec4db4ac7
        type: router
        router-port: lrp-5a2e35cb-178b-443b-9f15-4c6ec4db4ac7
    port 2d52a2bf-ab37-4a18-87bd-8808a99c67d3
        type: localport
        addresses: ["fa:16:3e:30:b4:a0 10.3.3.2"]
switch 3e03d5f1-4cfe-4c61-bd4c-8a661634d77b (neutron-b0b4017f-a9d1-4923-af35-
↪944b88b7a393) (aka flat-external-provider-network)
    port 022933b6-fb12-4f40-897f-745761f03186
        type: router
        router-port: lrp-022933b6-fb12-4f40-897f-745761f03186
    port 347a7d8d-fd0f-48be-be02-d603258f0a08
        addresses: ["fa:16:3e:f4:6a:17 192.168.25.5", "unknown"]
    port 29c83838-329d-4839-bddb-818c7e2e9bc7
        addresses: ["fa:16:3e:a3:48:a8 192.168.25.3", "unknown"]
    port 173c9ceb-4dd3-4268-aaa3-c7b0f693a557
        type: localport
        addresses: ["fa:16:3e:0c:37:ed 192.168.25.2"]
    port 7a0175fd-ac09-4466-b3d0-26f696e3769c
```

(continues on next page)

(continued from previous page)

```
        addresses: ["fa:16:3e:ad:19:c2 192.168.25.4", "unknown"]
    port provnet-525d3402-d582-49b4-b946-f28de8bbc615
        type: localnet
        addresses: ["unknown"]
router 5ebb0cdb-2026-4454-a32e-eb5425ae7296 (neutron-b0d6ca32-fda3-4fdc-b648-
↪82c8bee303dc) (aka project-router)
    port lrp-5a2e35cb-178b-443b-9f15-4c6ec4db4ac7
        mac: "fa:16:3e:3a:1c:bb"
        networks: ["10.3.3.1/24"]
    port lrp-022933b6-fb12-4f40-897f-745761f03186
        mac: "fa:16:3e:1f:cd:e9"
        networks: ["192.168.25.242/24"]
        gateway chassis: [cb6761f4-c14c-41f8-9654-16f3fc7cc7e6 ff66288c-5a7c-
↪41fb-ba54-6c781f95a81e 5335c34d-9233-47bd-92f1-fc7503270783]
    nat 79d8486c-8b5e-4d6c-a56f-9f0df115f77f
        external ip: "192.168.25.242"
        logical ip: "10.3.3.0/24"
        type: "snat"
    nat d338ccdf-d3c4-404e-b2a5-938d0c212e0d
        external ip: "192.168.25.246"
        logical ip: "10.3.3.49"
        type: "dnat_and_snat"
```

Checking Floating IPs

Floating IPs and Router SNAT are represented via NAT rules in the NB database, where FIP has type *dnat_and_snat*. You can fetch the list of NAT rules assigned to a specific router using the router name in the OVN database, which is formatted like `neutron-<UUID>`, where UUID is the UUID of the router in Neutron database. Command will look like this:

Note

Keep in mind, that GATEWAY_PORT will not be defined for dnat_and_snat rule when the external network of the router is on a geneve network and the router is bound to the chassis instead of its external port.

```
root@mnaio-controller1:~# ovn-nbctl lr-nat-list neutron-b0d6ca32-fda3-4fdc-
↪b648-82c8bee303dc
TYPE             GATEWAY_PORT          EXTERNAL_IP        EXTERNAL_PORT    ␣
↪LOGICAL_IP          EXTERNAL_MAC         LOGICAL_PORT
dnat_and_snat    lrp-16555e74-fbef-    192.168.25.246                       10.
↪3.3.49
snat                                   192.168.25.242                       10.
↪3.3.0/24
```

2.4.5 Logical Router operations

Locate Logical Router mapping to a Chassis

The mapping/location of the router to the gateway node can be established via logical ports of that router when the external network to which the router is connected happens to have either VLAN or FLAT type. For that you need to know the UUID of the external port attached to the router. The port name in the OVN database is constructed as `lrp-<UUID>`, where UUID is the Neutron port UUID. Given that an external network in the topic is named *public*, you can determine the gateway node in a following way:

```
root@mnaio-controller1:~# openstack port list --router b0d6ca32-fda3-4fdc-
↪b648-82c8bee303dc --network public -c ID
+--------------------------------------+
| ID                                   |
+--------------------------------------+
| 16555e74-fbef-4ecb-918c-2fb76bf5d42d |
+--------------------------------------+
root@mnaio-controller1:~# ovn-nbctl get Logical_Router_Port lrp-16555e74-fbef-
↪4ecb-918c-2fb76bf5d42d status:hosting-chassis
"5335c34d-9233-47bd-92f1-fc7503270783"
root@mnaio-controller1:~# ovn-sbctl get Chassis 5335c34d-9233-47bd-92f1-
↪fc7503270783 hostname
mnaio-compute1
root@mnaio-controller1:~# openstack network agent show 5335c34d-9233-47bd-
↪92f1-fc7503270783 -c host
+-------+-----------------------------+
| Field | Value                       |
+-------+-----------------------------+
| host  | mnaio-compute1              |
+-------+-----------------------------+
```

To list all gateway chassis on which the logical port is scheduled with their priorities, you can use:

```
root@mnaio-controller1:~# ovn-nbctl lrp-get-gateway-chassis lrp-16555e74-fbef-
↪4ecb-918c-2fb76bf5d42d | cut -d '_' -f 2
5335c34d-9233-47bd-92f1-fc7503270783     2
cb6761f4-c14c-41f8-9654-16f3fc7cc7e6     1
```

In cases when a Geneve network acts as the external network for the router, Logical Router will be pinned to the chassis instead of its LRP:

```
# ovn-nbctl --no-leader-only get Logical_Router neutron-b0d6ca32-fda3-4fdc-
↪b648-82c8bee303dc options
{always_learn_from_arp_request="false", chassis="5335c34d-9233-47bd-92f1-
↪fc7503270783", dynamic_neigh_routers="true", mac_binding_age_threshold="0"}
```

All LRPs of such routers will remain unbound.

Migrate a Logical Router between Chassis

In order to migrate active router logical port to another node, you can execute the following command:

```
root@mnaio-controller1:~# ovn-nbctl lrp-set-gateway-chassis lrp-16555e74-fbef-
↪4ecb-918c-2fb76bf5d42d ff66288c-5a7c-41fb-ba54-6c781f95a81e 10
```

Find all Logical Routers on the Chassis

With OVS/LXB ML2 plugins it used to be quite trivial to find all L3 Routers which are running on the network node and could be achieved with a single command, like `openstack router list --agent $L3_AGENT_UUID`.

Unfortunately, its a bit more complex with OVN as this command relies on L3 Router extension, which is not applicable for OVN.

Easy way to find out routers hosted on a specific Chassis with OpenStack CLI would be through listing ports binded to such chassis. For example:

```
for port in $(openstack port list --device-owner network:router_gateway --
↪host mnaio-compute1 -f value -c ID); do \
  openstack port show $port -c device_id -f value; \
done
```

However, this approach is slow and API-intensive, so can take a while to complete. Moreover, this will display only Neutron representation, which might not be absolutely accurate, as its OVN SouthBound database who is in charge of actually binding ports to Chassis. So it is way more efficient, but also more complex, to fetch this data directly from SB DB.

Firstly, we get a Chassis UUID from the SB DB and set it as `CHASSIS_ID` environmental variable. After that we can list all Logical Routers hosted on this Chassis:

```
CHASSIS=($(ovn-sbctl --no-headings --no-leader-only --columns=_uuid,name find␣
↪Chassis hostname=mnaio-compute1))
ovn-sbctl --format json --columns=external_ids find Port_Binding \
  chassis=${CHASSIS[0]} | \
  jq -r '.data[][][1][] | select(.[0] == "neutron:router_name") | .[1]' |␣
↪sort | uniq
```

Migrate all Logical Routers from Chassis

While OVN is smart enough and will move all routers from the Chassis on stopping `ovn-controller.service`, operators might want to explicitly empty out Chassis and remove it from any existing routers.

This section does incorporate results of previous two sections, so it is assumed being already in context of how Logical Router migration happens in OVN as well as challenges behind listing all affected Logical Router Ports bound to the chassis.

Below you may find an example of script which would perform migration of all Logical Routers from a specified by name Chassis to a selected destination.

```
SRC="mnaio-compute1"
DST="mnaio-compute2"
MAX_PRIO=10
```

(continues on next page)

(continued from previous page)

```
CHASSIS=($(ovn-sbctl --no-headings --no-leader-only --columns=_uuid,name find␣
↪Chassis hostname=${SRC}))
DEST_CHASSIS=($(ovn-sbctl --no-headings --no-leader-only --columns=_uuid,name␣
↪find Chassis hostname=${DST}))
LRPS=$(ovn-sbctl --no-leader-only --format json --columns=options find Port_
↪Binding chassis=${CHASSIS[0]} | jq -r '.data[][][1][] | select(.[0] ==
↪"distributed-port") | .[1]')
for lrp in ${LRPS}; do
    ovn-nbctl  --no-leader-only lrp-del-gateway-chassis $lrp ${DEST_
↪CHASSIS[1]}
    ovn-nbctl  --no-leader-only lrp-set-gateway-chassis $lrp ${DEST_
↪CHASSIS[1]} ${MAX_PRIO}
    ovn-nbctl  --no-leader-only lrp-del-gateway-chassis $lrp ${CHASSIS[1]}
    ovn-nbctl  --no-leader-only lrp-set-gateway-chassis $lrp ${CHASSIS[1]}
done
```

Now lets elaborate what this script actually does:

- Define source and destination Chassis by their hostnames via `SRC` and `DST`
- Define resulting priority on the new `DST` Chassis. The number should be higher or equal to the number of OVN Gateways in the deployment.
- Fetch Logical Router Ports (LRPs) bound to the `SRC` Chassis
- Remove `DST` Chassis from the list of applicable ones for LRP in order to avoid repeated records with conflicting priorities
- Re-add `DST` Chassis with `MAX_PRIO` priority to the LRP
- Remove `SRC` from the list of Chassis for LRP.
- Re-add `SRC` to the list of Chassis with the default (minimal) priority

Note

In environments with only two OVN gateway chassis, logical routers may not automatically migrate when a gateway node is powered off or loses connectivity. In practice, deployments with three or more gateway chassis show more reliable automatic router failover.

2.5 OVN database population

In case of OVN DB clustering failure and data loss as a result, you can always re-populate data in OVN SB/NB from stored state in Neutron database.

For that, you can execute the following:

```
root@mnaio-controller1:~# lxc-attach -n $(lxc-ls -1 | grep neutron-server)
root@mnaio-controller1-neutron-server-container-7510c8bf:/# source /etc/
↪openstack-release
root@mnaio-controller1-neutron-server-container-7510c8bf:/# /openstack/venvs/
↪neutron-${DISTRIB_RELEASE}/bin/neutron-ovn-db-sync-util --config-file /etc/
```

(continues on next page)

(continued from previous page)

```
↪neutron/neutron.conf --config-file /etc/neutron/plugins/ml2/ml2_conf.ini --
↪ovn-neutron_sync_mode repair
```

Command `neutron-ovn-db-sync-util` is also used during migration from OVS to OVN. For that, you need to supply `--ovn-neutron_sync_mode migrate` instead of *repair* as shown in the example above.

2.6 Notes

The `ovn-controller` service will check in as an agent and can be observed using the `openstack network agent list` command:

