

OpenStack Networking API in CEE with SDN

Cloud Execution Environment

INTERWORK DESCRIPTION

Copyright

© Ericsson AB 2016–2018. All rights reserved. No part of this document may be reproduced in any form without the written permission of the copyright owner.

Disclaimer

The contents of this document are subject to revision without notice due to continued progress in methodology, design and manufacturing. Ericsson shall have no liability for any error or damage of any kind resulting from the use of this document.

Trademark List

All trademarks mentioned herein are the property of their respective owners. These are shown in the document Trademark Information.



Contents

1	Introduction	1
1.1	API Version	1
1.2	Document References	1
2	Prerequisites	2
2.1	OpenStack Networking Configurations in CEE	2
2.2	Segmentation IDs	2
3	Supported Operations	3
3.1	Basic OpenStack Operations	3
3.2	OpenStack Extensions	3
4	Deviations	5
4.1	Partly Supported Operations	5
4.2	Operations Not Available in CEE	6
5	Ericsson Extensions	7
5.1	Trunkport Extension	7
6	Limitations and Recommendations	15
6.1	Limitations	15
7	Concepts and Use Cases	17
7.1	General Terms	17
7.2	IP Address Management	17
7.3	MAC Address Management	19
7.4	Internal Neutron Network without IP Address Management	20
7.5	Internal Neutron Network with Neutron IP Address Management	20
7.6	Internal L3 Connectivity Using Neutron Router	21
7.7	L2 Connectivity Using SR-IOV	21
7.8	External Connectivity Guidelines for Deployments with SDN	23
	Reference List	28





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.

It describes L2 and L3 guidelines for the case where Neutron is configured to use an ML2 mechanism driver and an L3 service plug-in for the tightly integrated SDN solution in Ericsson Hyperscale Datacenter System (HDS) deployments.

1.1 API Version

The CEE networking API is based on OpenStack Networking API v2.0.

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 [OpenStack Networking API v2.0](#).

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

This section lists the prerequisites.

2.1 OpenStack Networking Configurations in CEE

CEE networking can be configured during the installation of the CEE. For details about the configurations, refer to the [Configuration File Guide](#).

Neutron in CEE includes the following features and elements, compared to standard OpenStack Neutron:

- Cloud Software Defined Networking (SDN) Controller based on Open Daylight (ODL)
- Extensions described in Section 3.2 on page 3

2.2 Segmentation IDs

Neutron uses a range of segmentation IDs for internal network separation. In SDN deployments VXLAN is used for network separation, and the segmentation IDs are mapped to VXLAN VNIs. The range must be defined during the installation of CEE.

Table 1 Segmentation Allocation Example

Name	Range	Description
Default	100-10000	Used by Neutron for network separation

Note: The above 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 for several Neutron networks. Segmentation IDs on Neutron networks are static and cannot be changed after the creation of the Neutron network.



3 Supported Operations

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

3.1 Basic OpenStack Operations

For the detailed description of basic networking API operations, refer to the [OpenStack Networking API v2.0](#).

3.1.1 Limitations

For the CEE-specific limitations and recommendations, see [Section 6](#) on page 15.

3.2 OpenStack Extensions

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

The OpenStack BGPVPN extension is not available. The Cloud SDN Controller (CSC) instead provides the BGPVPN API outside Neutron. For more information, refer to the SDN documentation.

- List extensions

Note: Supported, but the reply contains unsupported operations.

- Quotas extension (quotas)

- Trunk Port extension

- ExtraRoute extension

- Layer 2 Gateway (L2-GW)

- Provider extended attributes

- Multiple provider extension

- Port binding extended attributes

- Routers

- Security groups

- Allowed address pairs

- L3 configurable external gateway mode



— External network



4 Deviations

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

4.1 Partly Supported Operations

The following operations are partly supported in CEE networking:

Network provider extended attributes (networks)

For non SR-IOV traffic, only the vxlan network type is supported. For SR-IOV interfaces, the flat and vlan network types are supported, the option is applicable per region.

Administrative State Down (ports)

Administration state must be up for trunkports and their subports.

