ERICSSON

OpenStack Networking API in CEE in HP and Dell Multi-Server Deployment

Cloud Execution Environment

INTERWORK DESCRIPTION

7/155 19-CRA 119 1862/5-V1 Uen A

Contents

1 Introduction

This document describes the use of the Application Programming Interface (API) for networking in the Cloud Execution Environment (CEE). The API is based on the OpenStack networking component (Neutron).

1.1 API Version

The CEE Networking API is based on OpenStack Networking API v2.

1.2 Document References

This section contains the official OpenStack API reference.

1.2.1 API Design Base Reference

For the description of the API operations and extensions of networking, refer to the sections ‘‘Networking API v2.0’’ and ‘‘Networking API v2.0 extensions’’ in the *OpenStack API Complete Reference*.

This is a stored copy of the OpenStack API Reference document version that was the base for the development of this version of CEE.

Note: It is possible that the date on the front page differs from the revision date in this document. The front page shows the date on which the document was generated.

2 Prerequisites

The following sections list the prerequisites.

2.1 CEE Networking Configurations

CEE networking can be configured during the installation of the CEE. For details about the configurations, refer to the document *Configuration File Guide*.

Table 1 Neutron Configurations

Name	Neutron in CEE, Compared to Standard OpenStack Neutron
`ericsson_extreme`	• Uses a driver to manage Extreme Networks switches to provide L2 connectivity for tenant networks. • Uses a driver to manage Extreme Networks switches to provide L3 service to tenant networks and does not use L3 agent and floating IPs. • Uses a topology database, see Section 2.3 on page 3. • Uses several extensions that are explained in this document.
`ericsson_user_spec`	• Does not use L3 agent and floating IPs. • Uses several extensions that are explained in this document. • Can use additional features if properly configured by the user at installation of CEE • Uses a user specific ML2 driver to provide L2 connectivity for tenant networks. • Does not use L3 service plugin.

2.2 Segmentation IDs

Neutron uses a range of segmentation IDs for internal tenant separation. This range must be defined during the installation of the CEE.

Border Gateway (BGW) to CEE region connectivity also requires Virtual LANs (VLANs). The Cloud manager has to manage a range of IDs to be used for this. This range must not overlap with the range used internally by Neutron.

Table 2 VLAN Allocation Example

Name	Range	Description
Default	50-3581	Used by Neutron for tenant separation
BGW	3582-4094	Used for Neutron to BGW connectivity

Note: The table is an example, it does not mandate specific or default values. The values must be configured before the CEE region is installed. The configuration of values is described in the *Configuration File Guide*.

Each Neutron network requires one segmentation ID. Segmentation IDs have to be unique, the same ID cannot be used on several Neutron networks. Segmentation IDs on Neutron networks are static and cannot be changed after creation of the Neutron network.

2.3 Topology Database

The topology database contains information about network configuration, including a physical connectivity mapping between the switch ports and the compute blades plus BGWs. It defines the BGWs (Name, Extreme switch port they are connected to).

Note: It must be possible to configure an even number of BGWs.

This is a static configuration supplied in the CEE delivery. The names are used when binding Neutron ports to BGW.

Table 3 Topology Database Example

Name	Description
BGW-1	Definition of BGW-1
BGW-2	Definition of BGW-2
node-1	Compute blade 1, slot 1
node-2	Compute blade 2, slot 3

Note: The table is an example, it does not contain specific or default values.

3 Supported Operations

The following sections contain information about the API operations and API extensions in CEE.

3.1 Basic OpenStack Operations

For the detailed description of basic Networking API operations, refer to the section ‘‘Networking API v2.0’’ in the *OpenStack API Complete Reference*.

3.1.1 Limitations

For the CEE-specific limitations and recommendations, see Section 6 on page 37.

3.2 OpenStack Extensions

This section contains information about which Networking API extensions are supported.

- Agent Management Extension

  The following agents are supported:

  – `neutron-dhcp-agent`

  – `neutron-openvswitch-agent`

- Agent Schedulers

  The following agents are supported:

  – DHCP Agent Scheduler

  – Device Agent Scheduler

- List Extensions

  Note: Supported, but the reply contains unsupported operations.

- Networks multiple provider extension (networks)

- Quotas extension (quotas)

4 Deviations

This chapter describes the deviations between vanilla OpenStack networking and CEE networking.

4.1 Partly Supported Operations

The following operations are partly supported in CEE Networking:

Port binding extended attributes (ports)

Only the `host_id` attribute is supported.

External networks (external-net)

Support for dynamic creation of external networks. It allows the addition of external networks towards additional BGWs in runtime.

Operations for subnets

Only IPv4 is supported.

Network provider extended attributes (networks)

For non SR-IOV traffic, only the `vlan` network type is supported. For SR-IOV interfaces, the `flat` and `vlan` network types are supported, the option is applicable per region. In case of flat configuration, the Neutron Extreme ToR mechanism driver is not supported.

Note: In `flat` networking type with SR-IOV on Extreme switches, VLANs are not created.

Administrative State Down (ports)

Administration state `down` is not supported for trunk ports and their subports.
If the administration state of a trunk port is set to `down` at the CIC, it will not take effect to the trunk port and its subports at the relevant Compute node, although Neutron will show that the trunk port and its subports are administratively down.

Administrative State Down (device ports)

Administration state `down` is not supported for device ports.
If the administration state of a device port is set to `down` at the CIC, it will not take effect to the device port, although Neutron will show that the trunk device port is administratively down.

4.2 Changed Operations

L3 Network Extension

The `external_gateway_info` attribute is not used.

Instead, in the case of `ericsson_extreme`, gateways are added by using `Add Interface to Router`.

4.3 Added Operations

The following extensions are added:

- Neutron router `active_device`

  In the case of `ericsson_extreme`, used for selecting which Traffic Switch (ToR) acts as the active VRF for a specific Neutron router. Used for VRRP active-passive mode.

- Trunkport Extension

  The Neutron Trunkport extension allows Virtual Machines (VMs) to send VLAN tagged traffic to Neutron networks. In this way, each VLAN tag is associated with a Neutron network.

- Topology database (in the case of `ericsson_extreme`)

  - Populate and query topology database: connection between host port, Traffic Switch (ToR) ports and BGW ports.
  - Traffic Switch (ToR) Management: maintenance mode, recovery, and so on.

4.4 Not Supported Operations

The following operations are not supported in CEE Networking:

- Configurable external gateway modes
- Extra DHCP options (`extra-dhcp-opt`)
- The ExtraRoute Extension
- Firewall as a Service (FwaaS)
- Load Balancer as a Service (LBaaS)
- Metering labels and rules
- Security groups and rules (`security-groups`)
- Virtual Private Network as a Service (VPNaaS)

- Neutron router, in the case of , `ericsson_user_spec` configuration for CEE networking

5 Ericsson Extensions

This section describes the added CEE API extensions in detail.

5.1 Neutron Router active_device Extension

The `active_device` extension is used to control Virtual Router Redundancy Protocol (VRRP) configuration for a Neutron router, in the case of `ericsson_extreme` configuration for CEE networking

If `active_device` is not specified, VRRP is configured as master/master.

If `active_device` is specified, VRRP is configured as master/backup.

The Virtual Router (VR) in the `active_device` is the master.

The `active_device` is a new attribute supplied in the `add_router_interface` API.

The attribute is shown in Table 4.

Table 4 Attributes for Neutron Router active_device extension

Attribute	Type	Required	CRUD (1)	Default Value	Validation Constraints	Notes
active_device	uuid-str	No	C	None	N/A	If specified, VRRP is configured master/backup. Value specifies device for master VR.

(1) C: Use the attribute in create operations.

5.1.1 API Operation for active_device

This section discusses the operation for adding a router interface by using the OpenStack Networking `active_device` extension.