```
+--------------------------------------+-----------------------------+-------
↪-----------+-------------------+-------+-------+--------------------------
↪-+
| ID                                   | Agent Type                  | Host ␣
↪            | Availability Zone | Alive | State | Binary                   ␣
↪ |
+--------------------------------------+-----------------------------+-------
↪-----------+-------------------+-------+-------+--------------------------
↪-+
| 5335c34d-9233-47bd-92f1-fc7503270783 | OVN Controller Gateway agent | mnaio-
↪compute1    |                   | :-)   | UP    | ovn-controller           ␣
↪ |
| ff66288c-5a7c-41fb-ba54-6c781f95a81e | OVN Controller Gateway agent | mnaio-
↪compute2    |                   | :-)   | UP    | ovn-controller           ␣
↪ |
| cb6761f4-c14c-41f8-9654-16f3fc7cc7e6 | OVN Controller Gateway agent | mnaio-
↪compute3    |                   | :-)   | UP    | ovn-controller           ␣
↪ |
| 38206799-af64-589b-81b2-405f0cfcd198 | OVN Metadata agent           | mnaio-
↪compute1    |                   | :-)   | UP    | neutron-ovn-metadata-
↪agent |
| 9e9b49c7-dd00-5f58-a3f5-22dd01f562c4 | OVN Metadata agent           | mnaio-
↪compute2    |                   | :-)   | UP    | neutron-ovn-metadata-
↪agent |
| 72b1a6e2-4cca-570f-83a4-c05dcbbcc11f | OVN Metadata agent           | mnaio-
↪compute3    |                   | :-)   | UP    | neutron-ovn-metadata-
↪agent |
+--------------------------------------+-----------------------------+-------
↪-----------+-------------------+-------+-------+--------------------------
↪-+
```

CHAPTER

THREE

SCENARIO - USING OPEN VSWITCH

3.1 Overview

Operators can choose to utilize Open vSwitch instead of Linux Bridges for the neutron ML2 agent. This offers different capabilities and integration points with neutron. This document outlines how to set it up in your environment.

3.2 Recommended reading

We recommend that you read the following documents before proceeding:

- Neutron documentation on Open vSwitch OpenStack deployments: https://docs.openstack.org/liberty/networking-guide/scenario-classic-ovs.html
- Blog post on how OpenStack-Ansible works with Open vSwitch: https://medium.com/@travistruman/configuring-openstack-ansible-for-open-vswitch-b7e70e26009d

3.3 Prerequisites

All compute nodes must have bridges configured:

- `br-mgmt`
- `br-vlan` (optional - used for vlan networks)
- `br-vxlan` (optional - used for vxlan tenant networks)
- `br-storage` (optional - used for certain storage devices)

For more information see: https://docs.openstack.org/project-deploy-guide/openstack-ansible/newton/targethosts-networkconfig.html

These bridges may be configured as either a Linux Bridge (which would connect to the Open vSwitch controlled by neutron) or as an Open vSwitch.

3.4 Configuring bridges (Linux Bridge)

The following is an example of how to configure a bridge (example: `br-mgmt`) with a Linux Bridge on Ubuntu 16.04 LTS:

`/etc/network/interfaces`

```
auto lo
iface lo inet loopback

# Management network
auto eth0
iface eth0 inet manual

# VLAN network
auto eth1
iface eth1 inet manual

source /etc/network/interfaces.d/*.cfg
```

/etc/network/interfaces.d/br-mgmt.cfg

```
# OpenStack Management network bridge
auto br-mgmt
iface br-mgmt inet static
  bridge_stp off
  bridge_waitport 0
  bridge_fd 0
  bridge_ports eth0
  address MANAGEMENT_NETWORK_IP
  netmask 255.255.255.0
```

One br-<type>.cfg is required for each bridge. VLAN interfaces can be used to back the br-<type> bridges if there are limited physical adapters on the system.

3.5 Configuring bridges (Open vSwitch)

Another configuration method routes everything with Open vSwitch. The bridge (example: br-mgmt) can be an Open vSwitch itself.

The following is an example of how to configure a bridge (example: br-mgmt) with Open vSwitch on Ubuntu 16.04 LTS: *

/etc/network/interfaces

```
auto lo
iface lo inet loopback

source /etc/network/interfaces.d/*.cfg

# Management network
allow-br-mgmt eth0
iface eth0 inet manual
  ovs_bridge br-mgmt
  ovs_type OVSPort

# VLAN network
allow-br-vlan eth1
```

(continues on next page)

(continued from previous page)

```
iface eth1 inet manual
  ovs_bridge br-vlan
  ovs_type OVSPort
```

`/etc/network/interfaces.d/br-mgmt.cfg`

```
# OpenStack Management network bridge
auto br-mgmt
allow-ovs br-mgmt
iface br-mgmt inet static
  address MANAGEMENT_NETWORK_IP
  netmask 255.255.255.0
  ovs_type OVSBridge
  ovs_ports eth0
```

One `br-<type>.cfg` is required for each bridge. VLAN interfaces can be used to back the `br-<type>` bridges if there are limited physical adapters on the system.

3.6 OpenStack-Ansible user variables

Create a group var file for your network hosts `/etc/openstack_deploy/group_vars/network_hosts`. It has to include:

```
# Ensure the openvswitch kernel module is loaded
openstack_host_specific_kernel_modules:
  - name: "openvswitch"
    pattern: "CONFIG_OPENVSWITCH"
```

Specify provider network definitions in your `/etc/openstack_deploy/openstack_user_config.yml` that define one or more Neutron provider bridges and related configuration:

Note

Bridges specified here will be created automatically. If `network_interface` is defined, the interface will be placed into the bridge automatically.

```
- network:
    container_bridge: "br-provider"
    container_type: "veth"
    type: "vlan"
    range: "101:200,301:400"
    net_name: "physnet1"
    network_interface: "bond1"
    group_binds:
      - neutron_openvswitch_agent
- network:
    container_bridge: "br-provider2"
    container_type: "veth"
```

(continues on next page)

(continued from previous page)

```
    type: "vlan"
    range: "203:203,467:500"
    net_name: "physnet2"
    network_interface: "bond2"
    group_binds:
      - neutron_openvswitch_agent
```

When using `flat` provider networks, modify the network type accordingly:

```
- network:
    container_bridge: "br-publicnet"
    container_type: "veth"
    type: "flat"
    net_name: "flat"
    group_binds:
      - neutron_openvswitch_agent
```

Specify an overlay network definition in your `/etc/openstack_deploy/openstack_user_config.yml` that defines overlay network-related configuration:

Note

The bridge name should correspond to a pre-created Linux bridge or OVS bridge.

```
- network:
    container_bridge: "br-vxlan"
    container_type: "veth"
    container_interface: "eth10"
    ip_from_q: "tunnel"
    type: "vxlan"
    range: "1:1000"
    net_name: "vxlan"
    group_binds:
      - neutron_openvswitch_agent
```

Set the following user variables in your `/etc/openstack_deploy/user_variables.yml`:

```
neutron_plugin_type: ml2.ovs

neutron_ml2_drivers_type: "flat,vlan,vxlan"
neutron_plugin_base:
  - router
  - metering
```

The overrides are instructing Ansible to deploy the OVS mechanism driver and associated OVS components. This is done by setting `neutron_plugin_type` to `ml2.ovs`.

The `neutron_ml2_drivers_type` override provides support for all common type drivers supported by OVS.

The `neutron_plugin_base` is used to defined list of plugins that will be enabled.

If provider network overrides are needed on a global or per-host basis, the following format can be used in `user_variables.yml` or per-host in `openstack_user_config.yml`.

Note

These overrides are not normally required when defining global provider networks in the `openstack_user_config.yml` file.

```
# When configuring Neutron to support vxlan tenant networks and
# vlan provider networks the configuration may resemble the following:
neutron_provider_networks:
  network_types: "vxlan"
  network_vxlan_ranges: "1:1000"
  network_vlan_ranges: "physnet1:102:199"
  network_mappings: "physnet1:br-provider"
  network_interface_mappings: "br-provider:bond1"

# When configuring Neutron to support only vlan tenant networks and
# vlan provider networks the configuration may resemble the following:
neutron_provider_networks:
  network_types: "vlan"
  network_vlan_ranges: "physnet1:102:199"
  network_mappings: "physnet1:br-provider"
  network_interface_mappings: "br-provider:bond1"

# When configuring Neutron to support multiple vlan provider networks
# the configuration may resemble the following:
neutron_provider_networks:
  network_types: "vlan"
  network_vlan_ranges: "physnet1:102:199,physnet2:2000:2999"
  network_mappings: "physnet1:br-provider,physnet2:br-provider2"
  network_interface_mappings: "br-provider:bond1,br-provider2:bond2"

# When configuring Neutron to support multiple vlan and flat provider
# networks the configuration may resemble the following:
neutron_provider_networks:
  network_flat_networks: "*"
  network_types: "vlan"
  network_vlan_ranges: "physnet1:102:199,physnet2:2000:2999"
  network_mappings: "physnet1:br-provider,physnet2:br-provider2"
  network_interface_mappings: "br-provider:bond1,br-provider2:bond2"
```

3.7 Open Virtual Switch (OVS) commands

The following commands can be used to provide useful information about the state of Open vSwitch networking and configurations.

The `ovs-vsctl show` command provides information about the virtual switches and connected ports currently configured on the host:

```
root@infra01:~# ovs-vsctl show
4ef304ff-b803-4d09-95f5-59a076323949
    Manager "ptcp:6640:127.0.0.1"
        is_connected: true
    Bridge br-int
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port "tap2e7e0507-e4"
            tag: 2
            Interface "tap2e7e0507-e4"
                type: internal
        Port int-br-vlan
            Interface int-br-vlan
                type: patch
                options: {peer=phy-br-provider}
        Port br-int
            Interface br-int
                type: internal
        Port "tap7796ab3d-e9"
            tag: 5
            Interface "tap7796ab3d-e9"
                type: internal
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
    Bridge br-tun
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        Port "vxlan-ac1df015"
            Interface "vxlan-ac1df015"
                type: vxlan
                options: {df_default="true", in_key=flow, local_ip="172.29.
↪240.20", out_key=flow, remote_ip="172.29.240.21"}
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
        Port "vxlan-ac1df017"
            Interface "vxlan-ac1df017"
                type: vxlan
                options: {df_default="true", in_key=flow, local_ip="172.29.
↪240.20", out_key=flow, remote_ip="172.29.240.23"}
        Port br-tun
            Interface br-tun
                type: internal
    Bridge br-provider
        Controller "tcp:127.0.0.1:6633"
```

(continues on next page)

(continued from previous page)

```
            is_connected: true
        fail_mode: secure
        Port "ens192"
            Interface "ens192"
        Port br-provider
            Interface br-provider
                type: internal
        Port phy-br-provider
            Interface phy-br-provider
                type: patch
                options: {peer=int-br-provider}
    ovs_version: "2.10.0"
```

Additional commands can be found in upstream Open vSwitch documentation.

3.8 Notes

The `neutron-openvswitch-agent` service will check in as an agent and can be observed using the `openstack network agent list` command:

```
root@infra01-utility-container-ce1509fd:~# openstack network agent list --
↪agent-type open-vswitch
+--------------------------------------+--------------------+-------------+---
↪---------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host        |␣
↪Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+-------------+---
↪---------------+-------+-------+---------------------------+
| 4dcef710-ec0c-4925-a940-dc319cd6849f | Open vSwitch agent | compute03   |␣
↪None              | :-)   | UP    | neutron-openvswitch-agent |
| 5e1f8670-b90e-49c3-84ff-e981aeccb171 | Open vSwitch agent | compute02   |␣
↪None              | :-)   | UP    | neutron-openvswitch-agent |
| 78746672-d77a-4d8a-bb48-f659251fa246 | Open vSwitch agent | compute01   |␣
↪None              | :-)   | UP    | neutron-openvswitch-agent |
| eebab5da-3ef5-4582-84c5-f29e2472a44a | Open vSwitch agent | infra01     |␣
↪None              | :-)   | UP    | neutron-openvswitch-agent |
+--------------------------------------+--------------------+-------------+---
↪---------------+-------+-------+---------------------------+
```

CHAPTER

FOUR

SCENARIO - USING OPEN VSWITCH W/ ASAP ^{2} (DIRECT MODE)

4.1 Overview

With appropriate hardware, operators can choose to utilize ASAP ^{2}-accelerated Open vSwitch instead of unaccelerated Open vSwitch for the Neutron virtual network infrastructure. ASAP ^{2} technology offloads packet processing onto hardware built into the NIC rather than using the CPU of the host. It requires careful consideration and planning before implementing. This document outlines how to set it up in your environment.

Note

ASAP ^{2} is a proprietary feature provided with certain Mellanox NICs, including the ConnectX-5 and ConnectX-6. Future support is not guaranteed. This feature is considered *EXPERIMENTAL* and should not be used for production workloads. There is no guarantee of upgradability or backwards compatibility.

Note

Hardware offloading is not yet compatible with the `openvswitch` firewall driver. To ensure flows are offloaded, port security must be disabled. Information on disabling port security is discussed later in this document.

4.2 Recommended reading

This guide is a variation of the standard Open vSwitch and SR-IOV deployment guides available at:

- https://docs.openstack.org/openstack-ansible-os_neutron/latest/app-openvswitch.html
- https://docs.openstack.org/openstack-ansible-os_neutron/latest/configure-network-services.html#sr-iov-support-optional

The following resources may also be helpful:

- https://docs.openstack.org/neutron/latest/admin/config-sriov.html
- https://docs.openstack.org/neutron/latest/admin/config-ovs-offload.html
- https://www.mellanox.com/related-docs/prod_software/ASAP2_Hardware_Offloading_for_vSwitches_User_Manual_v4.4.pdf
- https://docs.nvidia.com/networking/pages/viewpage.action?pageId=61869597

4.3 Prerequisites

To enable SR-IOV and PCI passthrough capabilities on a Linux platform, ensure that VT-d/VT-x are enabled for Intel processors and AMD-V/AMD-Vi are enabled for AMD processors. Such features are typically enabled in the BIOS.

On an Intel platform, the following kernel parameters are required and can be added to the GRUB configuration:

```
GRUB_CMDLINE_LINUX="... iommu=pt intel_iommu=on"
```

On an AMD platform, use these parameters instead:

```
GRUB_CMDLINE_LINUX="... iommu=pt amd_iommu=on"
```

Update GRUB and reboot the host(s).

SR-IOV provides virtual functions (VFs) that can be presented to instances as network interfaces and are used in lieu of tuntap interfaces. Configuration of VFs is outside the scope of this guide. The following links may be helpful:

- https://community.mellanox.com/s/article/getting-started-with-mellanox-asap-2
- https://community.mellanox.com/s/article/howto-configure-sr-iov-for-connectx-4-connectx-5-with-kvmethernet-x

4.4 Deployment

Configure your networking according the Open vSwitch implementation docs:

- Scenario - Using Open vSwitch

Note

At this time, only a single (non-bonded) interface is supported.

An example provider network configuration has been provided below:

```
- network:
    container_bridge: "br-provider"
    container_type: "veth"
    type: "vlan"
    range: "700:709"
    net_name: "physnet1"
    network_interface: "ens4f0"
    group_binds:
      - neutron_openvswitch_agent
```

Add a `nova_pci_passthrough_whitelist` entry to `user_variables.yml`, where `devname` is the name of the interface connected to the provider bridge and `physical_network` is the name of the provider network.