If the administration state of a trunkport is set to down at the vCIC, it will not take effect to the trunkport and its subports at the relevant compute node, although Neutron will show that the trunkport and its subports are administratively down.

Security Groups

The default security group and default security group rules are not recommended. The tenant must associate the Neutron ports with custom security groups and custom security group rules. Security groups are not available for SR-IOV.

Trunkports and subports are treated like any other Neutron ports, meaning that all subports are independent of each other and could potentially be configured with different security group rules. For more details on trunkports and subports, see Section 5.1 on page 7.

Neutron port

The port_security_enabled attribute cannot be changed in runtime using a Neutron port update command. The attribute can only be configured during deployment.

Neutron routers

IPv6 is not available.



4.2 Operations Not Available in CEE

The following operations are not available in CEE networking:

- Configurable external gateway modes
- Extra DHCP options (`extra-dhcp-opt`)
- Firewall as a Service (FaaS)
- Load Balancer as a Service (LBaaS)
- Metering labels and rules
- Virtual Private Network as a Service (VPNaaS)



5 Ericsson Extensions

This section describes the added CEE API extensions in detail.

5.1 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.

Note: The trunk port API is deprecated in this release. This API is fully supported in this release, but will be discontinued in the following CEE releases, and it is planned to be replaced by the OpenStack “VLAN aware VMs” API.

5.1.1 Topology

A trunkport can be a simple trunkport or a subport. The subports are associated to the simple trunkport.

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.

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

Note: Simple trunkports do not forward untagged traffic in CEE deployed with SDN. Use a normal Neutron port when untagged traffic is needed.

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 VLAN tag from the VM (the one associated with the subport) is stripped before entering the associated Neutron network. When entering the associated VXLAN network, 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 VXLAN overhead is stripped of and the VLAN tag associated with the subport is added before the traffic is forwarded to a VM.

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

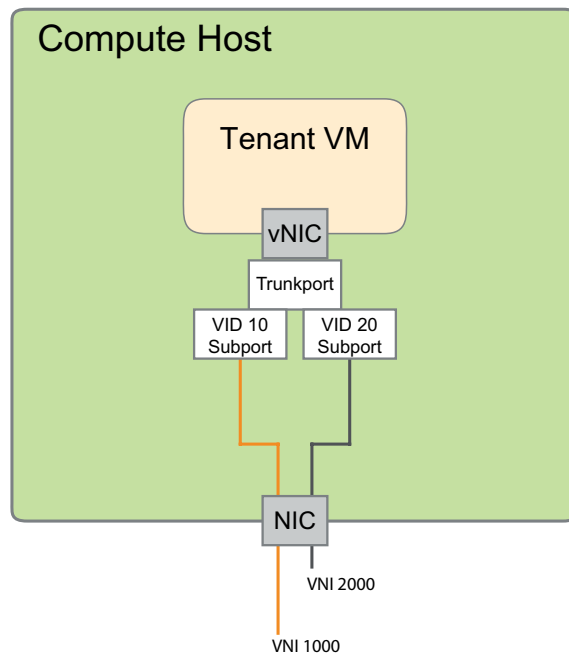


Figure 1 Trunkport Topology

5.1.2 Concepts

The trunkport-prefixed extended attributes for the **port** resource are shown in Table 2.

Table 2 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-string	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.



Table 2 Trunkport Attributes

Attribute	Type	Required	CRUD ⁽¹⁾	Default Value	Validation Constraints	Notes
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.

5.1.3 Trunkport API Operations

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

Note: Create and assign a dummy Neutron network when creating a trunkport. See Section 6.1.1 on page 15 for more information.