5.1.1.1 Add Router Interface

Table 5 Add Router Interface

Verb	URI	Description
PUT	/routers/*router_id*/add_router_interface	Adds an interface to logical router.

```
PUT /v2.0/routers/ a8628fb2-e6c9-4742-8496-2fb44d520717/⇒
add_router_interface
Accept: application/json

{"active_device_id":⇒
"4f0c049e-612e-491f-afa8-e0fab56337e8",⇒
"port_id":⇒
"6433edfc-d731-43a6-a8ee-fb5218777506"}
```

Example 1 Add Router Interface: JSON Request

```
{
"subnet_id": "89c09d0c-6033-413d-ab96-bdd4bfcbff72",
"tenant_id": "bc69dc7e33ac47b285562a4390d5468d",
"port_id": "6433edfc-d731-43a6-a8ee-fb5218777506",
"id": "a8628fb2-e6c9-4742-8496-2fb44d520717"
}
```

Example 2 Add Router Interface: JSON Response

5.2 Trunkport Extension

The Neutron Trunkport extension allows VMs to send VLAN tagged traffic to Neutron networks in a way that each VLAN tag is associated with a Neutron network.

5.2.1 Topology

A trunkport can be a simple trunkport or a subport.

A trunkport is a port that is connected to the VMs (representing a Network Interface Card - NIC).

Trunkports function much like a normal Neutron port.

For outgoing traffic from a VM, untagged traffic is forwarded to the network to which the trunkport is connected.

Incoming traffic from trunkports is untagged when entering the VM.

A subport is a port that is on a trunkport, and connected to a Neutron network.

For outgoing traffic from a VM, the VLAN tag is stripped off before the traffic is forwarded to the network connected to a subport. The tag from the VM is stripped before entering the associated Neutron network. When entering the associated Neutron network, the tag associated with the network is added. If the network is VxLAN based, a tunnel + key, associated with the network, is used to create the proper header.

For incoming traffic to a VM, from the network connected to a subport, the VLAN tag is added before the traffic is forwarded to a VM.

The topology of the trunkport concept is shown in Figure 1.

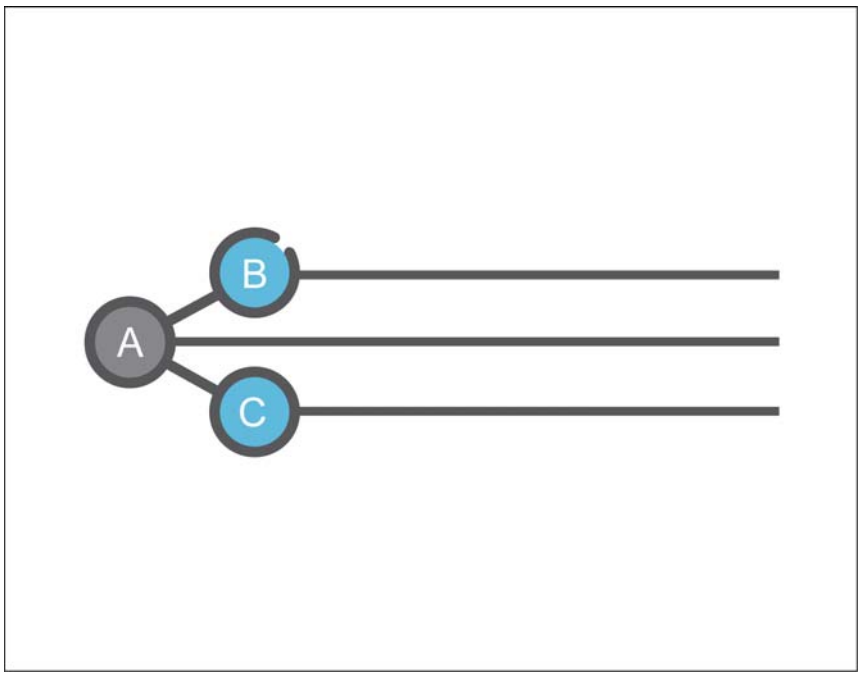

Figure 1 Trunkport Topology

A	Trunkport
B	Subport
C	Subport

5.2.2 Concepts

The `trunkport`-prefixed extended attributes for the **port** resource are shown in Table 6.

Table 6 Trunkport Attributes

Attribute	Type	Required	CRUD(1)	Default Value	Validation Constraints	Notes
trunkport:type	String	N/A	CR	None	Trunkport or subport	Defines whether a port is a trunkport or a subport.
trunkport:parent_id	uuid-str	N/A	CR	None	Valid `port id` of a trunkport	For subports, the id of the trunkport to which the subport is connected.
					Unset	For trunkports, unset.
trunkport:vid	Integer	N/A	CR	None	1-4094	For subports, segmentation ID that is used to access the given subport from the trunkport.
					Unset	For trunkports, unset.

(1) C : Use the attribute in create operations.
R: This attribute is returned in response to show and list operations.
U: You can update the value of this attribute.
D: You can delete the value of this attribute.

5.2.3 Trunkport API Operations

This section discusses the operations for setting and retrieving the trunkport port extension attributes for port objects.

5.2.3.1 List Ports

Table 7 List Ports

Verb	URI	Description
GET	/ports	Returns a list of ports with their attributes.

Normal response code: 200

Error response code: Unauthorized (401)

This operation returns all the ports defined in Neutron to which the given user has access.

```
{
"ports": [
{
"status": "DOWN",
"binding:host_id": null,
"name": "",
"allowed_address_pairs": [],
"admin_state_up": true,
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"tenant_id": "7ea98790cd854fb5a82ef3d41e5c156b",
"extra_dhcp_opts": [{"opt_value": "testfile.1",⇒
 "opt_name": "bootfile-name"},
{"opt_value": "123.123.123.45", "opt_name":⇒
 "server-ip-address"}, {"opt_value": "123.
123.123.123", "opt_name": "tftp-server"}],
"binding:vif_type": "ovs",
"device_owner": "",
"binding:capabilities": {"port_filter": true},
"mac_address": "fa:16:3e:52:92:3a",
"fixed_ips": [{"subnet_id": "99a8aea3-b9da-409d-a5e5-⇒
f45338ceb4d3"
, "ip_address": "172.24.4.228"}],
"id": "3c0c7a37-690a-43a8-8088-5d4c2c7f8484",
"security_groups": ["9bf6f19a-ba4a-470f-b8ce-28c9ad66556c"],
"device_id": "",
"trunkport:type": "trunk",
"trunkport:parent_id": "",
"trunkport:vid": ""
},
{
"status": "ACTIVE",
"binding:host_id": null,
"name": "",
"allowed_address_pairs": [],
"admin_state_up": true,
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"tenant_id": "7ea98790cd854fb5a82ef3d41e5c156b",
"extra_dhcp_opts": [],
"binding:vif_type": "ovs",
"device_owner": "compute:probe",
"binding:capabilities": {"port_filter": true},
"mac_address": "fa:16:3e:49:56:07",
"fixed_ips": [{"subnet_id"⇒
: "99a8aea3-b9da-409d-a5e5-f45338ceb4d3"⇒
, "ip_address": "172.24.4.227"}],
"id": "5212d40a-c2f5-4a5d-ad18-694658047654",
"security_groups": ["9bf6f19a-ba4a-470f-b8ce-28c9ad66556c"],
"device_id": "zvm2",
"trunkport:type": "subport",
"trunkport:parent_id": "3c0c7a37-690a-43a8-8088-5d4c2c7f8484",
"trunkport:vid": "10"
```

```
}
]
}
```

Example 3 List Ports: JSON Response

5.2.3.2 Show Port

Table 8 Show Port

Verb	URI	Description
GET	/ports/*port_id*	Returns details about a specific port including trunkport attributes.

Normal response code: 200

Error response codes: Unauthorized (401), Not Found (404)

This operation returns the port attributes of a port specified in the request URI. These attributes include the trunkport attributes.

```
{
"port":
{
"status": "DOWN",
"binding:host_id": null,
"name": "",
"allowed_address_pairs": [],
"admin_state_up": true,
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"tenant_id": "7ea98790cd854fb5a82ef3d41e5c156b",
"extra_dhcp_opts": [
{"opt_value": "testfile.1","opt_name": "bootfile-name"},
{"opt_value": "123.123.123.123", "opt_name": "tftp-server"},
{"opt_value": "123.123.123.45", "opt_name": "server-ip-address"}
],
"binding:vif_type": "ovs",
"device_owner": "",
"binding:capabilities": {"port_filter": true},
"mac_address": "fa:16:3e:52:92:3a",
"fixed_ips": [{"subnet_id"⇒
: "99a8aea3-b9da-409d-a5e5-f45338ceb4d3",
"ip_address": "172.24.4.228"}],
"id": "3c0c7a37-690a-43a8-8088-5d4c2c7f8484",
"security_groups": ["9bf6f19a-ba4a-470f-b8ce-28c9ad66556c"],
"device_id": "",
"trunkport:type": "trunk",
"trunkport:parent_id": "",
"trunkport:vid": ""
}
}
```

Example 4 Show Port with Trunkport Attributes: JSON Response

5.2.3.3 Create Port

Table 9 Create Port

Verb	URI	Description
POST	/ports	The operation creates a new port, and the supplied attributes show whether the port is a trunkport or a subport.