```
nova_pci_passthrough_whitelist: '{"devname":"ens4f0","physical_network":
↪"physnet1"}'
```

Note

In the respective network block configured in `openstack_user_config.yml`, `devname` corresponds to `network_interface` and `physical_network` corresponds to `net_name`.

To enable the `openvswitch` firewall driver rather than the default `iptables_hybrid` firewall driver, add the following overrides to `user_variables.yml`:

```
neutron_ml2_conf_ini_overrides:
  securitygroup:
    firewall_driver: openvswitch
neutron_openvswitch_agent_ini_overrides:
  securitygroup:
    firewall_driver: openvswitch
```

Note

Hardware-offloaded flows are **not** activated for ports utilizing security groups or port security. Be sure to disable port security *and* security groups on individual ports or networks when hardware offloading is required.

Once the OpenStack cluster is configured, start the OpenStack deployment as listed in the OpenStack-Ansible Install guide by running all playbooks in sequence on the deployment host.

4.5 Post-Deployment

Once the deployment is complete, create the VFs that will be used for SR-IOV. In this example, the physical function (PF) is `ens4f0`. It will simultaneously be connected to the Neutron provider bridge `br-provider`.

1. On each compute node, determine the maximum number of VFs a PF can support:

```
# cat /sys/class/net/ens4f0/device/sriov_totalvfs
```

Note

To adjust `sriov_totalvfs` please refer to Mellanox documentation.

2. On each compute node, create the VFs:

```
# echo '8' > /sys/class/net/ens4f0/device/sriov_numvfs
```

4.5.1 Configure Open vSwitch hardware offloading

1. Unbind the VFs from the Mellanox driver:

```
# for vf in `grep PCI_SLOT_NAME /sys/class/net/ens4f0/device/virtfn*/uevent |␣
↪cut -d'=' -f2`
  do
    echo $vf > /sys/bus/pci/drivers/mlx5_core/unbind
  done
```

2. Enable the switch in the NIC:

```
# PCI_ADDR=`grep PCI_SLOT_NAME /sys/class/net/ens4f0/device/uevent | sed 's:.
↪*PCI_SLOT_NAME=::'`
# devlink dev eswitch set pci/$PCI_ADDR mode switchdev
```

3. Enable hardware offload filters with TC:

```
# ethtool -K ens4f0 hw-tc-offload on
```

4. Rebind the VFs to the Mellanox driver:

```
# for vf in `grep PCI_SLOT_NAME /sys/class/net/ens4f0/device/virtfn*/uevent |␣
↪cut -d'=' -f2`
  do
    echo $vf > /sys/bus/pci/drivers/mlx5_core/bind
  done
```

5. Enable hardware offloading in OVS:

```
# ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
# ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
```

6. Restart Open vSwitch

```
# systemctl restart openvswitch-switch
```

7. Restart the Open vSwitch agent

```
# systemctl restart neutron-openvswitch-agent
```

8. Restart the Nova compute service

```
# systemctl restart nova-compute
```

Warning

Changes to `sriov_numvfs` as well as the built-in NIC switch will not persist a reboot and must be performed every time the server is started.

4.6 Verify operation

To verify operation of hardware-offloaded Open vSwitch, you must create a virtual machine instance using an image with the proper network drivers.

The following images are known to contain working drivers:

- Ubuntu 18.04 LTS (Bionic)
- Ubuntu 20.04 LTS (Focal)
- Centos 8 Stream
- Centos 9 Stream

Before creating an instance, a Neutron port must be created that has the following characteristics:

`--vnic-type direct --binding-profile '{"capabilities": ["switchdev"]}'`

To ensure flows are offloaded, disable port security with the `--disable-port-security` argument.

An example of the full command can be seen here:

```
# openstack port create \
  --network <network> \
  --vnic-type direct --binding-profile '{"capabilities": ["switchdev"]}' \
  --disable-port-security \
  <name>
```

The port can then be attached to the instance at boot. Once booted, the port will be updated to reflect the PCI address of the corresponding virtual function:

```
root@aio1-utility-container-8c0b0916:~# openstack port show -c binding_
↪profile testport2
+-----------------+-------------------------------------------------------------
↪----------------------------------------------------+
| Field           | Value                                                       ␣
↪                                                    |
+-----------------+-------------------------------------------------------------
↪----------------------------------------------------+
| binding_profile | capabilities='[u'switchdev']', pci_slot='0000:21:00.6',␣
↪pci_vendor_info='15b3:1016', physical_network='physnet1' |
+-----------------+-------------------------------------------------------------
↪----------------------------------------------------+
```

4.6.1 Observing traffic

From the compute node, perform a packet capture on the representor port that corresponds to the virtual function attached to the instance. In this example, the interface is `eth1`.

```
root@compute1:~# tcpdump -nnn -i eth1 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
```

Perform a ping from another host and observe the traffic at the representor port:

```
root@infra2:~# ping 192.168.88.151 -c5
PING 192.168.88.151 (192.168.88.151) 56(84) bytes of data.
64 bytes from 192.168.88.151: icmp_seq=1 ttl=64 time=48.3 ms
64 bytes from 192.168.88.151: icmp_seq=2 ttl=64 time=1.52 ms
64 bytes from 192.168.88.151: icmp_seq=3 ttl=64 time=0.586 ms
64 bytes from 192.168.88.151: icmp_seq=4 ttl=64 time=0.688 ms
64 bytes from 192.168.88.151: icmp_seq=5 ttl=64 time=0.775 ms

--- 192.168.88.151 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4045ms
rtt min/avg/max/mdev = 0.586/10.381/48.335/18.979 ms

root@compute1:~# tcpdump -nnn -i eth1 icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
19:51:09.684957 IP 192.168.88.254 > 192.168.88.151: ICMP echo request, id␣
↪11168, seq 1, length 64
19:51:09.685448 IP 192.168.88.151 > 192.168.88.254: ICMP echo reply, id 11168,
↪ seq 1, length 64
```

When offloading is handled in the NIC, only the first packet(s) of the flow will be visible in the packet capture.

The following command can be used to dump flows in the kernel datapath:

```
# ovs-appctl dpctl/dump-flows type=ovs
```

The following command can be used to dump flows that are offloaded:

```
# ovs-appctl dpctl/dump-flows type=offloaded
```

CHAPTER

FIVE

SCENARIO - USING OPEN VSWITCH WITH DVR

5.1 Overview

Operators can choose to utilize Open vSwitch with Distributed Virtual Routing (DVR) instead of Linux Bridges or plain Open vSwitch for the neutron ML2 agent. This offers the possibility to deploy virtual routing instances outside the usual neutron networking node. This document outlines how to set it up in your environment.

5.2 Recommended reading

This guide is a variation of the standard Open vSwitch deployment guide available at:

https://docs.openstack.org/openstack-ansible-os_neutron/latest/app-openvswitch.html

We recommend that you read the following documents before proceeding:

- Neutron documentation on Open vSwitch DVR OpenStack deployments: https://docs.openstack.org/neutron/latest/admin/deploy-ovs-ha-dvr.html
- Blog post on how OpenStack-Ansible works with Open vSwitch: https://trumant.github.io/openstack-ansible-dvr-with-openvswitch.html

5.3 Prerequisites

Configure your networking according the Open vSwitch setup:

- Scenario - Using Open vSwitch https://docs.openstack.org/openstack-ansible-os_neutron/latest/app-openvswitch.html

5.4 OpenStack-Ansible user variables

Create a group var file for your network hosts `/etc/openstack_deploy/group_vars/network_hosts`. It has to include:

```
# Ensure the openvswitch kernel module is loaded
openstack_host_specific_kernel_modules:
  - name: "openvswitch"
    pattern: "CONFIG_OPENVSWITCH"
```

Specify provider network definitions in your `/etc/openstack_deploy/openstack_user_config.yml` that define one or more Neutron provider bridges and related configuration:

Note

Bridges specified here will be created automatically. If `network_interface` is defined, the interface will be placed into the bridge automatically.

```
- network:
    container_bridge: "br-provider"
    container_type: "veth"
    type: "vlan"
    range: "101:200,301:400"
    net_name: "physnet1"
    network_interface: "bond1"
    group_binds:
      - neutron_openvswitch_agent
- network:
    container_bridge: "br-provider2"
    container_type: "veth"
    type: "vlan"
    range: "203:203,467:500"
    net_name: "physnet2"
    network_interface: "bond2"
    group_binds:
      - neutron_openvswitch_agent
```

When using `flat` provider networks, modify the network type accordingly:

```
- network:
    container_bridge: "br-provider"
    container_type: "veth"
    type: "flat"
    net_name: "flat"
    group_binds:
      - neutron_openvswitch_agent
```

Specify an overlay network definition in your `/etc/openstack_deploy/openstack_user_config.yml` that defines overlay network-related configuration:

Note

The bridge name should correspond to a pre-created Linux bridge or OVS bridge.

```
- network:
    container_bridge: "br-vxlan"
    container_type: "veth"
    container_interface: "eth10"
    ip_from_q: "tunnel"
    type: "vxlan"
    range: "1:1000"
    net_name: "vxlan"
```

(continues on next page)

(continued from previous page)

```
    group_binds:
      - neutron_openvswitch_agent
```

Set the following user variables in your `/etc/openstack_deploy/user_variables.yml`:

Note

The only difference a DVR deployment and the standard Open vSwitch deployment is the setting of the respective `neutron_plugin_type`.

```
neutron_plugin_type: ml2.ovs.dvr

neutron_ml2_drivers_type: "flat,vlan,vxlan"
neutron_plugin_base:
  - router
  - metering
```

The overrides are instructing Ansible to deploy the OVS mechanism driver and associated OVS and DVR components. This is done by setting `neutron_plugin_type` to `ml2.ovs.dvr`.

The `neutron_ml2_drivers_type` override provides support for all common type drivers supported by OVS.

For additional information regarding provider network overrides and other configuration options, please refer to the standard Open vSwitch deployment available at:

https://docs.openstack.org/openstack-ansible-os_neutron/latest/app-openvswitch.html

CHAPTER

SIX

SCENARIO - USING OPEN VSWITCH W/ DPDK

6.1 Overview

Operators can choose to utilize DPDK-accelerated Open vSwitch instead of unaccelerated Open vSwitch or Linux Bridges for the Neutron virtual network infrastructure. This architecture is best suited for NFV workloads and requires careful consideration and planning before implementing. This document outlines how to set it up in your environment.

Warning

The current implementation of DPDK in OpenStack-Ansible is experimental and not production ready. There is no guarantee of upgradability or backwards compatibility between releases.

6.2 Recommended reading

We recommend that you read the following documents before proceeding:

- Neutron with Open vSwitch Scenario: https://docs.openstack.org/openstack-ansible-os_neutron/latest/app-openvswitch.html
- Open vSwitch with DPDK datapath: https://docs.openstack.org/neutron/latest/admin/config-ovs-dpdk.html
- Getting the best performance from DPDK: https://doc.dpdk.org/guides/linux_gsg/nic_perf_intel_platform.html
- OpenStack documentation on hugepages: https://docs.openstack.org/nova/latest/admin/huge-pages.html

6.3 Prerequisites

To enable DPDK on a Linux platform, ensure that VT-d/VT-x are enabled for Intel processors and AMD-V/AMD-Vi are enabled for AMD processors. Such features are typically enabled in the BIOS.

On an Intel platform, the following kernel parameters are required and can be added to the GRUB configuration:

```
GRUB_CMDLINE_LINUX="... iommu=pt intel_iommu=on"
```

On an AMD platform, use these parameters instead:

```
GRUB_CMDLINE_LINUX="... iommu=pt amd_iommu=on"
```

Update GRUB and reboot the host(s).

Hugepages are required for DPDK. Instances leveraging DPDK-accelerated Open vSwitch must be configured to use hugepages by way of flavor attributes. Those attributes and the configuration of hugepages are described in this guide.

CPU frequency should be set to maximum for optimal performance. Many hardware vendors set the energy saving properties in the BIOS that may need to be modified. Changing the CPU frequency using `cpufreq` or similar utilities to `performance` from `ondemand` is recommended.

Note

The playbooks currently only support a single NIC interface for DPDK. Multiple ports per NIC are not yet supported but may be at a later time. This guide assumes the NIC is bound to NUMA node0, but the instructions can be modified for NICs bound to other NUMA nodes..

6.4 NUMA topology

Non-uniform memory access (NUMA) is a computer memory design used in multiprocessing. This guide cannot go into great depths about NUMA architecture. However, there are some configurations to be made that rely on the operator understanding NUMA characteristics of compute nodes hosting workloads using DPDK-accelerated Open vSwitch.

To view the NUMA topology of a particular system, use the `numactl` command shown here:

```
root@compute1:~# numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 16 17 18 19 20 21 22 23
node 0 size: 48329 MB
node 0 free: 31798 MB
node 1 cpus: 8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31
node 1 size: 48379 MB
node 1 free: 25995 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

The NUMA topology presented here corresponds to a host with 2x Intel Xeon 2450L processors with 96 GB of total RAM. The RAM is evenly split between the two NUMA nodes. Each CPU has 8 cores. With hyperthreading enabled, there are 16 threads per CPU for a total of 32 threads or cores presented to the operating system. It just so happens that this two-socket system has one NUMA node per socket, however, that will not always be the case. Consult your systems documentation for information unique to your system.