5.1.3.1 List Ports

Table 3 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 1 List Ports: JSON Response



5.1.3.2 Show Port

Table 4 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 2 Show Port with Trunkport Attributes: JSON Response



5.1.3.3 Create Port

Table 5 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 3 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",
: "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 4 Create Port with Trunkport Attributes: JSON Response

```
{
  "port":
  {
    "network_id": "87733bcc-8144-41b1-bb6b-d011d7a5168e",
    "fixed_ips": [{"subnet_id": "99a8aea3-b9da-409d-a5e5-f45338ceb4d3",
: "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 5 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 6 Create Port with Trunkport Attributes (Type Subport): JSON Response

5.1.3.4 Update Port

No attributes can be updated.

5.1.3.5 Delete Port

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



6 Limitations and Recommendations

This section lists the limitations and recommendations for CEE networking.

Note: For a list of CSC-specific limitations, refer to the SDN documentation.

6.1 Limitations

This section contains the limitations of CEE networking.

6.1.1 Creating Trunkports

Subports are created by assigning them to a simple trunkport. In CEE deployed with SDN, simple trunkports cannot be used for traffic, but as with normal Neutron ports, the `port-create` API operation fails if no Neutron network is assigned to the trunkport. Thus a Neutron network has to be assigned to the simple trunkport even though no traffic will be carried by the simple trunkport.

To create a dummy network and create the trunkport and subport, follow these steps:

1. Create the dummy Neutron network.
2. Create the trunkport.
3. Create the subport.

6.1.2 Deleting Trunkport and Subports

If a trunkport is deleted before 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 trunkport and subports are created with the same IP address, this can prevent VMs on the ports to get an IP address.

To delete a trunkport with subports, do the following:

1. Delete the related subports.
2. Delete the trunkport.

6.1.3 Updating `port_security_enabled` Flag

If the Neutron port parameter `port_security_enabled` is modified with `neutron port-update` command after port creation, the effect does not cascade down to the port.

To modify the value of the flag and expect the behavior change, do the following:

1. Delete the port.



2. Recreate the port with the required `port_security_enabled` flag.

6.1.4 VLAN Tagging Is Not Removed for VFs Belonging to a Deleted Port or Network

Neutron does not remove the VLAN tagging from a VF belonging to an already deleted port or network if the VF interface is not in use. However, the VF is free for other use, and once assigned to a VM, the VLAN ID and the MAC address will change accordingly.

6.1.5 VFs Do Not Obtain the IP Addresses Assigned to Them by the Neutron DHCP Service

The provisioned VMs which have SR-IOV VF interfaces with `flat` network type cannot use the Neutron DHCP service. Therefore, they must be configured manually in the guest VM.

6.1.6 Limited IPv6 Support

- IPv6 subnets are supported.
- IP address management with manual IPv6 address configuration or automatic IPv6 assignment (SLAAC) is supported.
- Stateless and Stateful DHCPv6 are not available.
- IPv6 routing and forwarding are not available: IPv6 Neutron routers can be created, but the usage is limited to SLAAC.

6.1.7 Security Group Association After Port Creation

Assigning or removing security groups of a Neutron port using the `neutron port-update` command is not available.

Neutron ports have to be associated with a security group during port creation. Updating the rules of the security group assigned to the port is possible.



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 Network

L2 broadcast domain

Neutron Subnet

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

VNI

VXLAN Network Identifier (or VXLAN Segment ID). A 24-bit segment ID that identifies and isolates VXLAN segments.

VXLAN

Network virtualization technique that uses MAC-in-UDP encapsulation by adding a UDP header and a VXLAN header before an original Ethernet packet.

SLAAC

Stateless Address Autoconfiguration

7.2 IP Address Management

For IPv4 address management, see Section 7.2.1 on page 17.

For IPv6 address management, see Section 7.2.2 on page 18.

7.2.1 IPv4 Address Management

The following types of IPv4 address management are supported:

Address Management Outside Neutron

The application handles the IP addresses on its own and does not involve Neutron.

Note: When port security is enabled, allowed address pairs and security groups must be configured to allow access for application managed IP addresses (IPv4 or IPv6). For more information on allowed address pairs and security groups, refer to [OpenStack Networking API v2.0](#).

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.



Address Management in Neutron

Application uses Neutron for IP address management.

This can be accomplished in several ways:

- The application can specify IP addresses per ports.
- The application can specify the IP subnets only and let Neutron allocate the addresses.

Note: A third option is 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`. Refer to the [OpenStack Networking API v2.0](#).