Normal response code: 200

Error response code: Unauthorized (401)

The operation creates a new port, and the supplied attributes show whether the port is a trunkport or a subport.

Note: When creating a subport, `mac_address` is copied from the parent port.

```
{
"port":
{
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"fixed_ips": [{"subnet_id"⇒
: "99a8aea3-b9da-409d-a5e5-f45338ceb4d3",
"ip_address": "172.24.4.230"}],
"admin_state_up": true,
"trunkport:type": "trunk"
}
}
```

Example 5 Create Port with Trunkport Attributes: JSON Request

```
{
"port":
{
"status": "DOWN",
"binding:host_id": null,
"name": "",
"allowed_address_pairs": [],
"admin_state_up": true,
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"tenant_id": "7ea98790cd854fb5a82ef3d41e5c156b",
"binding:vif_type": "ovs",
"device_owner": "",
"binding:capabilities": {"port_filter": true},
"mac_address": "fa:16:3e:43:3c:b7",
"fixed_ips": [{"subnet_id"⇒
: "99a8aea3-b9da-409d-a5e5-f45338ceb4d3",
"ip_address": "172.24.4.230"}],
"id": "055d27c0-0194-4782-be45-275ff2c95c61",
"security_groups": ["9bf6f19a-ba4a-470f-b8ce-28c9ad66556c"],
"device_id": "",
"trunkport:type": "trunk",
"trunkport:parent_id": "",
"trunkport:vid": ""
}
}
```

Example 6 Create Port with Trunkport Attributes: JSON Response

```
{
"port":
{
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"fixed_ips": [{"subnet_id"⇒
: "99a8aea3-b9da-409d-a5e5-f45338ceb4d3",
"ip_address": "172.24.4.230"}],
"admin_state_up": true,
"trunkport:type": "subport",
"trunkport:parent_id": "3c0c7a37-690a-43a8-8088-5d4c2c7f8484",
"trunkport:vid": "10"
}
}
```

Example 7 Create Port with Trunkport Attributes (Type Subport): JSON Request

```
{
"port":
{
"status": "DOWN",
"binding:host_id": null,
"name": "",
"allowed_address_pairs": [],
"admin_state_up": true,
"network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
"tenant_id": "7ea98790cd854fb5a82ef3d41e5c156b",
"binding:vif_type": "ovs",
"device_owner": "",
"binding:capabilities": {"port_filter": true},
"mac_address": "fa:16:3e:43:3c:b7",
"fixed_ips": [{"subnet_id"⇒
: "99a8aea3-b9da-409d-a5e5-f45338ceb4d3",
"ip_address": "172.24.4.230"}],
"id": "055d27c0-0194-4782-be45-275ff2c95c61",
"security_groups": ["9bf6f19a-ba4a-470f-b8ce-28c9ad66556c"],
"device_id": "",
"trunkport:type": "subport",
"trunkport:parent_id": "3c0c7a37-690a-43a8-8088-5d4c2c7f8484",
"trunkport:vid": "10"
}
}
```

Example 8 Create Port with Trunkport Attributes (Type Subport): JSON Response

5.2.3.4 Update Port

No attributes can be updated.

5.2.3.5 Delete Port

When deleting a trunkport, the subports that are connected to it are also deleted.

5.3 Topology Extension

The Neutron topology extension is used to describe switch devices and their connections to hosts, in the case of `ericsson_extreme` configuration for CEE networking.

Switch device information includes the management IP address for the switch and the status of the switch.

The status of the switch includes states like ACTIVE, ERROR, and MAINTENANCE.

5.3.1 Concepts

The extension introduces the following resources:

devices Defines switch devices.

device_ports Defines physical ports on switch devices.

hosts Defines hosts that can be connected to devices.

The attributes for the `devices` resource is shown in Table 10.

Table 10 Attributes for devices Resource

Attribute	Type	Required	CRUD(1)	Default Value	Validation Constraints	Notes
id	uuid-str	N/A	R	Generated	N/A	Id of the switch device
tenant_id	uuid-str	No	CR	N/A	No constrain t	Owner of the switch
name	String	No	CRU	None	N/A	Name of the switch
device_typ e	String	Yes	CR	None	N/A	Device type Only “TOR _SWITCH” supported
vendor	String	No	CR	None	N/A	Switch vendor

Table 10 Attributes for devices Resource

Attribute	Type	Required	CRUD(1)	Default Value	Validation Constraints	Notes
model	String	No	CR	None	N/A	Switch model
firmware_version	String	No	CRU	None	N/A	Firmware version
management_ip_address	String	No	CRU	None	IP address	Management IP address of the switch
user_name	String	No	CRU	None	N/A	Username for the switch
password	String	No	CRU	None	N/A	Password for the switch
vr_total	int	No	CR	None	N/A	Number of VRs supported by the switch
admin_state_up	Bool	No	CRU	None	N/A	Admin state of the switch
status	String	No	RU	None	One of 'ACTIVE', 'ERROR', 'MAINTENANCE', 'RECOVER', 'UNSUPERVISED'	Status of the switch

(1) C : Use the attribute in create operations.
R: This attribute is returned in response to show and list operations.
U: You can update the value of this attribute.
D: You can delete the value of this attribute.

The attributes for the `device_ports` resource is shown in Table 11.

Table 11 Attributes for device_ports

Attribute	Type	Required	CRUD(1)	Default Value	Validation Constraints	Notes
id	uuid-str	N/A	R	Generated	N/A	Id of the switch port

Table 11 Attributes for device_ports

Attribute	Type	Required	CRUD[(1)]	Default Value	Validation Constraints	Notes
device_id	uuid-str	No	CR	N/A	N/A	Owner of the switch device port
port_type	String	No	R	N/A	N/A	Define type of port. One of 'ISC', 'SERVER', 'GATEWAY', 'DISCONNECTED'
tenant_id	uuid-str	No	CR	N/A	No constraint	Owner of the switch port
name	String	No	CRU	None	N/A	Name of the switch port
physical_network	String	No	CR	None	N/A	The physical network the port is used for
slot_id	String	No	CR	None	N/A	N/A
port_id	int	No	CR	None	N/A	Physical port number of the switch
master	Bool	No	CR	None	N/A	Defines if the port is a master in a LAG
slave_ports	list (uuid-str)	No	CR	None	N/A	List of the slave ports in LAG

Table 11 Attributes for device_ports

Attribute	Type	Required	CRUD[1]	Default Value	Validation Constraints	Notes
admin_state_up[2]	Bool	No	CRU	None	N/A	Admin state of the switch port
status	String	No	R	None	N/A	Status of the switch port

(1) C : Use the attribute in create operations.
R: This attribute is returned in response to show and list operations.
U: You can update the value of this attribute.
D: You can delete the value of this attribute.
(2) Administrative state `down` is not supported for device ports, see Section 4.1 on page 5.

The attributes for the `hosts` resource is shown in Table 12.

Table 12 Attributes for hosts

Attribute	Type	Required	CRUD[1]	Default Value	Validation Constraints	Notes
id	uuid-str	N/A	R	Generated	N/A	Id of the host
tenant_id	uuid-str	No	CR	N/A	No constraint	Owner of the host
name	String	No	CRU	None	N/A	Name of the host
network_host	Bool	No	CRU	None	N/A	True for network hosts
compute_host	Bool	No	CRU	None	N/A	True for compute hosts
admin_state_up	Bool	No	CRU	None	N/A	Admin state of the host
status	String	No	R	None	N/A	Status of the host

(1) C : Use the attribute in create operations.
R: This attribute is returned in response to show and list operations.
U: You can update the value of this attribute.
D: You can delete the value of this attribute.