The first eight cores/cpus in the list for a given NUMA node can be considered physical cores in the CPU. For NUMA node0, this would be cores 0-7. The other eight cores, 16-23, are considered virtual sibling cores and are presented when hyperthreading is enabled. The physical-to-virtual mapping can be determined with the following commands:

```
root@compute1:~# for cpu in {0..7}; do cat /sys/devices/system/cpu/"cpu"$cpu/
↪topology/thread_siblings_list; done
0,16
1,17
2,18
3,19
4,20
5,21
6,22
7,23

root@compute1:~# for cpu in {8..15}; do cat /sys/devices/system/cpu/"cpu"$cpu/
↪topology/thread_siblings_list; done
8,24
9,25
10,26
11,27
12,28
13,29
14,30
15,31
```

A PCI slot typically corresponds to a single NUMA node. For optimal performance, a DPDK NIC and any instance utilizing the NIC should be restricted to the same NUMA node and its respective memory. Ensuring this behavior requires the use of flavors, host aggregates, and special kernel parameters and Open vSwitch/DPDK configuration settings.

In this example, a single 10G NIC installed in PCI slot 2 is bound to NUMA node0. Ideally, any instances utilizing the NIC would be limited to cores and memory associated with NUMA node0. This means cores 0-7 and 16-23, and up to 48GB of RAM. In reality, however, some cores and RAM from NUMA node0 will be reserved and made unavailable to instances. In addition, cores 8-15 and 24-31 associated with NUMA node1 should be made unavailable to instances. The configuration to do just that will be covered later in this guide.

It is considered good practice to reserve a single physical core and its respective virtual sibling from each NUMA node for normal (non-DPDK) operating system functions. In addition, at least one physical core (and sibling) from each NUMA node should be reserved for DPDK poll mode driver (PMD) functions, even when a NIC(s) is bound to a single NUMA node. The remaining cores can be reserved for virtual machine instances.

In this example, the breakdown would resemble the following:

```
` | Reserved Cores | Purpose | node0 | node1 | | ---------------------- |
--------------------- | --------- | ----- | | 0,8,16,24 | Host Operating
System | 0,16 | 8,24 | | 1,9,17,25 | DPDK PMDs | 1,17 | 9,25 | | 2-7,18-23
| Virtual Machines | 2-7,18-23 | N/A | `
```

The variables are overrides used to define this configuration are discussed in the following sections.

6.5 Hugepage configuration

DPDK requires the configuration of hugepages, which is a mechanism by which the Linux kernel can partition and address larger amounts of memory beyond the basic page unit (4096 bytes). Huge pages are blocks of contiguous memory that commonly come in 2MB and 1G sizes. The page tables used by 2MB pages are suitable for managing multiple gigabytes of memory, whereas the page tables of 1GB pages are preferred for scaling to terabytes of memory. DPDK requires the use of 1GB pages.

A typical x86 system will have a Huge Page Size of 2048 kBytes (2MB). The default huge page size may be found by looking at the output of /proc/meminfo:

```
# cat /proc/meminfo | grep Hugepagesize
Hugepagesize: 2048 kB
```

The number of Hugepages can be allocated at runtime by modifying `/proc/sys/vm/nr_hugepages` or by using the `sysctl` command.

To view the current setting using the `/proc` entry:

```
# cat /proc/sys/vm/nr_hugepages
0
```

To view the current setting using the `sysctl` command:

```
# sysctl vm.nr_hugepages
vm.nr_hugepages = 0
```

To set the number of huge pages using `/proc` entry:

```
# echo 5 > /proc/sys/vm/nr_hugepages
```

To set the number of hugepages using sysctl:

```
# sysctl -w vm.nr_hugepages=5
vm.nr_hugepages = 5
```

It may be necessary to reboot to be able to allocate the number of hugepages that is needed. This is due to hugepages requiring large areas of contiguous physical memory.

When 1G hugepages are used, they must be configured at boot time. The amount of 1G hugepages that should be created will vary based on a few factors, including:

- The total amount of RAM available in the system
- The amount of RAM required for the planned number of instances
- The number of NUMA nodes that will be used

The NUMA topology presented here corresponds to a host with 2x Intel Xeon 2450L processors with 96GB of total RAM. The RAM is evenly split between the two NUMA nodes. A DPDK NIC will be associated with a single NUMA node, and for optimal performance any instance utilizing the DPDK NIC should be limited to the same cores and memory associated with the NUMA node. On this example system, both DPDK and instances can only utilize *up to* the 48GB of RAM associated with NUMA node0, though some of that RAM will be utilized by the OS and other tasks.

Of the 48GB of RAM available on NUMA node0, 32GB will be reserved for 1GB hugepages to be consumed by DPDK PMDs and instances. Configuring hugepages using kernel parameters results in

the defined number of hugepages to be split evenly across NUMA nodes. With the following kernel parameter, each NUMA node will be assigned 32x 1G hugepages:

```
GRUB_CMDLINE_LINUX="... hugepagesz=1G hugepages=64"
```

Hugepages can be adjusted at runtime if necessary, but doing so is outside the scope of this guide.

6.6 OpenStack-Ansible variables and overrides

The ability to pin instances to certain cores is not new, and can be accomplished using the `vcpu_pin_set` override seen here:

```
nova_nova_conf_overrides:
  DEFAULT:
    vcpu_pin_set: 2-7,18-23
```

This change can be added to the `user_overrides.yml` file for global implementation, or to individual nodes in the `openstack_user_config.yml` file as shown here:

```
compute_hosts:
  compute01:
    ip: 172.29.236.200
    container_vars:
      ...
      nova_nova_conf_overrides:
        DEFAULT:
          vcpu_pin_set: 2-7,18-23
```

Cores reserved for host operating system functions (non-DPDK) must be converted to a hexidecimal mask and defined using the `ovs_dpdk_lcore_mask` override. To convert to a hex mask you must first establish the binary mask of chosen cores using the following table:

```
` | 31 | 30 | . | 24 | 23 | . | 17 | 16 | 15 | . | 9 | 8 | 7 | . | 1 | 0 | |
-- | -- | - | -- | -- | - | -- | -- | -- | - | -- | -- | -- | - | -- | -- | |
0 | 0 | . | 1 | 0 | . | 0 | 1 | 0 | . | 0 | 1 | 0 | . | 0 | 1 | `
```

The ellipses represent cores not shown. The binary mask for cores 0,8,16,24 can be determined in the following way:

```
00000001000000010000000100000001
```

The hexidecimal representation of that binary value is `0x1010101`. Set the `ovs_dpdk_lcore_mask` override accordingly in the `user_variables.yml` file or `openstack_user_config.yml`:

```
ovs_dpdk_lcore_mask: 1010101
```

The mask for cores 1,9,17,25 reserved for DPDK PMDs can be determined in a similar fashion. The table would resemble the following:

```
` | 31 | 30 | . | 25 | 24 | . | 17 | 16 | 15 | . | 9 | 8 | 7 | . | 1 | 0 | |
-- | -- | - | -- | -- | - | -- | -- | -- | - | -- | -- | -- | - | -- | -- | |
0 | 0 | . | 1 | 0 | . | 1 | 0 | 0 | . | 1 | 0 | 0 | . | 1 | 0 | `
```

The ellipses represent cores not shown. The binary mask for cores 1,9,17,254 can be determined in the following way:

```
00000010000000100000001000000010
```

The hexidecimal representation of that binary value is `0x2020202`. Set the `ovs_dpdk_pmd_cpu_mask` override accordingly in the `user_variables.yml` file or `openstack_user_config.yml`:

```
ovs_dpdk_pmd_cpu_mask: 2020202
```

Additional variables should be set, including:

- ovs_dpdk_driver
- ovs_dpdk_pci_addresses
- ovs_dpdk_socket_mem

The default value for `ovs_dpdk_driver` is `vfio-pci`. Overrides can be set globally or on a per-host basis.

Note

Please consult the DPDK Network Interface Controller Driver documentation for more inforation on supported network drivers for DPDK.

The value for `ovs_dpdk_pci_addresses` is the PCI bus address of the NIC port(s) associated with the DPDK NIC. In this example, the DPDK NIC is identified as address `0000:03:00`. The individual interfaces are `0000:03:00.0` and `0000:03:00.1`, respectively. The variable `ovs_dpdk_pci_addresses` is a list, and both values can be defined like so:

```
ovs_dpdk_pci_addresses:
  - 0000:03:00.0
  - 0000:03:00.1
```

The value for `ovs_dpdk_socket_mem` will vary based on the number of NUMA nodes, number of NICs per NUMA node, and the MTU. The default value assumes a single NUMA node and associates a single 1G hugepage to DPDK that can handle a 1500 MTU. When multiple NUMA nodes are available, even with a single NIC, the following should be set:

```
ovs_dpdk_socket_mem: "1024,1024"
```

For systems using a single NUMA node of a dual-NUMA system and a 9000 MTU, the following can be set:

```
ovs_dpdk_socket_mem: "3072,1024"
```

Determing socket memory required involves calculations that are out of the scope of this guide.

6.7 Flavor configuration

Instances that connect to a DPDK-accelerated Open vSwitch must be configured to utilize large (1G) hugepages by way of custom flavor attributes.

The `hw:mem_page_size` property can be set on a new or existing flavor to enable this functionality:

```
openstack flavor set m1.small --property hw:mem_page_size=large
```

NOTE: If small page size is used, or no page size is set, the interface may appear in the instance but will not be functional.

6.8 OpenStack-Ansible user variables

Create a group var file for your network hosts `/etc/openstack_deploy/group_vars/network_hosts`. It has to include:

```
# Ensure the openvswitch kernel module is loaded
openstack_host_specific_kernel_modules:
  - name: "openvswitch"
    pattern: "CONFIG_OPENVSWITCH"
```

Specify provider network definitions in your `/etc/openstack_deploy/openstack_user_config.yml` that define one or more Neutron provider bridges and related configuration:

Note

Bridges specified here will be created automatically. If *network_interface* is defined, the interface will be placed into the bridge automatically as a *DPDK-accelerated* interface.

```
- network:
    container_bridge: "br-provider"
    container_type: "veth"
    type: "vlan"
    range: "101:200,301:400"
    net_name: "physnet1"
    network_interface: "eno49"
    group_binds:
      - neutron_openvswitch_agent
```

A *DPDK-accelerated* **bond** interface can be created by specifying a list of member interfaces using *network_bond_interfaces*. The bond port will be created automatically and added to the respective bridge in OVS:

```
- network:
    container_bridge: "br-provider"
    container_type: "veth"
    type: "vlan"
    range: "101:200,301:400"
    net_name: "physnet1"
```

(continues on next page)

(continued from previous page)

```
    network_bond_interfaces:
      - "0000:04:00.0"
      - "0000:04:00.1"
    group_binds:
      - neutron_openvswitch_agent
```

Additional OVS bond parameters can be specified using the following keys:

- bond_mode (Default: active-backup)
- lacp (Default: off)
- bond_downdelay (Default: 100)
- bond_updelay (Default: 100)

```
- network:
    container_bridge: "br-provider"
    container_type: "veth"
    type: "vlan"
    range: "101:200,301:400"
    net_name: "physnet1"
    network_bond_interfaces:
      - "0000:04:00.0"
      - "0000:04:00.1"
    bond_mode: balance-tcp
    lacp: active
    bond_downdelay: 200
    bond_updelay: 200
    group_binds:
      - neutron_openvswitch_agent
```

For more information on possible values, visit: https://docs.ansible.com/ansible/latest/collections/openvswitch/openvswitch/openvswitch_bond_module.html

Set the following user variables in your `/etc/openstack_deploy/user_variables.yml` to enable the Open vSwitch driver and DPDK support:

```
neutron_plugin_type: ml2.ovs
neutron_ml2_drivers_type: "vlan"
neutron_plugin_base:
  - router
  - metering

# Enable DPDK support
ovs_dpdk_support: True

# Add these overrides or set on per-host basis in openstack_user_config.yml
ovs_dpdk_pci_addresses:
  - "0000:04:00.0"
  - "0000:04:00.1"
ovs_dpdk_lcore_mask: 1010101
```

(continues on next page)

(continued from previous page)

```
ovs_dpdk_pmd_cpu_mask: 2020202
ovs_dpdk_socket_mem: "1024,1024"
```

Note

Overlay networks are not supported on DPDK-enabled nodes at this time.