7.2.1.1

DHCP

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>`

7.2.2

IPv6 Address Management

The following types of IPv6 address management are supported:

Address Management outside Neutron

The application handles the IP addresses on its own and does not involve Neutron. No Neutron SLAAC or DHCP service is used. Only L2 services are available from Neutron.

Note: When port security is enabled, allowed address pairs and security groups must be configured to allow access for application managed IP addresses (IPv4 or IPv6). For more information on allowed address pairs and security groups, refer to [OpenStack Networking API v2.0](#)

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.

Manual Address Configuration

Create IPv6 subnets with the below attributes set:

- `enable_dhcp = false`
- `ipv6_address_mode= N/S`
- `ipv6_ra_mode = N/S`

An IPv6 address can be assigned per Neutron port, or Neutron can assign IPv6 addresses from the IPv6 subnets.

SLAAC

When using SLAAC, the router sending Router Advertisements (RAs) can either be a Neutron router or an external router.

In case of using a Neutron router, create IPv6 subnets with the below attributes, depending on whether a Neutron router or an external router (DC-GW) is used:

Attribute	Neutron router	External router
<code>enable_dhcp</code>	<code>true</code>	<code>true</code>
<code>ipv6_address_mode</code>	<code>slaac</code>	<code>slaac</code>
<code>ipv6_ra_mode</code>	<code>slaac</code>	<code>N/S</code>

Note: The Neutron router cannot be used for routing and forwarding between IPv6 subnets. It can only be used for SLAAC.

Once the VM is booted, it automatically retrieves IPv6 prefix(es) as well as MTU using SLAAC and creates global IPv6 address(es).

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 [OpenStack Networking API v2.0](#).

Neutron-allocated MAC addresses have a three byte fixed prefix. The rest will be a random number unique per Neutron network. Refer to the [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 Internal Neutron Network without IP Address Management

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 [OpenStack Networking API v2.0](#).

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 [OpenStack Networking API v2.0](#).

3. Create Neutron ports on Neutron network.

For the procedure of creating Neutron ports on a Neutron network, refer to the [OpenStack Networking API v2.0](#).

4. Create VMs using Neutron ports.

7.5 Internal Neutron Network with Neutron IP Address Management

This section describes how to connect VMs using Neutron DHCP IP address management.

1. Create Neutron network

For the procedure of creating a Neutron network, refer to the [OpenStack Networking API v2.0](#).

2. Create Neutron subnet on Neutron network

For the procedure of creating a Neutron subnet on Neutron network, refer to the [OpenStack Networking API v2.0](#).



3. Create Neutron ports on Neutron network

For the procedure of creating Neutron ports on a Neutron network, refer to the [OpenStack Networking API v2.0](#).

4. Create VMs using Neutron ports.

7.6 Internal L3 Connectivity Using Neutron Router

This section describes how to connect VMs using Neutron router.

1. Create two or more Neutron network

For the procedure of creating a Neutron network, refer to the [OpenStack Networking API v2.0](#).

2. Create Neutron subnets on the Neutron networks

For the procedure of creating a Neutron subnet on Neutron network, refer to the [OpenStack Networking API v2.0](#).

3. Create a Neutron router

For the procedure of creating a Neutron router, refer to the [OpenStack Networking API v2.0](#).

4. Create ports on the Neutron networks

For the Neutron router, creating ports is optional.

For the procedure of creating Neutron ports on a Neutron network, refer to the [OpenStack Networking API v2.0](#).

5. Add network interfaces to the Neutron router

Interfaces can be added either by specifying Neutron subnets or Neutron ports.

For the procedure of adding interfaces to a Neutron router, refer to the [OpenStack Networking API v2.0](#).

6. Create VMs using Neutron ports on the Neutron networks which subnets are connected to the Neutron router.

7.7 L2 Connectivity Using SR-IOV

Note: SR-IOV configuration must be added to the `config.yaml` file before CEE deployment. Refer to the [Configuration File Guide](#) for more details.

This section describes how to connect VMs to DC-GW with SR-IOV interfaces providing external L2 (VLAN) connectivity.



Type vlan SR-IOV Neutron networks are configured using the multi-provider Neutron extension for multi-segment networks. Switch side utilization is controlled by the physical networks configured for SR-IOV in `config.yaml`.

1. Create L2-GWs representing the switch clusters with ports used for SR-IOV connectivity and DC-GW connectivity respectively.

Note: When HDS L2 switch fabric is utilized, the DC-GW is connected to the spine switch cluster. If HDS L3 switch fabric is used, the DC-GW is connected to a leaf switch cluster.

The transition between the VXLAN connected to the VMs and the VLAN connected to the DC-GW is handled by a L2-GW. The L2-GW connects a Neutron network to VLANs on physical switch ports. The switch cluster and port names can be retrieved from HDS using CCM.