5.3.2 Topology API Operations

This section discusses the operations for the topology extension.

5.3.2.1 List Devices

Table 13 List Devices

Verb	URI	Description
GET	/topology/devices	Returns a list of the switch devices.

Normal response code: 200

Error response codes: Unauthorized (401)

This operation returns a list of all switch devices defined in neutron.

Note: This operation does not require a request body; this operation returns a response body.

```
GET /v2.0/topology/devices
Accept: application/json
```

Example 9 List Devices: JSON Request

```
{
"devices":
[{
"status": "ACTIVE",
"model": "SummitX670V",
"vr_total": 189,
"vendor": "extreme",
"name": "DC154_SWA_X670V",
"admin_state_up": true,
"device_type": "TOR_SWITCH",
"firmware_version": "15.6.1.4",
"password": "NetworkAdmin",
"user_name": "network_admin",
"id": "91d4e7e4-1385-4c47-9b1f-69e04a0954bc",⇒
"management_ip_address": "192.168.6.1"
},
{
"status": "ACTIVE",
"model": "SummitX670V",
"vr_total": 189,
"vendor": "extreme",
"name": "DC154_SWB_X670V",
"admin_state_up": true,
"device_type": "TOR_SWITCH",
"firmware_version": "15.6.1.4",
"password": "NetworkAdmin",
"user_name": "network_admin",
"id": "06b7e54a-b0ec-4090-a953-5c0031842889",⇒
"management_ip_address": "192.168.6.2"
}]
}
```


Example 10 List Devices: JSON Response

5.3.2.2 Show Device

Table 14 Show Device

Verb	URI	Description
GET	/topology/devices/*device_id*	Returns the details of a switch device.

Normal response code: 200

Error response codes: Unauthorized (401), Forbidden (403), Not Found (404)

This operation returns the details for a specific device, whose identifier is specified on the request URI. Users can control which attributes are returned by using the fields query parameter.

Note: This operation does not require a request body; this operation returns a response body.

```
GET /v2.0/topology/devices/06b7e54a-b0ec-4090-a953-5c0031842889
Accept: application/json
```

Example 11 Show Device: JSON Request

```
{
"device":
{"status": "ACTIVE",
"model": "SummitX670V",
"vr_total": 189,
"vendor": "extreme",
"name": "DC154_SWB_X670V",
"admin_state_up": true,
"device_type": "TOR_SWITCH",
"firmware_version": "15.6.1.4",
"password": "NetworkAdmin",
"user_name": "network_admin",
"id": "06b7e54a-b0ec-4090-a953-5c0031842889",⇒
"management_ip_address": "192.168.6.2"
}
}
```

Example 12 Show Device: JSON Response

5.3.2.3 Create Device

Table 15 Create Device

Verb	URI	Description
POST	/topology/devices	Creates a new switch device.

Normal response code: 201

Error response codes: Unauthorized (401), Bad Request (400)

This operation creates a new switch device.

Note: This operation requires a request body. This operation returns a response body.

```
POST /v2.0/topology/devices
Accept: application/json
{
"device":
{
"vendor": "extreme",
"device_type": "TOR_SWITCH",
"admin_state_up": true,
"model": "SummitX670V",
"password": "NetworkAdmin",
"user_name": "network_admin",
"firmware_version": "15.6.1.4",
"management_ip_address": "192.168.6.5"
```

Example 13 Create Device: JSON Request

```
{
"device":
{
"status": "ACTIVE",
"vendor": "extreme",
"device_type": "TOR_SWITCH",
"firmware_version": "15.6.1.4",
"password": "NetworkAdmin",
"id": "98f1a2c7-5f6f-413c-a364-d545c17f96ee",⇒
"management_ip_address": "192.168.6.5",
"name": "",
"admin_state_up": true,
"model": "SummitX670V",
"user_name": "network_admin"
}
}
```

Example 14 Create Device: JSON Response

5.3.2.4 Update Device

Table 16 Update Device

Verb	URI	Description
PUT	/topology/devices/*device_id*	Updates a switch device

Normal response code: 200

Error response codes: Unauthorized (401), Bad Request (400)

This operation updates attributes on a switch device.

Note: This operation requires a request body. This operation returns a response body.

```
PUT /v2.0/topology/devices/b06fdcfc-5c3e-466e-b0c2-da4669e27cdd
Accept: application/json
{
"device":
{
"status": "MAINTENANCE
}
}
```

Example 15 Update Device: JSON Request

```
{
"device":
{
"status": "MAINTENANCE",
"model": "SummitX670V",
"vr_total": 189,
"vendor": "extreme",
"name": "DC154_SWA_X670V",
"admin_state_up": true,
"device_type": "TOR_SWITCH",
"firmware_version": "15.6.1.4",
"password": "NetworkAdmin",
"user_name": "network_admin",
"id": "b06fdcfc-5c3e-466e-b0c2-da4669e27cdd",⇒
"management_ip_address": "192.168.6.1"
}
}
```

Example 16 Update Device: JSON Response

5.3.2.5 Delete Device

Table 17 Delete Device

Verb	URI	Description
DELETE	/topology/devices/*device_id*	Deletes a switch device.

Normal response code: 204

Error response codes: Unauthorized (401), Not Found (404), Conflict (409)

This operation removes a switch device.

Note: This operation does not require a request body. This operation does not return a response body.

```
DELETE /v2.0/topology/devices/98f1a2c7-5f6f-413c-a364-d545c17f96ee
Accept: application/json
```

Example 17 Delete Device: JSON Request

5.3.2.6 List Device Ports

Table 18 List Device Ports

Verb	URI	Description
GET	/topology/device_ports	Returns a list of the switch ports.

Normal response code: 200

Error response codes: Unauthorized (401)

This operation returns a list of all switch ports defined in neutron.

Note: This operation does not require a request body; this operation returns a response body.

```
GET /v2.0/topology/device_ports
Accept: application/json
```

Example 18 List Device Ports: JSON Request

```
{
"device_ports":
[{
"status": "ACTIVE",
"name": "DC154_SWA_X670V_port_5",
"admin_state_up": false,
"slot_id": null,
"physical_network": "default",
"port_type": "SERVER",
"port_id": 5,
"id": "01f76bbc-8ac9-471a-a152-39c7aaa10314",
"device_id": "b06fdcfc-5c3e-466e-b0c2-da4669e27cdd"
},
{
"status": "ACTIVE",
"name": "DC154_SWB_X670V_port_7",
"admin_state_up": true,
"slot_id": null,
"physical_network": "default",
"port_type": "SERVER",
"port_id": 7,
"id": "0492fffa-6c3a-45f0-a1e8-2e2f6c64dddb",
"device_id": "06b7e54a-b0ec-4090-a953-5c0031842889"
},
{
"status": "ACTIVE",
"name": "DC154_SWB_X670V_port_9",
"admin_state_up": true,
"slot_id": null,
"physical_network":
"default", "port_type": "SERVER",
"port_id": 9,
"id": "31dc2a36-a4e3-44b4-a03d-55e322df0249",
"device_id": "06b7e54a-b0ec-4090-a953-5c0031842889"
}]
}
```


Example 19 List Device Ports: JSON Response

5.3.2.7 Show Device Port

Table 19 Show Device Port

Verb	URI	Description
GET	/topology/device_ports/*device_port_id*	Returns the details of a switch port.

Normal response code: 200

Error response codes: Unauthorized (401), Forbidden (403), Not Found (404)

This operation returns the details for a specific device port, whose identifier is specified on the request URI. Users can control which attributes are returned by using the fields query parameter.

