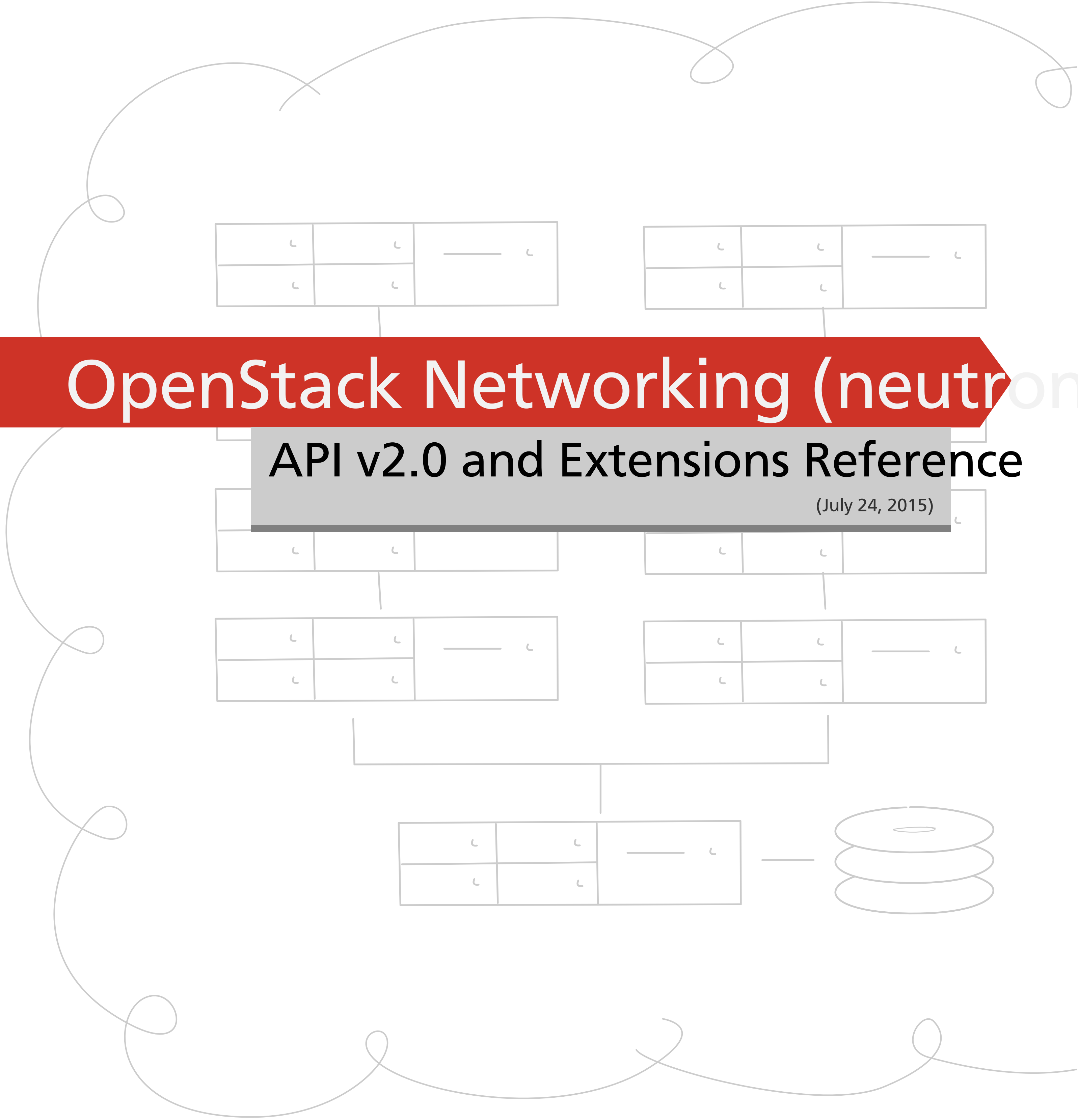

OpenStack Networking (neutron

API v2.0 and Extensions Reference

(July 24, 2015)

openstack™

docs.openstack.org

OpenStack Networking (neutron) API v2.0 and Extensions Reference

Table of Contents

1. Networking API v2.0 (CURRENT)

Use virtual networking services among devices that are managed by the OpenStack Compute service. The Networking (neutron) API v2.0 combines the API v1.1 functionality with some essential Internet Protocol Address Management (IPAM) functionality. Enables users to associate IP address blocks and other network configuration settings with an OpenStack Networking network. You can choose a specific IP address from the block or let OpenStack Networking choose the first available IP address.