6.9 Post-installation

Once the playbooks have been run and OVS/DPDK has been configured, it may be necessary to add a physical interface to the provider bridge before networking can be fully established *if network_interface* or *network_bond_interfaces* have not been defined.

On compute nodes, the following command can be used to attach a NIC port `0000:04:00.0` to the provider bridge `br-provider`:

```
ovs-vsctl add-port br-provider 0000:04:00.0 -- set interface 0000:04:00.0␣
↪type=dpdk options:dpdk-devargs=0000:04:00.0
```

Additionally, it may be necessary to make post-installation adjustments to interface queues or other parameters to avoid errors within Open vSwitch:

```
ovs-vsctl set interface 0000:04:00.0 options:n_txq=5
ovs-vsctl set interface 0000:04:00.0 options:n_rxq=5
```

The command(s) can be adjusted according to your configuration.

Warning

Adding multiple ports to a bridge may result in bridging loops unless bonding is configured.

CHAPTER

SEVEN

SCENARIO - USING OPEN VSWITCH WITH SFC

7.1 Overview

Operators can choose to configure SFC mechanism with Open vSwitch instead of ODL through the neutron networking-sfc project. The SFC configuration results in OVS flows being configured with SFC specifics using MPLS as dataplane technology. This document outlines how to set it up in your environment.

7.2 Recommended reading

We recommend that you read the following documents before proceeding:

- General overview of neutron networking-sfc: https://docs.openstack.org/networking-sfc/latest/index.html
- How to configure networking-sfc in neutron: https://docs.openstack.org/networking-sfc/latest/install/configuration.html

7.3 Prerequisites

Configure your networking according the Open vSwitch setup:

- Scenario - Using Open vSwitch https://docs.openstack.org/openstack-ansible-os_neutron/latest/app-openvswitch.html

7.4 OpenStack-Ansible user variables

Create a group var file for your network hosts `/etc/openstack_deploy/group_vars/network_hosts`. It has to include:

```
# Ensure the openvswitch kernel module is loaded
openstack_host_specific_kernel_modules:
  - name: "openvswitch"
    pattern: "CONFIG_OPENVSWITCH"
```

Set the following user variables in your `/etc/openstack_deploy/user_variables.yml`:

```
### neutron specific config
neutron_plugin_type: ml2.ovs

```

(continues on next page)

(continued from previous page)

```
neutron_ml2_drivers_type: "flat,vlan"

neutron_plugin_base:
 - router
 - metering
 - flow_classifier
 - sfc

# Typically this would be defined by the os-neutron-install
# playbook. The provider_networks library would parse the
# provider_networks list in openstack_user_config.yml and
# generate the values of network_types, network_vlan_ranges
# and network_mappings. network_mappings would have a
# different value for each host in the inventory based on
# whether or not the host was metal (typically a compute host)
# or a container (typically a neutron agent container)
#
# When using Open vSwitch, we override it to take into account
# the Open vSwitch bridge we are going to define outside of
# OpenStack-Ansible plays
neutron_provider_networks:
  network_flat_networks: "*"
  network_types: "vlan"
  network_vlan_ranges: "physnet1:102:199"
  network_mappings: "physnet1:br-provider"
```

Note: The only difference to the Standard Open vSwitch configuration is the setting of the `neutron_plugin_base`.

CHAPTER

EIGHT

SCENARIO - USING THE NUAGE NEUTRON PLUGIN

8.1 Introduction

Follow these steps to deploy Nuage Networks VCS with OpenStack-Ansible (OSA):

- Install prerequisites.
- Configure neutron to use the Nuage Networks neutron plugin.
- Configure the Nuage Networks neutron plugin.
- Download Nuage Networks VCS components and playbooks.
- Execute the playbooks.

8.2 Prerequisites

1. The deployment environment is configured according to OSA best practices such as cloning OSA software and bootstrapping Ansible. See OpenStack-Ansible Install Guide.
2. VCS stand-alone components, VSD and VSC, are configured and deployed. See the Nuage Networks VSD and VSC Install Guides.
3. Nuage VRS playbooks were cloned to the deployment host from https://github.com/nuagenetworks/nuage-openstack-ansible. This guide assumes a deployment host path of `/opt/nuage-openstack-ansible`.

8.3 Configure Nuage neutron plugin

Configuring the neutron plugin requires creating or editing of parameters in the following two files:

- `/etc/openstack_deploy/user_nuage_vars.yml`
- `/etc/openstack_deploy/user_variables.yml`

On the deployment host, copy the Nuage user variables file from `/opt/nuage-openstack-ansible/etc/user_nuage_vars.yml` to the `/etc/openstack_deploy/` folder.

```
# cp /opt/nuage-openstack-ansible/etc/user_nuage_vars.yml \
  /etc/openstack_deploy/
```

Next, modify the following parameters in that file as per your Nuage VCS environment:

1. Replace *VSD Enterprise Name* with the name of VSD Enterprise:

```
nuage_net_partition_name: "<VSD Enterprise Name>"
```

2. Replace *VSD IP* and *VSD GUI Port* as per your VSD configuration:

```
nuage_vsd_ip: "<VSD IP>:<VSD GUI Port>"
```

3. Replace *VSD Username, VSD Password*, and *VSD Organization Name* with your login credentials for the VSD GUI:

```
nuage_vsd_username: "<VSD Username>"

nuage_vsd_password: "<VSD Password>"

nuage_vsd_organization: "<VSD Organization Name>"
```

4. Replace *Nuage VSP Version* with the Nuage VSP release for Integration. For example, for Nuage VSP release 3.2 this value would be *v3_2*.

```
nuage_base_uri_version: "<Nuage VSP Version>"
```

5. Replace *Nuage VSD CMS Id* with the `CMS-Id` generated by VSD to manage your OpenStack cluster:

```
nuage_cms_id: "<Nuage VSD CMS Id>"
```

6. Replace *Active VSC-IP* with the IP address of your active VSC node and *Standby VSC-IP* with the IP address of your standby VSC node:

```
active_controller: "<Active VSC-IP>"

standby_controller: "<Standby VSC-IP>"
```

7. Replace *Local Package Repository* with the link of the local repository hosting the Nuage VRS packages. For example: `http://192.0.2.10/debs/3.2/vrs/`:

```
nuage_vrs_debs_repo: "deb <Local Package Repository>"
```

8. On the Deployment host, add the following lines to the `/etc/openstack_deploy/user_variables.yml` file, replacing the *Local PyPi Mirror URL* with the link to the PyPi server hosting the Nuage OpenStack Python packages in .whl format:

```
neutron_plugin_type: "nuage"
nova_network_type: "nuage"
pip_links:
  - { name: "openstack_release", link: "{{ openstack_repo_url \
  }}/os-releases/{{ openstack_release }}/" }
  - { name: "nuage_repo", link: "<Local PyPi Mirror URL>" }
```

8.4 Installation

1. After you set up the multi-node OpenStack cluster, start the OpenStack deployment as listed in the OpenStack-Ansible Install guide by running all playbooks in sequence on the deployment host.

2. After OpenStack deployment is complete, deploy Nuage VRS on all compute target hosts in the OpenStack cluster by running the Nuage VRS playbooks in `/opt/nuage-openstack-ansible/nuage_playbook` on your deployment host:

```
# cd /opt/nuage-openstack-ansible/nuage_playbooks
# openstack-ansible nuage_all.yml
```

Note

To obtain Nuage Networks VSP software packages, user documentation, and licenses, contact `info@nuagenetworks.net`.

CHAPTER

NINE

SCENARIO - VMWARE NSX PLUGIN

9.1 Introduction

This document covers the steps to integrate the VMware NSX plugin with OpenStack Ansible.

Warning

Currently, only NSX-T Policy API is supported.

Please follow these steps:

- Configure Neutron to use the NSX plugin

9.2 Prerequisites

1. The deployment environment is configured according to OSA best practices such as cloning OSA software and bootstrapping Ansible. See OpenStack-Ansible Install Guide.
2. NSX-T has been deployed per its installation guide and compute nodes have been properly configured as transport nodes. See *NSX-T Data Center Installation Guide <https://docs.vmware.com/en/VMware-NSX-T-Data-Center/3.0/installation/GUID-3E0C4CEC-D593-4395-84C4-150CD6285963.htm> _*.

9.3 Configure Neutron to use the NSX plugin

Copy the neutron environment overrides to `/etc/openstack_deploy/env.d/neutron.yml` and disable agent creation, since it is not needed.

```
neutron_agents_container:
  belongs_to:
    - network_containers
  contains: { }
```

Copy the nova environment overrides to `/etc/openstack_deploy/env.d/nova.yml` and disable neutron agent creation, since it is not needed.

```
container_skel:
  nova_api_container:
    belongs_to:
```

(continues on next page)

(continued from previous page)

```
      - compute-infra_containers
      - os-infra_containers
    contains:
      - nova_api_metadata
      - nova_api_os_compute
      - nova_conductor
      - nova_scheduler
      - nova_console
  nova_compute_container:
    belongs_to:
      - compute_containers
      - kvm-compute_containers
      - qemu-compute_containers
    contains:
      - nova_compute
    properties:
      is_metal: true
```

Set the following required variables in your `/etc/openstack_deploy/user_variables.yml`

```
neutron_plugin_type: vmware.nsx
nova_network_type: nsx
nsx_api_password: <password>
nsx_api_managers:
  - nsx-manager-01
  - nsx-manager-02
  - nsx-manager-03
```

Optionally specify additional parameters using overrides

```
neutron_nsx_conf_ini_overrides:
  nsx_p:
    default_tier0_router: my-tier0-router
    default_overlay_tz: my-overlay-tz
    default_vlan_tz: my-vlan-tz
    metadata_proxy: my-metadata-proxy-profile
    dhcp_profile: my-dhcp-profile
```

Warning

If NSX has defined more than one tier 0, overlay/vlan tz, metadata proxy, or dhcp profile, then you must explicitly define those using conf overrides. Neutron will fail to start if these are not defined in those conditions.

9.4 Installation

After the environment has been configured as detailed above, start the OpenStack deployment as listed in the OpenStack-Ansible Install Guide.

CHAPTER

TEN

SCENARIO - NETWORKING GENERIC SWITCH

10.1 Overview

Operators can choose to utilize the Networking Generic Switch (NGS) mechanism driver to manage physical switches when Ironic is integrated with Neutron. The Networking Generic Switch mechanism driver can be deployed alongside other drivers, such as Open vSwitch or LinuxBridge. This document outlines how to set it up in your environment.

10.2 Recommended reading

It is recommended to familiarize yourself with project-specific documentation to better understand deployment and configuration options:

- Networking Generic Switch

10.3 Prerequisites

- Ironic Bare-Metal Provisioning Service
- Supported Network Hardware
- Network connectivity from the node(s) running the *neutron-server* service to the management interface of the physical switch(es) connected to Ironic bare-metal nodes. This is outside the scope of OpenStack-Ansible.