Note: This operation does not require a request body; this operation returns a response body.

```
GET /v2.0/topology/device_ports/⇒
73b23ff1-24f7-418c-9048-6b52f5edf8d2
Accept: application/json
```

Example 20 Show Device Port: JSON Request

```
{
"device_port":
{
"status": "ACTIVE",
"name": "",
"admin_state_up": true,
"slot_id": null,
"physical_network": "loopback",
"port_type": "DISCONNECTED",
"port_id": 9999,
"id": "73b23ff1-24f7-418c-9048-6b52f5edf8d2",
"device_id": "06b7e54a-b0ec-4090-a953-5c0031842889"
}
}
```

Example 21 Show Device Port: JSON Response

5.3.2.8 Create Device Port

Table 20 Create Device Port

Verb	URI	Description
POST	/topology/device_ports	Creates a new switch port.

Normal response code: 201

Error response codes: Unauthorized (401), Bad Request (400)

This operation creates a new device port.

Note: This operation requires a request body. This operation returns a response body.

```
POST /v2.0/topology/device_ports
Accept: application/json
{
"device_port":
{
"port_id": "7999",
"device_id": "06b7e54a-b0ec-4090-a953-5c0031842889",⇒
"physical_network": "default",
"name": "mytestdevport",
"admin_state_up": true
}
}
```

Example 22 Create Device Port: JSON Request

```
{
"device_port":
{
"status": "ACTIVE",
"name": "mytestdevport",
"admin_state_up": true,
"slot_id": null,
"physical_network": "default",
"port_type": "DISCONNECTED",
"port_id": 7999,
"id": "273feb26-2a7b-4370-961c-51a8276f1728",
"device_id": "06b7e54a-b0ec-4090-a953-5c0031842889"
}
}
```

Example 23 Create Device Port: JSON Response

5.3.2.9 Update Device Port

Table 21 Update Device Port

Verb	URI	Description
PUT	/topology/device_ports/*device_port_id*	Updates a switch port.

Normal response code: 200

Error response codes: Unauthorized (401), Bad Request (400)

This operation updates attributes on a device port.

Note: This operation requires a request body. This operation returns a response body.

```
PUT /v2.0/topology/device_ports/⇒
273feb26-2a7b-4370-961c-51a8276f1728
Accept: application/json
{
"device_port":
{
"admin_state_up": "False"
}
}
```

Example 24 Update Device Port: JSON Request

```
{
"device_port":
{
"status": "ACTIVE",
"name": "mytestdevport",
"admin_state_up": false,
"slot_id": null,
"physical_network": "default",
"port_type": "DISCONNECTED",
"port_id": 7999,
"id": "273feb26-2a7b-4370-961c-51a8276f1728",
"device_id": "06b7e54a-b0ec-4090-a953-5c0031842889"
}
}
```

Example 25 Update Device Port: JSON Response

5.3.2.10 Delete Device Port

Table 22 Delete Device Port

Verb	URI	Description
DELETE	/topology/device_ports/*device_port_id*	Deletes a switch port.

Normal response code: 204

Error response codes: Unauthorized (401), Not Found (404), Conflict (409)

This operation removes a switch port.

Note: This operation does not require a request body. This operation does not return a response body.

```
DELETE /v2.0/topology/device_ports/⇒
273feb26-2a7b-4370-961c-51a8276f1728
Accept: application/json
```

Example 26 Delete Device Port: JSON Request

5.3.2.11 Add Device Port Link

Table 23 Add Device Port Link

Verb	URI	Description
PUT	/topology/device_ports/*device_port_id*/add_device_port_link	Adds device port link.

Normal response code: 200

Error response codes: Unauthorized (401), Bad Request (400)

This operation adds a link from device port to host.

Note: This operation requires a request body. This operation returns a response body.

```
PUT /v2.0/topology/device_ports/⇒
f21d9795-e881-4934-b023-a96a1c53cdc4/add_device_port_link
Accept: application/json
{
"host_id": "27141b8a-4103-4745-9ffa-ad768fe37511"
}
```

Example 27 Add Device Port Link: JSON Request

```
{
"peer_host": "27141b8a-4103-4745-9ffa-ad768fe37511",
"device_port": "f21d9795-e881-4934-b023-a96a1c53cdc4",⇒
"device_port_type": "SERVER"
}
```

Example 28 Add Device Port Link: JSON Response

5.3.2.12 Remove Device Port Link

Table 24 Remove Device Port Link

Verb	URI	Description
PUT	/topology/device_ports/*device_port_id*/remove_device_port_link	Removes device port link.

Normal response code: 200

Error response codes: Unauthorized (401), Not Found (404)

This operation removes a device port link.

Note: This operation does not require a request body. This operation does not return a response body.

```
PUT /v2.0/topology/device_ports/⇒
f21d9795-e881-4934-b023-a96a1c53cdc4/remove_device_port_link
Accept: application/json
```

Example 29 Remove Device Port Link: JSON Request

5.3.2.13 List Hosts

Table 25 List Hosts

Verb	URI	Description
GET	/topology/hosts	Returns a list of hosts.

Normal response code: 200

Error response codes: Unauthorized (401)

This operation returns a list of all hosts defined in neutron.

Note: This operation does not require a request body; this operation returns a response body.

```
GET /v2.0/topology/hosts
Accept: application/json
```

Example 30 List Hosts: JSON Request

```
{
"hosts":
[{
"status": "ACTIVE",
"name": "compute-0-5",
"admin_state_up": true,
"compute_host": true,
"network_host": false,
"id": "10b711e9-bc83-425d-9acd-0f27faed3e6f"
},
{
"status": "ACTIVE",
"name": "cic-0-2",
"admin_state_up": true,
"compute_host": false,
"network_host": true,
"id": "27141b8a-4103-4745-9ffa-ad768fe37511"
},
{
"status": "ACTIVE",
"name": "BGW-1",
"admin_state_up": true,
"compute_host": false,
"network_host": false,
"id": "4349b87f-4e43-4b93-a003-db031f7aab0c"
}]
}
```


Example 31 List Hosts: JSON Response

5.3.2.14 Show Host

Table 26 Show Host

Verb	URI	Description
GET	/topology/hosts/*host_id*	Returns the details of a host.

Normal response code: 200

Error response codes: Unauthorized (401), Forbidden (403), Not Found (404)

This operation returns the details for a specific host, whose identifier is specified on the request URI. Users can control which attributes are returned by using the fields query parameter.

Note: This operation does not require a request body; this operation returns a response body.

```
GET /v2.0/topology/hosts/10b711e9-bc83-425d-9acd-0f27faed3e6f
Accept: application/json
```

Example 32 Show Host: JSON Request

```
{
"host":
{
"status": "ACTIVE",
"name": "compute-0-5",
"admin_state_up": true,
"compute_host": true,
"network_host": false,
"id": "10b711e9-bc83-425d-9acd-0f27faed3e6f"
}
}
```

Example 33 Show Host: JSON Response

5.3.2.15 Create Host

Table 27 Create Host

Verb	URI	Description
POST	/topology/hosts	Create a new host.

Normal response code: 201

Error response codes: Unauthorized (401), Bad Request (400)

This operation creates a new host.

Note: This operation requires a request body. This operation returns a response body.

```
POST /v2.0/topology/hosts
Accept: application/json
{
"host":
{
"compute_host": false,
"network_host": false,
"name": "testhost",
"admin_state_up": true
}
}
```

Example 34 Create Host: JSON Request

```
{
"host":
{
"status": "ACTIVE",
"name": "testhost",
"admin_state_up": true,
"compute_host": false,
"network_host": false,
"id": "6d389551-200a-47ee-8e20-2ccf06244f2f"
}
}
```

Example 35 Create Host: JSON Response

5.3.2.16 Update Host

Table 28 Update Host

Verb	URI	Description
PUT	/topology/hosts/*host_id*	Update a host.

Normal response code: 200

Error response codes: Unauthorized (401), Bad Request (400)

This operation updates attributes on a host.

Note: This operation requires a request body. This operation returns a response body.

```
PUT /v2.0/topology/hosts/6d389551-200a-47ee-8e20-2ccf06244f2f
Accept: application/json
{
"host":
{
"admin_state_up": "False"
}
}
```

Example 36 Update Host: JSON Request

```
{
"host":
{
"status": "ACTIVE",
"name": "testhost",
"admin_state_up": false,
"compute_host": false,
"network_host": false,
"id": "6d389551-200a-47ee-8e20-2ccf06244f2f"
}
}
```

Example 37 Update Host: JSON Response

5.3.2.17 Delete Host

Table 29 Delete Host

Verb	URI	Description
DELETE	/topology/hosts/*host_id*	Delete a host.