```
neutron l2-gateway-create <l2_gateway_name> =>
--device name=<switch_name>,interface_names=>
"<port_name1>;<port_name_2>"
```

2. Create a multi-segment Neutron network with one VXLAN and one VLAN segment.

The VLAN segment must have the physical network set to the corresponding physical network configured for SR-IOV connectivity in `config.yaml` at deployment time. For SR-IOV connectivity, the segmentation ID must be set to the VLAN ID.

For the procedure of creating a multi-segment Neutron network, refer to the [OpenStack Networking API v2.0](#).

3. Create one or two L2-GW connections using the multi-segment network created in Step 2, and the VLAN used for SR-IOV connectivity.

Note: When HDS L2 switch fabric is utilized, the DC-GW is connected to the spine switch cluster. If HDS L3 switch fabric is used, the DC-GW is connected to a leaf switch cluster.

- a. For internal connectivity only, create an L2-GW connection in the L2-GW(s) representing the switch cluster(s) used for SR-IOV connectivity:

```
neutron l2-gateway-connection-create =>
<switch_name> <neutron_network> =>
segmentation_id <vid>
```

- b. When external DC-GW connectivity is required, a connection must also be created in the L2-GW representing the switch cluster connected to the DC-GW:

```
neutron l2-gateway-connection-create =>
<switch_name> <neutron_network> =>
```



`segmentation_id <vid>`

4. Create Neutron subnet on the 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 more information, refer to the [OpenStack Networking API v2.0](#).

5. Create a Neutron port for SR-IOV with `vnic-type` set to `direct` on the Neutron network created in Step 2.

For more information on creating Neutron ports on a Neutron network, refer to the [OpenStack Networking API v2.0](#).

6. Create VMs using Neutron ports.

7.8 External Connectivity Guidelines for Deployments with SDN

In HDS based deployments, CEE operates as an HDS tenant in its own Virtualized Performance Optimized Datacenter (vPOD). HDS uses hardware VXLAN Tunnel End Points (VTEP) in spine switches to provide L2 external network connectivity for HDS tenants, as shown in Figure 2. For external L3 connectivity, BGP L3VPN is used directly from the SW VTEPs. For CEE deployed on HDS with an L3 fabric, HDS uses VXLAN TEPs in the leaf switches to provide external network connectivity. The hardware VTEP must be configured by Neutron using an L2 Gateway (L2-GW) API.