10.4 OpenStack-Ansible user variables

Add `ml2.genericswitch` to the `neutron_plugin_types` list in `/etc/openstack_deploy/user_variables.yml`:

```
neutron_plugin_types:
- ml2.genericswitch
```

To interface with a supported network switch, configure ini overrides for each connected switch in your environment:

```
neutron_ml2_conf_genericswitch_ini_overrides:
  genericswitch:arista01:
    device_type: netmiko_arista_eos
    ngs_mac_address: "00:1c:73:29:ea:ca"
```

(continues on next page)

(continued from previous page)

```
    ip: "192.168.90.2"
    username: "openstack"
    password: "0p3nst@ck"
    ngs_port_default_vlan: 3
  genericswitch:arista02:
    device_type: netmiko_arista_eos
    ngs_mac_address: "00:1c:73:29:ea:cb"
    ip: "192.168.90.3"
    username: "openstack"
    password: "0p3nst@ck"
    ngs_port_default_vlan: 3
```

Lastly, configure an override to Ironic to enable the `neutron` interface:

```
ironic_enabled_network_interfaces_list: neutron
ironic_default_network_interface: neutron
```

10.5 Notes

Ironic bare-metal ports that are associated with bare-metal nodes can be configured with the respective connection details using the `openstack baremetal port set` command:

```
openstack baremetal port set 3a948c3b-6c41-4f68-8389-c4f5ca667c63 \
--local-link-connection switch_info=arista01 \
--local-link-connection switch_id="00:1c:73:29:ea:ca" \
--local-link-connection port_id="et11"
```

When a server is deployed using a bare-metal node, Neutron will connect to the respective switch(es) and configure the switchport interface(s) according.

This role installs the following Systemd services:

- neutron-server
- neutron-agents

To clone or view the source code for this repository, visit the role repository for os_neutron.

CHAPTER

ELEVEN

DEFAULT VARIABLES

```
###
### Verbosity Options
###

debug: false

###
### Service setup options
###

# Set the host which will execute the shade modules
# for the service setup. The host must already have
# clouds.yaml properly configured.
neutron_service_setup_host: "{{ openstack_service_setup_host | default(
↪'localhost') }}"
neutron_service_setup_host_python_interpreter: >-
  {{
    openstack_service_setup_host_python_interpreter | default(
      (neutron_service_setup_host == 'localhost') | ternary(ansible_playbook_
↪python, ansible_facts['python']['executable']))
  }}

###
### Packages Options
###

# Set the package install state for distribution
# Options are 'present' and 'latest'
neutron_package_state: "{{ package_state | default('latest') }}"

# Set installation method.
neutron_install_method: "{{ service_install_method | default('source') }}"
neutron_venv_python_executable: "{{ openstack_venv_python_executable |␣
↪default('python3') }}"

###
### Python code details
###
```

(continues on next page)

(continued from previous page)

```
# Set the package install state for pip_package
# Options are 'present' and 'latest'
neutron_pip_package_state: "latest"

# Source git repo/branch settings
neutron_git_repo: https://opendev.org/openstack/neutron
neutron_git_install_branch: master
neutron_fwaas_git_repo: https://opendev.org/openstack/neutron-fwaas
neutron_fwaas_git_install_branch: master
neutron_vpnaas_git_repo: https://opendev.org/openstack/neutron-vpnaas
neutron_vpnaas_git_install_branch: master
neutron_dynamic_routing_git_repo: https://opendev.org/openstack/neutron-
↪dynamic-routing
neutron_dynamic_routing_git_install_branch: master
networking_odl_git_repo: https://opendev.org/openstack/networking-odl
networking_odl_git_install_branch: master
networking_ovn_bgp_git_repo: https://opendev.org/openstack/ovn-bgp-agent
networking_ovn_bgp_git_install_branch: master
networking_sfc_git_repo: https://opendev.org/openstack/networking-sfc
networking_sfc_git_install_branch: master
networking_bgpvpn_git_repo: https://opendev.org/openstack/networking-bgpvpn
networking_bgpvpn_git_install_branch: master
ceilometer_git_repo: https://opendev.org/openstack/ceilometer
ceilometer_git_install_branch: master
networking_baremetal_git_repo: https://opendev.org/openstack/networking-
↪baremetal
networking_baremetal_git_install_branch: master
networking_generic_switch_git_repo: https://opendev.org/openstack/networking-
↪generic-switch
networking_generic_switch_git_install_branch: master
networking_nsx_git_repo: https://opendev.org/x/vmware-nsx
networking_nsx_git_install_branch: master
networking_nsxlib_git_repo: https://opendev.org/x/vmware-nsxlib
networking_nsxlib_git_install_branch: master

neutron_upper_constraints_url: >-
  {{ requirements_git_url | default('https://releases.openstack.org/
↪constraints/upper/' ~ requirements_git_install_branch | default('master')) }
↪}
neutron_git_constraints:
  - "--constraint {{ neutron_upper_constraints_url }}"

neutron_pip_install_args: "{{ pip_install_options | default('') }}"

# Name of the virtual env to deploy into
neutron_venv_tag: "{{ venv_tag | default('untagged') }}"

###
```

(continues on next page)

(continued from previous page)

```
### Generic Neutron Config
###

# Fatal Deprecations
neutron_fatal_deprecations: false

# If ``neutron_api_workers`` is unset the system will use half the number of␣
↪available VCPUs to
# compute the number of api workers to use with a default capping value of 16.
# neutron_api_workers: 16

## Cap the maximun number of threads / workers when a user value is␣
↪unspecified.
neutron_api_threads_max: 16
neutron_api_threads: "{{ [[ansible_facts['processor_vcpus'] | default(2) // 2,
↪ 1] | max, neutron_api_threads_max] | min }}"

neutron_agent_down_time: 120
neutron_agent_polling_interval: 5
neutron_report_interval: "{{ neutron_agent_down_time | int / 2 | int }}"

neutron_dns_domain: "{{ dhcp_domain | default('openstacklocal.') }}"

# If ``neutron_num_sync_threads`` is unset, the system will use the value of
# neutron_api_threads in templates/dhcp_agent.ini.j2 for num_sync_threads.
# neutron_num_sync_threads: 4

###
### DNSMasq configuration
###
# Dnsmasq doesn't work with config_template override, a deployer
# should instead configure its own neutron_dhcp_config key/values like this:
# neutron_dhcp_config:
#   dhcp-option-force: "26,1500"
neutron_dhcp_config: {}

# Dnsmasq has furthermore some options in its configuration that are not
# key/value pairs but just options. A deployer can configure those with this
# list:
neutron_dhcp_config_list: []

# Disable dnsmasq to resolve DNS via local resolv.conf.
# When dnsmasq_dns_servers are not set,
# and neutron_dnsmasq_noresolv is set to True, dnsmasq will reply with
# empty respose on DNS requests.
neutron_dnsmasq_noresolv: false

###
### Tunable Overrides (Sorted alphabetically)
```

(continues on next page)

(continued from previous page)

```
###

# These variables facilitate adding config file entries
# for anything supported by the service. See the section
# 'Overriding OpenStack configuration defaults' in the
# 'Advanced configuration' appendix of the Deploy Guide.
neutron_api_paste_ini_overrides: {}
neutron_bgp_dragent_ini_overrides: {}
neutron_bgp_dragent_init_overrides: {}
neutron_dhcp_agent_ini_overrides: {}
neutron_dhcp_agent_init_overrides: {}
neutron_ironic_neutron_agent_ini_overrides: {}
neutron_ironic_neutron_agent_init_overrides: {}
neutron_l3_agent_ini_overrides: {}
neutron_l3_agent_init_overrides: {}
neutron_metadata_agent_ini_overrides: {}
neutron_metadata_agent_init_overrides: {}
neutron_metering_agent_ini_overrides: {}
neutron_metering_agent_init_overrides: {}
neutron_ml2_conf_ini_overrides: {}
neutron_ml2_conf_genericswitch_ini_overrides: {}
neutron_neutron_conf_overrides: {}
neutron_nuage_conf_ini_overrides: {}
neutron_openvswitch_agent_ini_overrides: {}
neutron_openvswitch_agent_init_overrides: {}
neutron_ovn_bgp_agent_ini_overrides: {}
neutron_ovn_bgp_agent_init_overrides: {}
neutron_nsx_conf_ini_overrides: {}
# Provide a list of access controls to update the default policy.json with.
# These changes will be merged
# with the access controls in the default policy.json. E.g.
# neutron_policy_overrides:
#   "create_subnet": "rule:admin_or_network_owner"
#   "get_subnet": "rule:admin_or_owner or rule:shared"
neutron_policy_overrides: {}
_neutron_rootwrap_conf_overrides:
  DEFAULT:
    filters_path: "{{ neutron_conf_dir }}/rootwrap.d,/usr/share/neutron/
↪rootwrap"
    exec_dirs: "{{ neutron_bin }},/sbin,/usr/sbin,/bin,/usr/bin,/usr/local/
↪bin,/usr/local/sbin,/etc/neutron/kill_scripts"
neutron_rootwrap_conf_overrides: {}

neutron_api_uwsgi_ini_overrides: {}
neutron_periodic_workers_init_overrides: {}
neutron_server_init_overrides: {}
neutron_rpc_server_init_overrides: {}
neutron_sriov_nic_agent_ini_overrides: {}
neutron_sriov_nic_agent_init_overrides: {}
```

(continues on next page)

(continued from previous page)

```
neutron_ovn_maintenance_init_overrides: {}
neutron_ovn_metadata_agent_ini_overrides: {}
neutron_ovn_metadata_agent_init_overrides: {}

###
### UWSGI
###
neutron_wsgi_processes_max: 16
neutron_wsgi_processes: "{{ [[ansible_facts['processor_vcpus'] | default(1),␣
↪1] | max * 2, neutron_wsgi_processes_max] | min }}"
neutron_wsgi_threads: 1
neutron_uwsgi_tls:
  crt: "{{ neutron_ssl_cert }}"
  key: "{{ neutron_ssl_key }}"

###
### Quotas
###

neutron_default_quota: -1
neutron_quota_floatingip: 50
neutron_quota_health_monitor: -1
neutron_quota_member: -1
neutron_quota_network: 100
neutron_quota_network_gateway: 5
neutron_quota_packet_filter: 100
neutron_quota_pool: 10
neutron_quota_port: 500
neutron_quota_router: 10
neutron_quota_security_group: 10
neutron_quota_security_group_rule: 100
neutron_quota_subnet: 100
neutron_quota_vip: 10
neutron_quota_firewall: 10
neutron_quota_firewall_policy: 10
neutron_quota_firewall_rule: 100

###
### DB (Galera) integration
###

neutron_db_setup_host: "{{ openstack_db_setup_host | default('localhost') }}"
neutron_db_setup_python_interpreter: >-
  {{
    openstack_db_setup_python_interpreter | default(
      (neutron_db_setup_host == 'localhost') | ternary(ansible_playbook_
↪python, ansible_facts['python']['executable']))
  }}
neutron_galera_address: "{{ galera_address | default('127.0.0.1') }}"
```

(continues on next page)

(continued from previous page)

```
neutron_galera_user: neutron
neutron_galera_database: neutron
neutron_db_max_overflow: "{{ openstack_db_max_overflow | default('50') }}"
neutron_db_max_pool_size: "{{ openstack_db_max_pool_size | default('5') }}"
neutron_db_pool_timeout: "{{ openstack_db_pool_timeout | default('30') }}"
neutron_db_connection_recycle_time: "{{ openstack_db_connection_recycle_time␣
↪| default('600') }}"
neutron_galera_use_ssl: "{{ galera_use_ssl | default(False) }}"
neutron_galera_ssl_ca_cert: "{{ galera_ssl_ca_cert | default('') }}"
neutron_galera_port: "{{ galera_port | default('3306') }}"

###
### Oslo Messaging
###

# RabbitMQ

# RPC

neutron_oslomsg_rpc_host_group: "{{ oslomsg_rpc_host_group | default(
↪'rabbitmq_all') }}"
neutron_oslomsg_rpc_setup_host: "{{ (neutron_oslomsg_rpc_host_group in␣
↪groups) | ternary(groups[neutron_oslomsg_rpc_host_group][0], 'localhost') }}
↪"
neutron_oslomsg_rpc_transport: "{{ oslomsg_rpc_transport | default('rabbit') }
↪}"
neutron_oslomsg_rpc_servers: "{{ oslomsg_rpc_servers | default('127.0.0.1') }}
↪"
neutron_oslomsg_rpc_port: "{{ oslomsg_rpc_port | default('5672') }}"
neutron_oslomsg_rpc_use_ssl: "{{ oslomsg_rpc_use_ssl | default(False) }}"
neutron_oslomsg_rpc_userid: neutron
neutron_oslomsg_rpc_policies: []
neutron_oslomsg_rpc_vhost:
  - name: /neutron
    state: "{{ neutron_oslomsg_rabbit_quorum_queues | ternary('absent',
↪'present') }}"
  - name: neutron
    state: "{{ neutron_oslomsg_rabbit_quorum_queues | ternary('present',
↪'absent') }}"
neutron_oslomsg_rpc_ssl_version: "{{ oslomsg_rpc_ssl_version | default('TLSv1_
↪2') }}"
neutron_oslomsg_rpc_ssl_ca_file: "{{ oslomsg_rpc_ssl_ca_file | default('') }}"

# Notify
neutron_oslomsg_notify_configure: "{{ oslomsg_notify_configure |␣
↪default(neutron_ceilometer_enabled) }}"
neutron_oslomsg_notify_host_group: "{{ oslomsg_notify_host_group | default(
↪'rabbitmq_all') }}"
neutron_oslomsg_notify_setup_host: >-
```