Normal response code: 204

Error response codes: Unauthorized (401), Not Found (404), Conflict (409)

Note: This operation does not require a request body. This operation does not return a response body.

```
DELETE /v2.0/topology/hosts/6d389551-200a-47ee-8e20-2ccf06244f2f
Accept: application/json
```

Example 38 Delete Host: JSON Request

6 Limitations

This section lists the limitations and recommendations for CEE Networking.

6.1 Limitations

This section contains the limitations of CEE Networking.

6.1.1 Segmentation IDs

The total number of segmentation IDs is limited to 4094, see Section 2.2 on page 2.

6.1.2 Routers

The number of Neutron routers is limited by the number of Neutron networks connected to them from VMs in the case of `ericsson_extreme` configuration for CEE networking. The limiting factor is the number of VRRP instances. In CEE, one VRRP instance is used per internal network. However, networks between Neutron router and BGW are not limiting, because BGW does not have VRRP. The limitation differs for different hardwares.

6.1.3 Deleting Trunkport and Subports

If a trunk port is deleted before the associated subports have been deleted, there are remains left in Neutron that prevent using the same IP address with a different MAC number. If a new trunk port and subports are created with the same IP address, this can prevent VMs on the ports to get an IP address.

To delete a trunk port with subports, do the following:

1. Delete the related subports.

2. Delete the trunk port.

6.1.4 VFs Do Not Obtain the IP Addresses Assigned to Them by the Neutron DHCP Agent

The provisioned VMs which have SR-IOV VF interfaces do not obtain the IP addresses that the Neutron DHCP agent assigns. Therefore they must be configured manually in the guest VM.

6.1.5 Switch Status Is Not Updated During Reboot of Switch

If Extreme switches are rebooted manually using EXOS CLI, the command `neutron device-show <device_id>` cannot track the state of the rebooted device.

6.1.6 Extreme Switch Cannot Be Rebooted by the Neutron device-reboot Command

Rebooting the Extreme switch is not possible from Neutron, as the `neutron device-reboot <device_id>` command is not supported.

6.1.7 CEE VLAN Tracking Is Not Recovered in Extreme Switches

When an external network is created that includes BGW port(s), and it is connected to a specific VR, Neutron will enable VLAN to track the BGW tenant network. This VLAN tracking is not recovered automatically during the switch recovery.

To ensure that VLAN tracking is enabled, follow these steps:

1. Before performing the device recovery operation:

    a. Extract the ID of the virtual router inside the Extreme switch:

    ```
    <VR_ID>=VR_$(neutron router-show <ROUTER_NAME> -F id -f value)
    ```

    b. Display the VLAN configuration of the switches associated to that router:

    ```
    ssh <SWITCH_LOGIN_NAME>@<SWA_IP> show vlan <VR_ID> | grep tenant
    ssh <SWITCH_LOGIN_NAME>@<SWB_IP> show vlan <VR_ID> | grep tenant
    ```

    SWA_IP and *SWB_IP* are the IP addresses of the switches.

    c. Extract *<INTERNAL_VID>*, *<SWA_VID>*, and *<SWB_VID>* from the displayed results.

    VLAN ID (VID) is displayed after the `tenant_` string, with the same value visible in the second column. *<INTERNAL_VID>* has the same VID value on both switches, while *<SWA_VID>* and *<SWB_VID>* have different values.

2. After performing the device recovery operation, issue the following commands on the corresponding switches:

```
configure vrrp vlan tenant_<INTERNAL_VID> vrid 1 add track-vlan tenant_<SWA_VID>
configure vrrp vlan tenant_<INTERNAL_VID> vrid 1 add track-vlan tenant_<SWB_VID>
```

3. Check the VLAN configuration on the switches to confirm that VLAN tracking is enabled.

7 Concepts and Use Cases

The following sections describe the various concepts and use cases that are connected to the CEE Networking API.

7.1 General Terms

Neutron Router Virtual Routing and Forwarding (VRF) implemented in the Traffic Switch (ToR) in the case of `ericsson_extreme` configuration for CEE networking.

Neutron Network L2 broadcast domain

Neutron Subnet Definition of a subnet. It s attached to a Neutron network. Used to configure the Neutron DHCP service.

7.2 IP Address Management

The following three methods are available for IP address management:

- Application handles the IP addresses on its own and do not involve Neutron.

  Note: IP addresses can be privately handled only if ARP spoofing is switched off. It is not recommended to switch off ARP spoofing, as it is a security feature of the system.

  No Neutron DHCP service is used. Only L2 services from Neutron can be used.

  Note: Nova requires a subnet attached to a Neutron network in order to start VMs on the network. User is required to create a dummy subnet with DHCP disabled as a workaround.

- Application uses Neutron for IP address management.

  This can be done in several ways: the application can specify IP addresses per ports or only specify IP subnets and let Neutron allocate the addresses.

- A mix of the above two.

  In this case, it is the responsibility of the application to align the configuration view of the application with the configuration view of Neutron.

IP addresses in Neutron can be reused on different Neutron networks. IP addresses known by Neutron must have matching Neutron subnets.

To manually assign an IP address to a port use the `fixed_ips` attribute in the Neutron port when creating the Neutron port.

The `fixed_ips` attribute includes `ip_address` and `subnet_id`. Please refer to the section ‘‘Networking API v2.0’’ in the *OpenStack API Complete Reference*.

7.2.1 Neutron DHCP

Neutron DHCP can be enabled or disabled per subnet.

This is done by using the `enable_dhcp` attribute on the Neutron subnet.

```
enable_dhcp <True/False>
```

7.3 MAC Address Management

The following two methods are available for MAC address management:

- The application handles MAC addresses on its own, and provides the MAC address to Neutron when creating the Neutron port.
- The application allows Neutron to allocate a MAC address when creating the Neutron port.

MAC addresses have to be unique per Neutron network, but can be reused on different Neutron networks.

To manually assign a MAC address use the `mac_address` attribute in the Neutron port when creating the Neutron port, refer to the sections ‘‘Networking API v2.0’’ and ‘‘Networking API v2.0 extensions’’ in the *OpenStack API Complete Reference*.

Neutron-allocated MAC addresses have a three byte fixed prefix. The rest will be a random number unique per Neutron network. Refer to *Configuration File Guide*, for the prefix used in the Neutron configuration file.

Note: It is advisable to use the local administered MAC ranges.

7.4 IPv6

Neutron L3 Networks are not supported for IPv6, thus no Network Routing or Address Management for IPv6 is supported.

7.5 Internal Neutron Network

This section describes how to connect VMs using internal L2. Create Neutron network, create Neutron ports on Neutron network, and then create VMs using those Neutron ports.

1. Create Neutron network

   For the procedure of creating a Neutron network, refer to the sections ‘‘Networking API v2.0’’ and ‘‘Networking API v2.0 extensions’’ in the *OpenStack API Complete Reference*.

2. Create Neutron subnet on Neutron network

   Nova requires a subnet attached to a Neutron network in order to start VMs on the network. User is required to create a dummy subnet with DHCP disabled as a workaround.

   For the procedure of creating a Neutron subnet on Neutron network, refer to the section ‘‘Networking API v2.0’’ in the *OpenStack API Complete Reference*.

3. Create Neutron ports on Neutron network

   For the procedure of creating Neutron ports on a Neutron network, refer to the sections ‘‘Networking API v2.0’’ and ‘‘Networking API v2.0 extensions’’ in the*OpenStack API Complete Reference*.

4. Create VMs using Neutron ports

7.6 Internal Neutron Network with Neutron IP Address Management

This chapter describes how to connect VMs using internal L3.

1. Create Neutron network

   For the procedure of creating a Neutron network, refer to the section ‘‘Networking API v2.0’’ in the *OpenStack API Complete Reference*.

2. Create Neutron subnet on Neutron network

   For the procedure of creating a Neutron subnet on Neutron network, refer to the section ‘‘Networking API v2.0’’ in the *OpenStack API Complete Reference*.