For more information on L2 connectivity using HW VTEPs provided by HDS, refer to “External L2 Gateways” in the HDS documentation.

Note: The terms Neutron L2GW and HDS L2GW are not interchangeable, as they refer to different concepts.

For more information on DC-GW configuration, refer to “Data Network Switch Fabric to DC-GW Connections” in the HDS documentation.

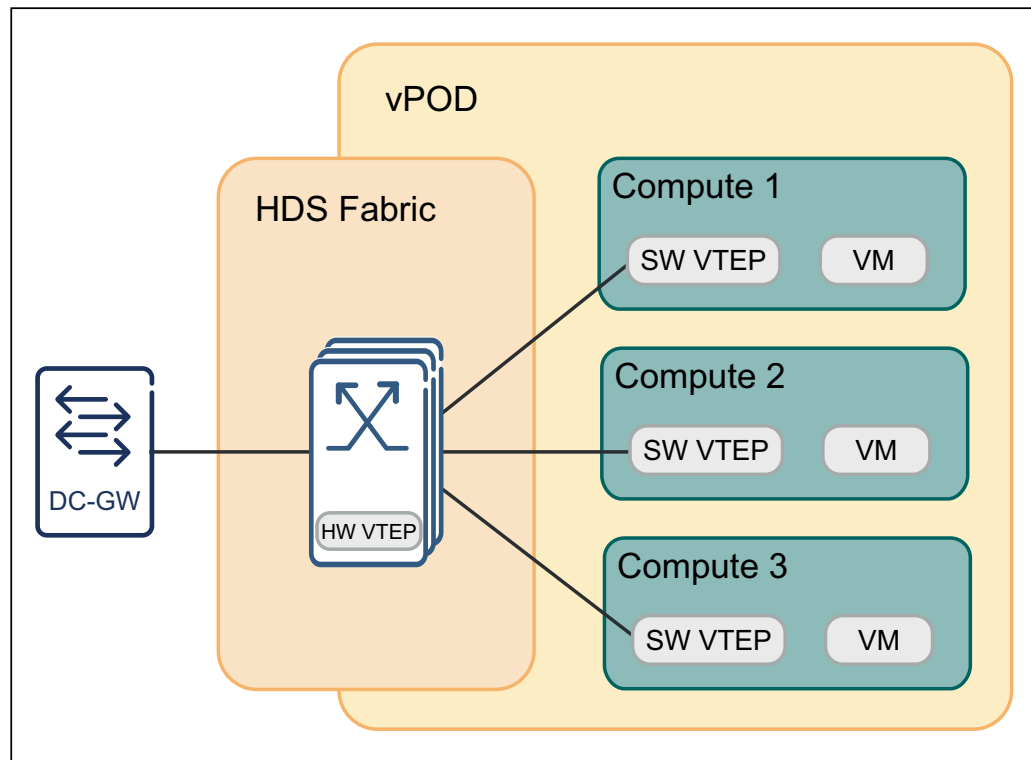


Figure 2 HDS External Tenant Connectivity

7.8.1 L2 Connectivity to DC-GW

This section describes how to connect VMs to DC-GW using an L2 network.

1. Configure physical network.

In cases when there are no physical networks configured for SR-IOV usage, specific physical network segments between the switches and DC-GW must be configured at CEE installation. Configuration is achieved using the `l2gw` Fuel plugin section of `config.yaml`. Refer to the [Configuration File Guide](#) for more details. When physical networks are configured for SR-IOV, they can also be used for connectivity to DC-GW.

2. Create a L2-GW representing the switch cluster with ports connected to DC-GW.

Note: When HDS L2 switch fabric is utilized, the DC-GW is connected to the spine switch cluster. If HDS L3 switch fabric is used, the DC-GW is connected to a leaf switch cluster.

The transition between the VXLAN connected to the VMs and the VLAN connected to the DC-GW is handled by an L2-GW. The L2-GW connects a Neutron network to VLANs on physical switch ports. Use CCM to retrieve the switch cluster and port names from HDS.



```
neutron l2-gateway-create <l2_gateway_name> =>
--device name=<switch_name>,interface_names=>
"<port_name1>;<port_name_2>"
```

3. Create a multi-segment Neutron network with one VXLAN and one VLAN segment.

The VLAN segment must have the physical network set to the corresponding physical network configured at deployment time, as described in Step 1. The segmentation id set to the VLAN id to be used for communication with the DC-GW. For communication with the DC-GW, the segmentation ID must correspond with the VLAN ID.

For the procedure of creating a multi-segment Neutron network, refer to the [OpenStack Networking API v2.0](#).

4. Create a L2-GW connection using the multi-segment network created in Step 3 and the VLAN used for communication with the DC-GW.