(continues on next page)

(continued from previous page)

```
  {{ (neutron_oslomsg_notify_host_group in groups) | ternary(groups[neutron_
↪oslomsg_notify_host_group][0], 'localhost') }}
neutron_oslomsg_notify_transport: "{{ oslomsg_notify_transport | default(
↪'rabbit') }}"
neutron_oslomsg_notify_servers: "{{ oslomsg_notify_servers | default('127.0.0.
↪1') }}"
neutron_oslomsg_notify_port: "{{ oslomsg_notify_port | default('5672') }}"
neutron_oslomsg_notify_use_ssl: "{{ oslomsg_notify_use_ssl | default(False) }}
↪"
neutron_oslomsg_notify_userid: "{{ neutron_oslomsg_rpc_userid }}"
neutron_oslomsg_notify_password: "{{ neutron_oslomsg_rpc_password }}"
neutron_oslomsg_notify_vhost: "{{ neutron_oslomsg_rpc_vhost }}"
neutron_oslomsg_notify_ssl_version: "{{ oslomsg_notify_ssl_version | default(
↪'TLSv1_2') }}"
neutron_oslomsg_notify_ssl_ca_file: "{{ oslomsg_notify_ssl_ca_file | default('
↪') }}"
neutron_oslomsg_notify_policies: []

###
### (RabbitMQ) integration
###
neutron_oslomsg_rabbit_quorum_queues: "{{ oslomsg_rabbit_quorum_queues |␣
↪default(True) }}"
neutron_oslomsg_rabbit_stream_fanout: "{{ oslomsg_rabbit_stream_fanout |␣
↪default(neutron_oslomsg_rabbit_quorum_queues) }}"
neutron_oslomsg_rabbit_transient_quorum_queues: "{{ oslomsg_rabbit_transient_
↪quorum_queues | default(neutron_oslomsg_rabbit_stream_fanout) }}"
neutron_oslomsg_rabbit_qos_prefetch_count: "{{ oslomsg_rabbit_qos_prefetch_
↪count | default(neutron_oslomsg_rabbit_stream_fanout | ternary(10, 0)) }}"
neutron_oslomsg_rabbit_queue_manager: "{{ oslomsg_rabbit_queue_manager |␣
↪default(neutron_oslomsg_rabbit_quorum_queues) }}"
neutron_oslomsg_rabbit_quorum_delivery_limit: "{{ oslomsg_rabbit_quorum_
↪delivery_limit | default(0) }}"
neutron_oslomsg_rabbit_quorum_max_memory_bytes: "{{ oslomsg_rabbit_quorum_max_
↪memory_bytes | default(0) }}"
neutron_rpc_thread_pool_size: 64
neutron_rpc_conn_pool_size: 30
neutron_rpc_response_timeout: 60
neutron_rpc_workers_max: 16
neutron_rpc_workers: >-
  {{ [[(ansible_facts['processor_vcpus'] // ansible_facts['processor_threads_
↪per_core']) | default(1), 1] | max * 2, neutron_rpc_workers_max] | min }}

###
### Identity (Keystone) integration
###

neutron_service_project_name: service
neutron_service_project_domain_id: default
```

(continues on next page)

(continued from previous page)

```
neutron_service_user_domain_id: default
neutron_service_role_names:
  - admin
  - service
neutron_service_token_roles:
  - service
neutron_service_token_roles_required: "{{ openstack_service_token_roles_
↪required | default(True) }}"
neutron_service_user_name: neutron
neutron_service_name: neutron
neutron_service_type: network
neutron_service_description: "OpenStack Networking"
neutron_api_bind_address: "{{ openstack_service_bind_address | default('0.0.0.
↪0') }}"
neutron_service_port: 9696
neutron_service_proto: http
neutron_service_publicuri_proto: "{{ openstack_service_publicuri_proto |␣
↪default(neutron_service_proto) }}"
neutron_service_adminuri_proto: "{{ openstack_service_adminuri_proto |␣
↪default(neutron_service_proto) }}"
neutron_service_internaluri_proto: "{{ openstack_service_internaluri_proto |␣
↪default(neutron_service_proto) }}"
neutron_service_publicuri: "{{ neutron_service_publicuri_proto }}://{{␣
↪external_lb_vip_address }}:{{ neutron_service_port }}"
neutron_service_publicurl: "{{ neutron_service_publicuri }}"
neutron_service_adminuri: "{{ neutron_service_adminuri_proto }}://{{ internal_
↪lb_vip_address }}:{{ neutron_service_port }}"
neutron_service_adminurl: "{{ neutron_service_adminuri }}"
neutron_service_internaluri: "{{ neutron_service_internaluri_proto }}://{{␣
↪internal_lb_vip_address }}:{{ neutron_service_port }}"
neutron_service_internalurl: "{{ neutron_service_internaluri }}"
neutron_service_region: "{{ service_region | default('RegionOne') }}"
neutron_keystone_auth_plugin: "{{ neutron_keystone_auth_type }}"
neutron_keystone_auth_type: password
neutron_service_in_ldap: "{{ service_ldap_backend_enabled | default(False) }}"

###
### Availability zones
###
# Availability zone defines current AZ of the component. For OVN you can␣
↪define
# multiple AZs separated with a colon, ie "az1:az2"
neutron_availability_zone: nova

# Default availability zones do define a list of zones to where routers/agents
# will be scheduled by default. This is a list, since a deployment might␣
↪stretch
# networks across AZs.
# Default: Neutron will attempt scheduling across all defined AZs for Neutron␣
```

(continues on next page)

(continued from previous page)

```
↪hosts.
neutron_default_availability_zones: >-
  {{
    groups['neutron_all'] | map(
      'extract', hostvars, 'neutron_availability_zone') | map(
        'default', neutron_availability_zone) | map('split', ':') | flatten |␣
↪unique
  }}

###
### Telemetry integration
###

neutron_ceilometer_enabled: "{{ (groups['ceilometer_all'] is defined) and␣
↪(groups['ceilometer_all'] | length > 0) }}"

###
### Designate integration
###

neutron_designate_enabled: "{{ (groups['designate_all'] is defined) and␣
↪(groups['designate_all'] | length > 0) }}"
neutron_allow_reverse_dns_lookup: true
neutron_ipv4_ptr_zone_prefix_size: 24
neutron_ipv6_ptr_zone_prefix_size: 116

###
### Plugins Loading
###

# Other plugins can be added to the system by simply extending the list␣
↪`neutron_plugin_base`.
# neutron_plugin_base:
#   - router
#   - firewall_v2
#   - neutron_dynamic_routing.services.bgp.bgp_plugin.BgpPlugin
#   - vpnaas
#   - metering
#   - qos
#   - dns/dns_domain_ports/subnet_dns_publish_fixed_ip either one or the␣
↪other, not both
#   - port_forwarding
neutron_plugin_base:
  - ovn-router

###
### Memcache override
###
neutron_memcached_servers: "{{ memcached_servers }}"
```

(continues on next page)

(continued from previous page)

```
###
### ML2 Plugin Configuration
###

# The neutron core plugin (ML2) is defined with neutron_plugin_type,
# you can not load multiple ML2 plugins as core.
neutron_plugin_type: "ml2.ovn"

# Additional ML2 plugins can be loaded with neutron_plugin_types (as list)
neutron_plugin_types: []

# ml2 network type drivers to load
neutron_ml2_drivers_type: "geneve,vlan,flat"

# Enable or disable L2 Population.
# When using ovs dvr it must be enabled
neutron_l2_population: "{{ neutron_plugin_type == 'ml2.ovs.dvr' }}"

neutron_vxlan_enabled: true

## The neutron multicast group address. This should be set as a host variable␣
↪if used.
neutron_vxlan_group: "239.1.1.1"

# The neutron multicast time-to-live. Number of L3 hops before routers will␣
↪drop the traffic
neutron_vxlan_ttl: 32

neutron_sriov_excluded_devices: ""

# neutron_local_ip is used for the VXLAN local tunnel endpoint
neutron_local_ip: "{{ tunnel_address | default('127.0.0.1') }}"

## Set this variable to configure the provider networks that will be available
## When setting up networking in things like the ml2_conf.ini file. Normally
## this will be defined as a host variable used within neutron as network␣
↪configuration
## are likely to differ in between hosts.
# neutron_provider_networks:
#   network_flat_networks: "flat"
#   network_mappings: "flat:eth12,vlan:eth11"
#   network_types: "vxlan,flat,vlan"
#   network_vlan_ranges: "vlan:1:1,vlan:1024:1025"
#   network_vxlan_ranges: "1:1000"
#   network_geneve_ranges: "1:1000"
#   network_sriov_mappings: "vlan:p4p1"

###
```

(continues on next page)

(continued from previous page)

```
### L3 Agent Plugin Configuration
###

# L3HA configuration options
neutron_ha_vrrp_auth_type: PASS
neutron_l3_ha_net_cidr: 169.254.192.0/18

neutron_l3_cleanup_on_shutdown: false

## List of extensions enabled for L3
## The list can be extended by operator if needed, ie in user_variables.yml:
# neutron_l3_agent_extensions: "{{ _neutron_l3_agent_extensions + ['neutron_
↪fip_qos'] }}"
neutron_l3_agent_extensions: "{{ _neutron_l3_agent_extensions }}"

# Specify the maximum number of L3 agents per tenant network. Defaults to the␣
↪total number of agents deployed
# neutron_l3_agents_max: 2

###
### DHCP Agent Plugin Configuration
###

# Comma-separated list of DNS servers which will be used by dnsmasq as␣
↪forwarders.
# This variable will be used for the same purpose for OVN, when dnsmasq is␣
↪not used.
neutron_dnsmasq_dns_servers: ""

# Limit number of leases to prevent a denial-of-service.
neutron_dnsmasq_lease_max: 16777216

# Specify if dnsmasq should send a route to metadata server through DHCP 121␣
↪message to VM
neutron_dnsmasq_force_metadata: false

# Specify the maximum number of DHCP agents per tenant network. Defaults to␣
↪the total number of agents deployed
# neutron_dhcp_agents_max: 2

###
### Metadata Agent Plugin Configuration
###

# If ``neutron_metadata_workers`` is unset the system will use half the number␣
↪of available VCPUs to
# compute the number of api workers to use with a default capping value of 16.
# neutron_metadata_workers: 16
neutron_metadata_backlog: 4096
```

(continues on next page)

(continued from previous page)

```
# The port used by neutron to access the nova metadata service.
neutron_nova_metadata_port: "{{ nova_metadata_port | default(8775) }}"

# The protocol used by neutron to access the nova metadata service.
neutron_nova_metadata_protocol: "{{ nova_metadata_protocol | default('http') }
↪}"

# If the nova_metadata_protocol is using a self-signed cert, then
# this flag should be set to a boolean True.
neutron_nova_metadata_insecure: "{{ nova_metadata_insecure | default(False) }}
↪"

###
### FWaaS Configuration
###

neutron_driver_fwaasv2: iptables_v2
neutron_fwaasv2_service_provider: FIREWALL_V2:fwaas_db:neutron_fwaas.services.
↪firewall.service_drivers.agents.agents.FirewallAgentDriver:default

###
### VPNaaS Configuration
###

# See VPNaaS documentation for driver/service provider selection
# in case you want to override it.
neutron_driver_vpnaas: "{{ _neutron_driver_vpnaas }}"
neutron_vpnaas_service_provider: "{{ _neutron_vpnaas_service_provider }}"

# Set this variable to use custom config file for strongswan/openswan
# neutron_vpnaas_custom_config:
#   - src: "/etc/openstack_deploy/strongswan/strongswan.conf.template"
#     dest: "{{ neutron_conf_dir }}/strongswan.conf.template"
#     condition: "{{ ansible_facts['os_family'] | lower == 'debian' }}"

neutron_vpnaas_custom_config: []
neutron_ovn_vpn_agent_overrides: {}
neutron_ovn_vpn_agent_init_overrides: {}

# OVN Defaults
neutron_ovn_ssl: true
ovn_proto: "{{ (neutron_ovn_ssl) | ternary('ssl', 'tcp') }}"
neutron_ovn_log_parameters:
  - --syslog-method=unix:/dev/log
  - --verbose=console:off
  - "--verbose=syslog:{{ (debug | bool) | ternary('dbg', 'info') }}"
  - --verbose=file:off
```

(continues on next page)