3. Create Neutron ports on Neutron network

   For the procedure of creating Neutron ports on a Neutron network, refer to the sections ‘‘Networking API v2.0’’ and ‘‘Networking API v2.0 extensions’’ in the *OpenStack API Complete Reference*.

4. Create VMs using Neutron ports.

7.7 L2 Connection to BGW

The following subsections describe how to connect VMs to BGW using an L2 network, in the case of `ericsson_extreme` configuration of CEE networking.

The L2 BGW connection is shown in Figure 2.

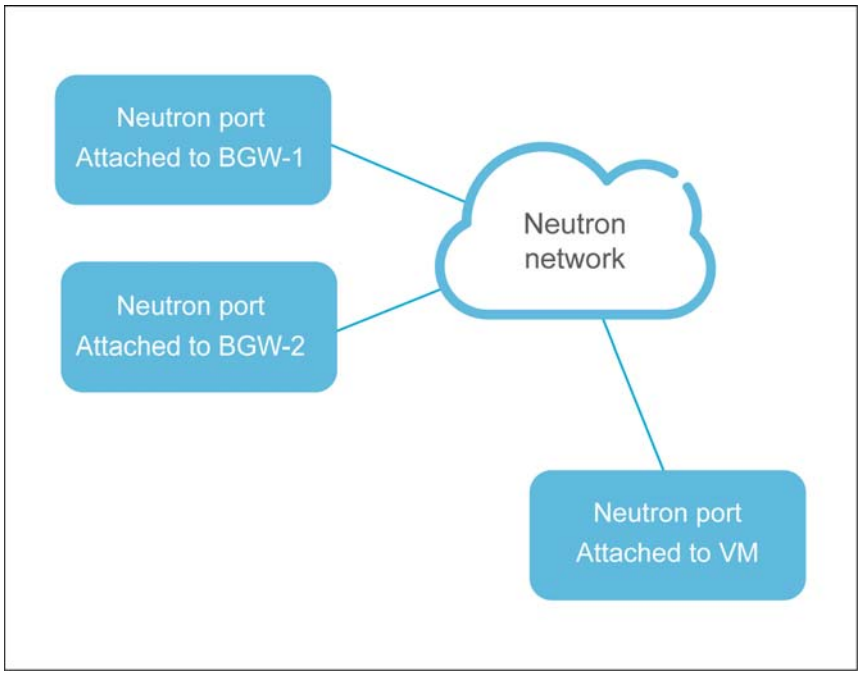

Figure 2 L2 BGW Connection

7.7.1 Create Neutron Network

Create a Neutron network using the `provider` network extension (`provider ID`). The Cloud manager is recommended for the management of these segmentation IDs. The IDs must be picked from the BGW pool, see Section 2.2 on page 2.

For the procedure of creating a Neutron network, refer to the section "Networking API v2.0" in the *OpenStack API Complete Reference*.

For the `provider` Extended Attributes for Networks, refer to the section "Networking API v2.0 extensions" in the *OpenStack API Complete Reference*.

`provider:segmentation_id`	*<VID>*
`provider:network_type`	`vlan`(1)
`provider:physical_network`	`default`

(1) Only the `vlan` network type is supported.

7.7.2 Create Neutron Subnet

Create Neutron subnet on Neutron network. A subnet is required regardless if IP address management is going to be handled by Neutron or not.

nova requires a subnet attached to a Neutron network in order to start VMs on the network. User is required to create a dummy subnet with DHCP disabled as a workaround in cases when IP address management is handled by the application.

For the procedure of creating a Neutron subnet on Neutron network, refer to the section ‘‘Networking API v2.0’’ in the *OpenStack API Complete Reference*.

7.7.3 Create Neutron Ports Connected to BGWs

1. Create two Neutron ports on the Neutron network.

   For the procedure of creating Neutron ports on a Neutron network, refer to the sections ‘‘Networking API v2.0’’ and ‘‘Networking API v2.0 extensions’’ in the *OpenStack API Complete Reference*.

2. Bind one Neutron port to BGW-1 and the other Neutron port to BGW-2.

   For the procedure of binding extended attributes for ports, refer to the sections ‘‘Networking API v2.0’’ and ‘‘Networking API v2.0 extensions’’ in the *OpenStack API Complete Reference*

BGW-1:

`device_owner`	`baremetal:BGW-1`
`network_id`	*`<BGW Net id>`*
`binding:host_id`	`BGW-1`

BGW-2:

`device_owner`	`baremetal:BGW-2`
`network_id`	*`<BGW Net id>`*
`binding:host_id`	`BGW-2`

7.7.4 Configure BGW

The traffic is configured to reach the BGW. The BGW must be configured to handle this traffic.

For the configuration of BGW for this segmentation ID, see Figure 3.

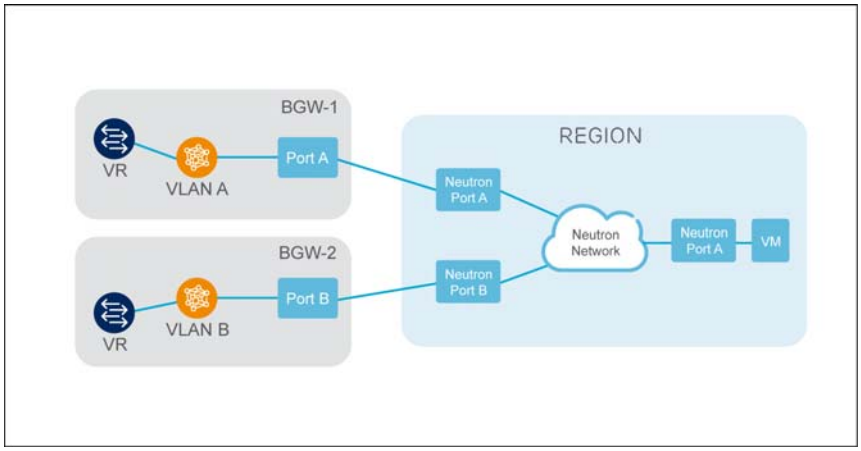

Figure 3 L2 BGW Configuration

7.7.4.1 Create a Virtual Router

Create VRs in the Border Gateways.

7.7.4.2 Configure Virtual Router

Configure the VRs with their applicable parameters. Both IPv4 and IPv6 can be used.

Static routing together with VRRP v3 is recommended, as described in Section 7.7.4.3 on page 44.

7.7.4.3 Create VLANs in the Virtual Router and Configure VRRP v3

After a VR is created, and support for any required routing protocols are added, you can create a VLAN in a VR.

VLAN names must reflect what they are used for.

VLAN IDs must be unique within the same broadcast domain.

7.7.4.4 Add Ports to Virtual Router

Add the applicable physical ports to the VRs.

7.8 L3 connection to BGW using Neutron Router

The following subsections describe how to connect VMs to BGW using Neutron Routers, in the case of `ericsson_extreme` configuration of CEE networking or ODL-based SDN.

The L3 BGW connection is shown in Figure 4.