Method	URI	Description
	API versions	
GET	`/`	Lists information about all Networking API versions.
GET	`/v2.0`	Shows details for Networking API v2.0.
	Networks	
POST	`/v2.0/networks`	Creates a network.
POST	`/v2.0/networks`	Creates multiple networks in a single request.
GET	`/v2.0/networks/{network_id}`	Shows information for a specified network.
PUT	`/v2.0/networks/{network_id}`	Updates a specified network.
DELETE	`/v2.0/networks/{network_id}`	Deletes a specified network and its associated resources.
	Subnets	
GET	`/v2.0/subnets{?display_name, network_id,gateway_ip,ip_version, cidr,id,enable_dhcp,ipv6_ra_mode, ipv6_address_mode}`	Lists subnets to which the specified tenant has access.
POST	`/v2.0/subnets`	Creates a subnet on a specified network.
POST	`/v2.0/subnets`	Creates multiple subnets in a single request. Specify a list of subnets in the request body.
GET	`/v2.0/subnets/{subnet_id}`	Shows information for a specified subnet.
PUT	`/v2.0/subnets/{subnet_id}`	Updates a specified subnet.
DELETE	`/v2.0/subnets/{subnet_id}`	Deletes a specified subnet.
	Ports	
GET	`/v2.0/ports{?status,display_name, admin_state,network_id, device_owner,mac_address,port_id, security_groups,device_id}`	Lists ports to which the tenant has access.
POST	`/v2.0/ports`	Creates a port on a specified network.
POST	`/v2.0/ports`	Creates multiple ports in a single request. Specify a list of ports in the request body.
GET	`/v2.0/ports/{port_id}`	Shows information for a specified port.
PUT	`/v2.0/ports/{port_id}`	Updates a specified port.
DELETE	`/v2.0/ports/{port_id}`	Deletes a specified port.

1.1. API versions

Lists information for all Networking API versions and shows details about API v2.

Method	URI	Description
GET	`/`	Lists information about all Networking API versions.
GET	`/v2.0`	Shows details for Networking API v2.0.

1.1.1. List API versions

Method	URI	Description
GET	/	Lists information about all Networking API versions.

Normal response codes: 200, 300

1.1.1.1. Request

This operation does not accept a request body.

1.1.1.2. Response

Example 1.1. List API versions: JSON response

```
{
    "versions": [
        {
            "status": "CURRENT",
            "id": "v2.0",
            "links": [
                {
                    "href": "http://23.253.228.211:9696/v2.0",
                    "rel": "self"
                }
            ]
        }
    ]
}
```

Example 1.2. List API versions: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<versions xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <version>
        <status>CURRENT</status>
        <id>v2.0</id>
        <links>
            <link>
                <href>http://23.253.228.211:9696/v2.0</href>
                <rel>self</rel>
            </link>
        </links>
    </version>
</versions>
```

This operation does not return a response body.

1.1.2. Show API v2.0 details

Method	URI	Description
GET	`/v2.0`	Shows details for Networking API v2.0.

Normal response codes: 200, 203

1.1.2.1. Request

This operation does not accept a request body.

1.1.2.2. Response

Example 1.3. Show API v2.0 details: JSON response