(continued from previous page)

```
neutron_ovn_primary_cluster_node: "{{ groups[neutron_services['neutron-ovn-
↪northd']['group']] | first }}"
neutron_ovn_northd_service_name: ovn-northd
neutron_ovn_controller_service_name: ovn-controller
neutron_ovn_l3_scheduler: leastloaded
neutron_ovn_distributed_fip: false
neutron_ovn_nb_connection: >-
  {{ ovn_proto }}:{{ groups['neutron_ovn_northd'] | map('extract', hostvars, [
↪'ansible_host']) | join(':6641,' + ovn_proto + ':') }}:6641
neutron_ovn_sb_connection: >-
  {{ ovn_proto }}:{{ groups['neutron_ovn_northd'] | map('extract', hostvars, [
↪'ansible_host']) | join(':6642,' + ovn_proto + ':') }}:6642
neutron_ovsdb_manager_host: 127.0.0.1
neutron_ovsdb_manager_port: 6640
neutron_ovsdb_manager_proto: tcp
neutron_ovsdb_manager: "p{{ [neutron_ovsdb_manager_proto, neutron_ovsdb_
↪manager_port, neutron_ovsdb_manager_host] | select | join(':') }}"
neutron_ovsdb_manager_connection: "{{ [neutron_ovsdb_manager_proto, neutron_
↪ovsdb_manager_host, neutron_ovsdb_manager_port] | select | join(':') }}"
neutron_ovn_sb_inactivity_probe: 60000
neutron_ovn_nb_inactivity_probe: 60000

# OVN BGP Agent
neutron_ovn_bgp_enable: false
neutron_ovn_bgp_agent_group: "{{ neutron_ovn_distributed_fip | ternary(
↪'neutron_ovn_controller', 'neutron_ovn_gateway') }}"
neutron_ovn_bgp_agent_driver: nb_ovn_bgp_driver
neutron_ovn_bgp_exposing_method: underlay
neutron_ovn_bgp_expose_tenant_networks: false
neutron_ovn_bgp_expose_ipv6_gua_tenant_networks: false
# Provide config needed for BGP peering
# neutron_ovn_bgp_config:
#   AS: 64999
#   nic: bgp-nic
#   vrf: bgp-vrf
#   vrf_table_id: 10
neutron_ovn_bgp_config: {}
neutron_frr_bgp_config: []
neutron_frr_staticd_routes: []

# This section is used when neutron_ovn_bgp_exposing_method
# is set to "ovn".
# This requires a standalone "local" cluster per node where
# ovn-bgp-agent runs.
neutron_ovn_bgp_local_nbdb: tcp:127.0.0.1:6641
neutron_ovn_bgp_local_nics: []
neutron_ovn_bgp_local_peers: []
neutron_ovn_bgp_provider_networks_prefixes: []
```

(continues on next page)

(continued from previous page)

```
# Storage location for SSL certificate authority
neutron_ovn_pki_dir: "{{ openstack_pki_dir }}"
# Delegated host for operating the certificate authority
neutron_ovn_pki_setup_host: "{{ openstack_pki_setup_host | default('localhost
↪') }}"
# The local address used for the neutron_ovn node
neutron_ovn_node_address: "{{ management_address | default('127.0.0.1') }}"
# neutron OVN server certificate
neutron_ovn_pki_keys_path: "{{ neutron_ovn_pki_dir ~ '/certs/private/' }}"
neutron_ovn_pki_certs_path: "{{ neutron_ovn_pki_dir ~ '/certs/certs/' }}"
neutron_ovn_pki_intermediate_cert_name: "{{ openstack_pki_service_
↪intermediate_cert_name }}"
neutron_ovn_pki_intermediate_chain_path: >-
  {{ neutron_ovn_pki_dir ~ '/roots/' ~ neutron_ovn_pki_intermediate_cert_name␣
↪~ '/certs/' ~ neutron_ovn_pki_intermediate_cert_name ~ '-chain.crt' }}
neutron_ovn_pki_regen_cert: ""
neutron_ovn_pki_certificates:
  - name: "neutron_ovn_{{ ansible_facts['hostname'] }}"
    cn: "{{ ansible_facts['hostname'] }}"
    san:
      dns:
        - "{{ ansible_facts['hostname'] }}"
      ip:
        - "{{ neutron_ovn_node_address }}"
    # standalone backend only
    provider: ownca
    signed_by: "{{ neutron_ovn_pki_intermediate_cert_name }}"

# OVN destination files for SSL certificates
neutron_ovn_ssl_cert: "neutron_ovn.pem"
neutron_ovn_ssl_key: "neutron_ovn.key"
neutron_ovn_ssl_ca_cert: "neutron_ovn-ca.pem"
# User supplied certificates/keys, provide the path to the files on the␣
↪deployment host
neutron_ovn_user_ssl_cert: ""
neutron_ovn_user_ssl_key: ""
neutron_ovn_user_ssl_ca_cert: ""

neutron_ovn_conf_dir: "/etc/openvswitch"
# Installation details for SSL certificates
neutron_ovn_pki_install_certificates:
  - name: "neutron_ovn_{{ ansible_facts['hostname'] }}"
    type: "certificate_chain"
    dest: "{{ [neutron_ovn_conf_dir, neutron_ovn_ssl_cert] | join('/') }}"
    owner: "{{ neutron_ovn_system_user_name }}"
    group: "{{ neutron_ovn_system_user_name }}"
    condition: "{{ (neutron_ovn_ssl and neutron_needs_openvswitch and neutron_
↪plugin_type == 'ml2.ovn') }}"
    # standalone backend only
```

(continues on next page)

(continued from previous page)

```
    src: "{{ neutron_ovn_user_ssl_cert }}"
  - name: "neutron_ovn_{{ ansible_facts['hostname'] }}"
    type: "private_key"
    dest: "{{ [neutron_ovn_conf_dir, neutron_ovn_ssl_key] | join('/') }}"
    owner: "{{ neutron_ovn_system_user_name }}"
    group: "{{ neutron_ovn_system_user_name }}"
    condition: "{{ (neutron_ovn_ssl and neutron_needs_openvswitch) }}"
    # standalone backend only
    src: "{{ neutron_ovn_user_ssl_key }}"
  - name: "neutron_ovn_{{ ansible_facts['hostname'] }}"
    type: ca_bundle
    dest: "{{ [neutron_ovn_conf_dir, neutron_ovn_ssl_ca_cert] | join('/') }}"
    owner: "{{ (neutron_services['neutron-server']['group'] in group_names) |␣
↪ternary(neutron_service_user_name, neutron_ovn_system_user_name) }}"
    group: "{{ (neutron_services['neutron-server']['group'] in group_names) |␣
↪ternary(neutron_service_user_name, neutron_ovn_system_user_name) }}"
    condition: "{{ (neutron_ovn_ssl and neutron_needs_openvswitch and neutron_
↪plugin_type == 'ml2.ovn') }}"
    # standalone backend only
    src: "{{ neutron_ovn_user_ssl_ca_cert }}"
  - name: "neutron_ovn_{{ ansible_facts['hostname'] }}"
    type: "certificate_chain"
    dest: "{{ [neutron_conf_version_dir, neutron_ovn_ssl_cert] | join('/') }}"
    owner: "{{ neutron_service_user_name }}"
    group: "{{ neutron_service_user_name }}"
    condition: "{{ (neutron_ovn_ssl and neutron_plugin_type == 'ml2.ovn' and␣
↪(filtered_neutron_services | length + uwsgi_neutron_services | length) > 0)␣
↪}}"
    # standalone backend only
    src: "{{ neutron_ovn_user_ssl_cert }}"
  - name: "neutron_ovn_{{ ansible_facts['hostname'] }}"
    type: "private_key"
    dest: "{{ [neutron_conf_version_dir, neutron_ovn_ssl_key] | join('/') }}"
    owner: "{{ neutron_service_user_name }}"
    group: "{{ neutron_service_user_name }}"
    condition: "{{ (neutron_ovn_ssl and neutron_plugin_type == 'ml2.ovn' and␣
↪(filtered_neutron_services | length + uwsgi_neutron_services | length) > 0)␣
↪}}"
    # standalone backend only
    src: "{{ neutron_ovn_user_ssl_key }}"
  - name: "neutron_ovn_{{ ansible_facts['hostname'] }}"
    type: ca_bundle
    dest: "{{ [neutron_conf_version_dir, neutron_ovn_ssl_ca_cert] | join('/')␣
↪}}"
    owner: "{{ neutron_service_user_name }}"
    group: "{{ neutron_service_user_name }}"
    condition: "{{ (neutron_ovn_ssl and neutron_plugin_type == 'ml2.ovn' and␣
↪(filtered_neutron_services | length + uwsgi_neutron_services | length) > 0)␣
↪}}"
```

(continues on next page)

(continued from previous page)

```
    # standalone backend only
    src: "{{ neutron_ovn_user_ssl_ca_cert }}"

# Define user-provided SSL certificates in:
# /etc/openstack_deploy/user_variables.yml
neutron_ovnnb_user_ssl_cert: ""
neutron_ovnnb_user_ssl_key: ""
neutron_ovnsb_user_ssl_cert: ""
neutron_ovnsb_user_ssl_key: ""

###
### DPDK Configuration
###

ovs_datapath: "netdev"
ovs_dpdk_pci_addresses: []
ovs_dpdk_driver: vfio-pci
ovs_dpdk_support: false
ovs_dpdk_lcore_mask: 1
ovs_dpdk_pmd_cpu_mask: 2
ovs_dpdk_socket_mem: "1024"
ovs_dpdk_nr_1g_pages: 0
ovs_dpdk_nr_2m_pages: 0

###
### Backend TLS
###

# Define if communication between haproxy and service backends should be
# encrypted with TLS.
neutron_backend_ssl: "{{ openstack_service_backend_ssl | default(False) }}"

# Storage location for SSL certificate authority
neutron_pki_dir: "{{ openstack_pki_dir | default('/etc/openstack_deploy/pki')␣
↪}}"

# Delegated host for operating the certificate authority
neutron_pki_setup_host: "{{ openstack_pki_setup_host | default('localhost') }}
↪"

# neutron server certificate
neutron_pki_keys_path: "{{ neutron_pki_dir ~ '/certs/private/' }}"
neutron_pki_certs_path: "{{ neutron_pki_dir ~ '/certs/certs/' }}"
neutron_pki_intermediate_cert_name: "{{ openstack_pki_service_intermediate_
↪cert_name | default('ExampleCorpIntermediate') }}"
neutron_pki_regen_cert: ""
neutron_pki_san: "{{ openstack_pki_san | default({'dns': [ansible_facts[
↪'hostname']], 'ip': [management_address]}) }}"
neutron_pki_backend: "{{ openstack_pki_backend | default('standalone') }}"
```

(continues on next page)

(continued from previous page)

```
neutron_pki_certificates:
  - name: "neutron_{{ ansible_facts['hostname'] }}"
    cn: "{{ ansible_facts['hostname'] }}"
    san: "{{ neutron_pki_san }}"
    # standalone backend only
    provider: ownca
    signed_by: "{{ neutron_pki_intermediate_cert_name }}"

# neutron destination files for SSL certificates
neutron_ssl_cert: "{{ neutron_conf_version_dir }}/neutron.pem"
neutron_ssl_key: "{{ neutron_conf_version_dir }}/neutron.key"

# Installation details for SSL certificates
neutron_pki_install_certificates:
  - name: "neutron_{{ ansible_facts['hostname'] }}"
    type: "certificate_chain"
    dest: "{{ neutron_ssl_cert }}"
    owner: "{{ neutron_system_user_name }}"
    group: "{{ neutron_system_user_name }}"
    # standalone backend only
    src: "{{ neutron_user_ssl_cert }}"
  - name: "neutron_{{ ansible_facts['hostname'] }}"
    type: "private_key"
    dest: "{{ neutron_ssl_key }}"
    owner: "{{ neutron_system_user_name }}"
    group: "{{ neutron_system_user_name }}"
    # standalone backend only
    src: "{{ neutron_user_ssl_key }}"

# Define user-provided SSL certificates
neutron_user_ssl_cert: ""
neutron_user_ssl_key: ""
```

CHAPTER

TWELVE

DEPENDENCIES

This role needs pip >= 7.1 installed on the target host.

CHAPTER

THIRTEEN

EXAMPLE PLAYBOOK

```yaml
---
- name: Installation and setup of Neutron
  hosts: neutron_all
  user: root
  roles:
    - role: os_neutron
      tags:
        - neutron-install
        - neutron-config
  vars:
    neutron_galera_address: "{{ internal_lb_vip_address }}"
    galera_root_user: root
  vars_prompt:
    - name: "galera_root_password"
      prompt: "What is galera_root_password?"
```

CHAPTER

FOURTEEN

TAGS

This role supports two tags: `neutron-install` and `neutron-config`. The `neutron-install` tag can be used to install and upgrade. The `neutron-config` tag can be used to maintain the configuration of the service.