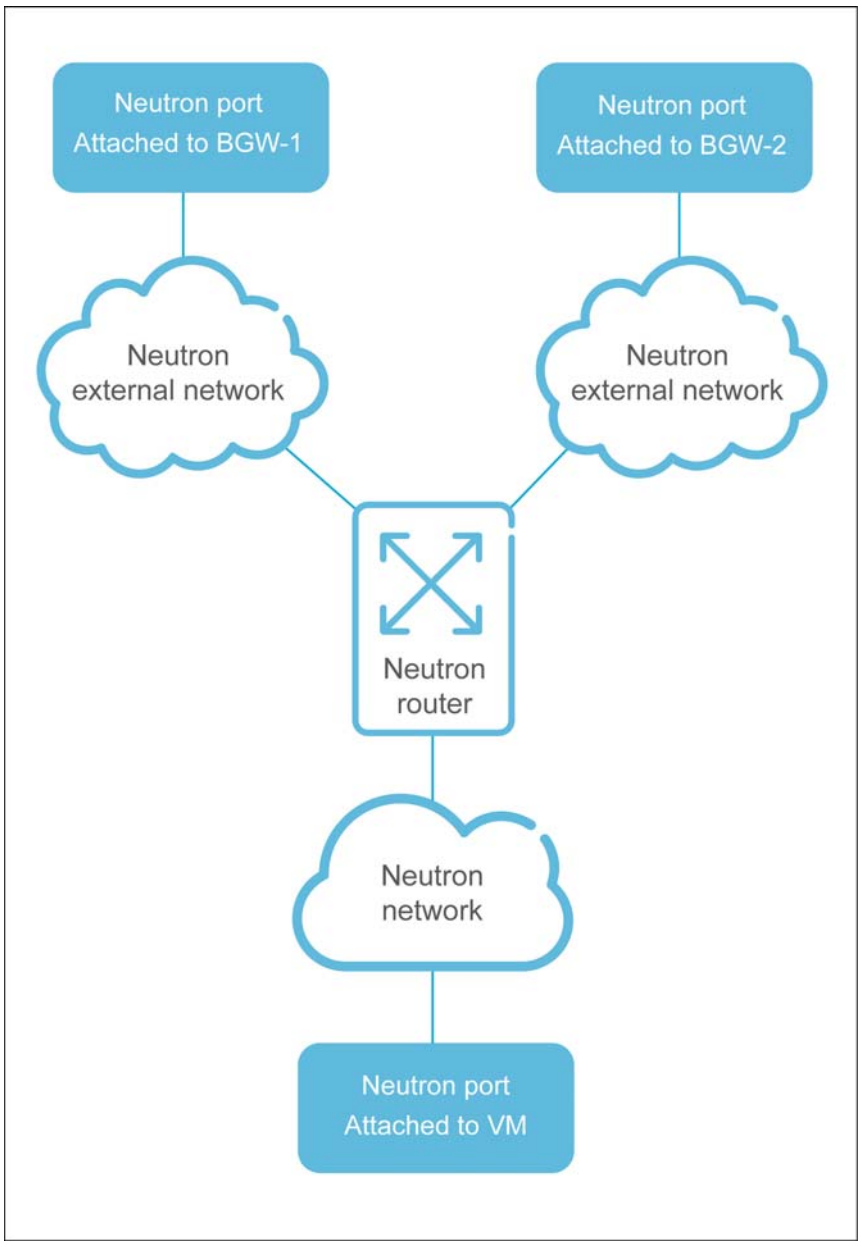

Figure 4 L3 BGW Connection

7.8.1 Create Neutron External Networks

1. Create two Neutron networks using provider network extension.

   For the procedure of creating a Neutron network, refer to the section "Networking API v2.0" in the *OpenStack API Complete Reference*.

2. Provide two segmentation IDs (one per network) and `router:external` true. These networks are used to pass the traffic to the BGWs. the Cloud

manager manages these IDs. The IDs are picked from the BGW pool. See Section 2.2 on page 2.

For the `provider` Extended Attributes for Networks, refer to the section "Networking API v2.0 extensions" in the *OpenStack API Complete Reference*.

For the concepts for info on router external, refer to the sections "Networking API v2.0" and "Networking API v2.0 extensions" in the *OpenStack API Complete Reference*.

NET A:

```
provider:segmentation_id     <VID-A>
provider:network_type        vlan
provider:physical_network    default
router:external              true
```

NET B:

```
provider:segmentation_id     <VID-B>
provider:network_type        vlan
provider:physical_network    default
router:external              true
```

7.8.2 Create Neutron Subnets

Create two subnets, one on each of the previously created networks. The `gateway_ip` on the subnets must be set to the BGW interface address. The address is used as the destination address of an automatically created default route in the Neutron router.

For the procedure of creating a Neutron subnet on Neutron network, refer to the sections "Networking API v2.0" and "Networking API v2.0 extensions" in the *OpenStack API Complete Reference*.

BGW-1:

```
network_id            <BGW-1 Net id>
gateway_ip            <BGW-1 interface address>
```

BGW-2:

```
network_id            <BGW-2 Net id>
gateway_ip            <BGW-2 interface address>
```

7.8.3 Create Neutron Ports

1. Create two “BGW” Neutron ports, one on each Neutron network.

   For the procedure of creating Neutron ports on a Neutron network, refer to the sections “Networking API v2.0” and “Networking API v2.0 extensions” in the *OpenStack API Complete Reference*.

IP address must be set to the BGW interface address.

2. Bind one Neutron port to BGW-1 and the other Neutron port to BGW-2.

   For the procedure of binding, refer to the sections “Networking API v2.0” and “Networking API v2.0 extensions” in the *OpenStack API Complete Reference*.

BGW-1:

```
fixed_ips           {
   ip_address <BGW-1 interface address>
   subnet_id  <BGW-1 subnet_id> }
device_owner        baremetal:BGW-1
network_id          <BGW-1 Net id>
binding:host_id     BGW-1
```

BGW-2:

```
fixed_ips           {
   ip_address <BGW-2 interface address>
   subnet_id  <BGW-2 subnet_id> }
device_owner        baremetal:BGW-2
network_id          <BGW-2 Net id>
binding:host_id     BGW-2
```

7.8.4 Configure BGW

Configure BGW for the segmentation IDs on the external networks.

For the configuration of BGW for the IDs on the external networks, see Figure 5.

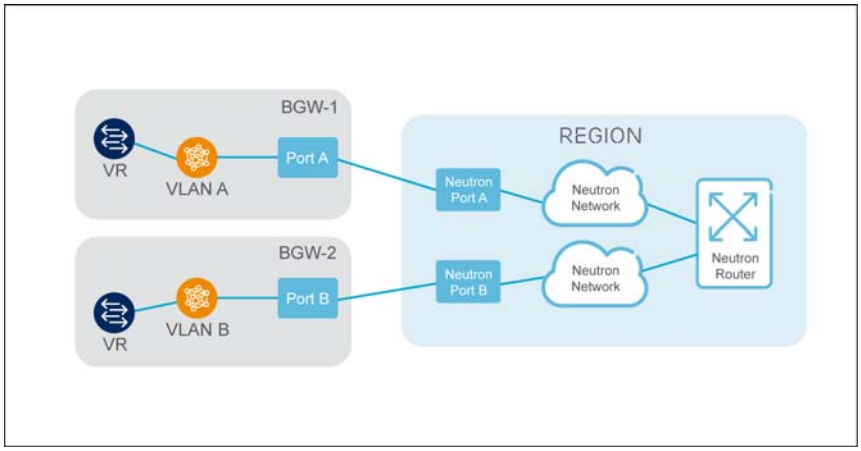

Figure 5 L3 BGW Configuration

7.8.4.1 Create a Virtual Router

Create VRs in the BGWs.

7.8.4.2 Configure Virtual Router

Configure the VR with its applicable parameters.

Note: Only IPv4 is applicable.

7.8.4.3 Create VLANs in the Virtual Router

After a VR is created and support for any required routing protocols is added, you can create a VLAN in a VR. In order to achieve redundancy there are two VLANs, one for BGW-1 and one for BGW-2.

VLAN names must reflect what they are used for.

VLAN IDs must be unique within the same broadcast domain.

7.8.4.4 Add Ports to Virtual Router

Add the applicable ports to the VR.

7.8.5 Create Neutron Router

For the procedure of creating a router, refer to the section ‘‘Networking API v2.0 extensions’’ in the *OpenStack API Complete Reference*.

Note: Do not use the `external_gatewayinfo` attribute, since external connections are handled in a different way in CEE. See Section 7.8.6 on page 48.

7.8.6 Connect Router to External Networks

Connect router to external networks. This is done by adding the previously created subnets to the router. Routing between added subnets will be done. Routing between added subnets and default route will be done.

Note: The Neutron ports on the BGW networks have to be created and bound to the BGW before adding the corresponding subnets to the router.

For the procedure of adding an interface to a router, refer to the section “Networking API v2.0 extensions” in the *OpenStack API Complete Reference*.

7.8.7 Attach VM Networks

It is required as a prerequisite that the internal L3 Network used by VMs has been created. See Section 7.6 on page 41.

For the procedure of creating a Neutron network, refer to the section “Networking API v2.0” in the *OpenStack API Complete Reference*.

7.8.8 Configure Neutron Router

Routing between added subnets is set up automatically. Routing between added subnets and default route is set up automatically.