```
{
    "resources": [
        {
            "links": [
                {
                    "href": "http://23.253.228.211:9696/v2.0/subnets",
                    "rel": "self"
                }
            ],
            "name": "subnet",
            "collection": "subnets"
        },
        {
            "links": [
                {
                    "href": "http://23.253.228.211:9696/v2.0/networks",
                    "rel": "self"
                }
            ],
            "name": "network",
            "collection": "networks"
        },
        {
            "links": [
                {
                    "href": "http://23.253.228.211:9696/v2.0/ports",
                    "rel": "self"
                }
            ],
            "name": "port",
            "collection": "ports"
        }
    ]
}
```

This table shows the body parameters for the show api v2.0 details response:

Name	Type	Description
`location`	AnyURI *(Required)*	Full URL to a service or server.

Example 1.4. Show API v2.0 details: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<resources xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <resource>
        <links>
            <link>
                <href>http://23.253.228.211:9696/v2.0/subnets</href>
                <rel>self</rel>
            </link>
        </links>
        <name>subnet</name>
        <collection>subnets</collection>
    </resource>
    <resource>
        <links>
            <link>
                <href>http://23.253.228.211:9696/v2.0/networks</href>
                <rel>self</rel>
            </link>
        </links>
        <name>network</name>
        <collection>networks</collection>
    </resource>
    <resource>
        <links>
            <link>
                <href>http://23.253.228.211:9696/v2.0/ports</href>
                <rel>self</rel>
            </link>
        </links>
        <name>port</name>
        <collection>ports</collection>
    </resource>
</resources>
```

This operation does not return a response body.

1.2. Networks

Lists, shows information for, creates, updates, and deletes networks.

Method	URI	Description
POST	`/v2.0/networks`	Creates a network.
POST	`/v2.0/networks`	Creates multiple networks in a single request.
GET	`/v2.0/networks/{network_id}`	Shows information for a specified network.
PUT	`/v2.0/networks/{network_id}`	Updates a specified network.
DELETE	`/v2.0/networks/{network_id}`	Deletes a specified network and its associated resources.

1.2.1. Create network

Method	URI	Description
POST	`/v2.0/networks`	Creates a network.

A request body is optional. An administrative user can specify another tenant ID, which is the tenant who owns the network, in the request body.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

1.2.1.1. Request

Example 1.5. Create network: JSON request

```
{
    "network": {
        "name": "sample_network",
        "admin_state_up": true
    }
}
```

Example 1.6. Create network: XML request

```
<?xml version="1.0" encoding="UTF-8"?>
<network>
    <name>sample_network2</name>
</network>
```

1.2.1.2. Response

Example 1.7. Create network: JSON response

```
{
    "network": {
        "status": "ACTIVE",
        "subnets": [],
        "name": "net1",
        "admin_state_up": true,
        "tenant_id": "9bacb3c5d39d41a79512987f338cf177",
        "router:external": false,
        "segments": [
            {
                "provider:segmentation_id": 2,
                "provider:physical_network": "8bab8453-1bc9-45af-8c70-
f83aa9b50453",
                "provider:network_type": "vlan"
            },
            {
                "provider:segmentation_id": null,
                "provider:physical_network": "8bab8453-1bc9-45af-8c70-
f83aa9b50453",
                "provider:network_type": "stt"
            }
        ],
        "shared": false,
        "id": "4e8e5957-649f-477b-9e5b-f1f75b21c03c"
    }
}
```

Example 1.8. Create network: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<network xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:provider="http://docs.openstack.org/ext/provider/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <status>ACTIVE</status>
    <subnets quantum:type="list"/>
    <name>sample_network2</name>
    <provider:physical_network xsi:nil="true"/>
    <admin_state_up quantum:type="bool">True</admin_state_up>
    <tenant_id>4fd44f30292945e481c7b8a0c8908869</tenant_id>
    <provider:network_type>local</provider:network_type>
    <shared quantum:type="bool">False</shared>
    <id>c220b026-ece1-4ead-873f-83537f4c9f92</id>
    <provider:segmentation_id xsi:nil="true"/>
</network>
```

1.2.2. Bulk create networks

Method	URI	Description
POST	`/v2.0/networks`	Creates multiple networks in a single request.

In the request body, specify a list of networks.

The bulk create operation is always atomic. Either all or no networks in the request body are created.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

1.2.2.1. Request

Example 1.9. Bulk create networks: JSON request

```
{
    "networks": [
        {
            "name": "sample_network3",
            "admin_state_up": true
        },
        {
            "name": "sample_network4",
            "admin_state_up": true
        }
    ]
}
```

Example 1.10. Bulk create networks: XML request