```
neutron l2-gateway-connection-create =>
<l2_gateway_name> <neutron_network> =>
--default-segmentation-id <vid>
```

5. 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 [OpenStack Networking API v2.0](#).

6. Create Neutron ports on the Neutron network.

For the procedure of creating Neutron ports on a Neutron network, refer to the [OpenStack Networking API v2.0](#).

7. Configure DC-GW

The traffic is configured to reach the DC-GW. The DC-GW must be configured to handle this traffic.

For the configuration of DC-GW for this segmentation ID, see Figure 3.

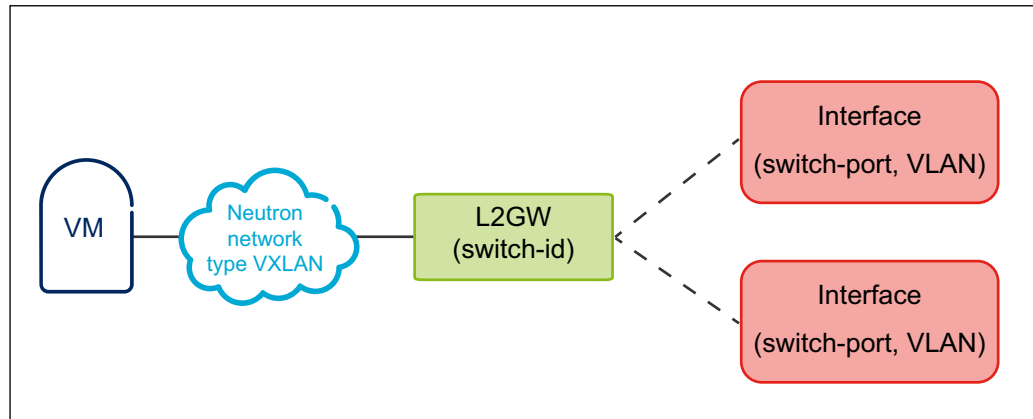


Figure 3 L2 DC-GW Configuration

8. Create VMs using Neutron ports.

7.8.2

L3 Connectivity to DC-GW

VMs can connect to DC-GW using BGP L3VPN over Multiprotocol Label Switching (MPLS) and Generic Routing Encapsulation (GRE). The user, Ericsson Cloud Manager (ECM), configures BGP L3VPN using CSC REST API directly; OpenStack Networking (Neutron) APIs are not used in this case.

Note: For the configuration of the GRE peer, check the `config.yaml`. The `remote_gre_term: <REMOTE.GRE.TERM.IP>` points to GRE endpoints on DC-GW. These endpoints must be defined on DC-GW.

To establish a L3 connectivity to DC-GW, follow these steps:

1. Create a Neutron provider network with `provider:network_type gre` and `provider:segmentation_id` set to the GRE key.

For the procedure of creating a Neutron network, refer to the [OpenStack Networking API v2.0](#).

2. Create Neutron subnet on the Neutron network.

For the procedure of creating a Neutron subnet on Neutron network, refer to the [OpenStack Networking API v2.0](#).

3. Create a Neutron router.

For the procedure of creating a Neutron router, refer to the [OpenStack Networking API v2.0](#).

4. Create ports on the Neutron network.

For the Neutron router, creating ports is optional.

For the procedure of creating Neutron ports on a Neutron network, refer to the [OpenStack Networking API v2.0](#).



5. Create a BGP L3VPN with a Route Distinguisher (RD), import Route Target (iRT), export Route Target (eRT) and Tenant ID as attributes, using the CSC REST API.

For the procedure of creating a BGP L3VPN, refer to the SDN document, RESTCONF Interfaces, Reference [1].

6. Add network interfaces to the Neutron router.

Interfaces can be added either by specifying Neutron subnets or Neutron ports.

For the procedure of adding interfaces to a Neutron router, refer to the [OpenStack Networking API v2.0](#).

7. Associate L3VPN using the CSC REST API.

The association can be done in two ways:

- a. Associate the Neutron router with the BGP L3VPN instance. All subnets of the router are associated with the BGP L3VPN.
- b. Associate the network(s) with BGP L3VPN instance. The corresponding subnet is automatically associated with the BGP L3VPN instance.

For the procedure of association of BGP L3VPN, refer to the SDN document, RESTCONF Interfaces, Reference [1].



Reference List

- [1] RESTCONF Interfaces, 2/155 19-HSD 101 048/3 Uen B