```
<?xml version="1.0" encoding="UTF-8"?>
<networks>
    <network>
        <name>sample_network_5</name>
    </network>
    <network>
        <name>sample_network_6</name>
    </network>
</networks>
```

1.2.2.2. Response

Example 1.11. Bulk create networks: JSON response

```
{
    "networks": [
        {
            "status": "ACTIVE",
            "subnets": [],
            "name": "sample_network3",
            "provider:physical_network": null,
            "admin_state_up": true,
            "tenant_id": "4fd44f30292945e481c7b8a0c8908869",
            "provider:network_type": "local",
            "shared": false,
            "id": "bc1a76cb-8767-4c3a-bb95-018b822f2130",
            "provider:segmentation_id": null
        },
        {
            "status": "ACTIVE",
            "subnets": [],
            "name": "sample_network4",
            "provider:physical_network": null,
            "admin_state_up": true,
            "tenant_id": "4fd44f30292945e481c7b8a0c8908869",
            "provider:network_type": "local",
            "shared": false,
            "id": "af374017-c9ae-4a1d-b799-ab73111476e2",
            "provider:segmentation_id": null
        }
    ]
}
```

Example 1.12. Bulk create networks: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<networks xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:provider="http://docs.openstack.org/ext/provider/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <network>
        <status>ACTIVE</status>
        <subnets quantum:type="list"/>
        <name>sample_network_5</name>
        <provider:physical_network xsi:nil="true"/>
        <admin_state_up quantum:type="bool">True</admin_state_up>
        <tenant_id>4fd44f30292945e481c7b8a0c8908869</tenant_id>
        <provider:network_type>local</provider:network_type>
        <shared quantum:type="bool">False</shared>
        <id>1f370095-98f6-4079-be64-6d3d4a6adcc6</id>
        <provider:segmentation_id xsi:nil="true"/>
    </network>
    <network>
        <status>ACTIVE</status>
        <subnets quantum:type="list"/>
        <name>sample_network_6</name>
        <provider:physical_network xsi:nil="true"/>
        <admin_state_up quantum:type="bool">True</admin_state_up>
        <tenant_id>4fd44f30292945e481c7b8a0c8908869</tenant_id>
        <provider:network_type>local</provider:network_type>
        <shared quantum:type="bool">False</shared>
        <id>ee2d3158-3e80-4fb3-ba87-c99f515d85e7</id>
        <provider:segmentation_id xsi:nil="true"/>
    </network>
</networks>
```

1.2.3. Show network

Method	URI	Description
GET	`/v2.0/networks/{network_id}`	Shows information for a specified network.

You can control which response parameters are returned by using the fields query parameter. For information, see Filtering and column selection.

In addition to the following response parameters, the response might show extension response parameters. For information, see Networks multiple provider extension (networks).

Normal response codes: 200

Error response codes: unauthorized (401), itemNotFound (404)

1.2.3.1. Request

This table shows the URI parameters for the show network request:

Name	Type	Description
`{network_id}`	UUID	The UUID for the network of interest to you.

This operation does not accept a request body.

1.2.3.2. Response

Example 1.13. Show network: JSON response

```
{
    "network": {
        "status": "ACTIVE",
        "subnets": [
            "54d6f61d-db07-451c-9ab3-b9609b6b6f0b"
        ],
        "name": "private-network",
        "router:external": false,
        "admin_state_up": true,
        "tenant_id": "4fd44f30292945e481c7b8a0c8908869",
        "mtu": 0,
        "shared": true,
        "port_security_enabled": true,
        "id": "d32019d3-bc6e-4319-9c1d-6722fc136a22"
    }
}
```

Example 1.14. Show network: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<network xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:provider="http://docs.openstack.org/ext/provider/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:router="http://docs.openstack.org/ext/neutron/router/api/v1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <status>ACTIVE</status>
    <subnets>
        <subnet>54d6f61d-db07-451c-9ab3-b9609b6b6f0b</subnet>
    </subnets>
    <name>private-network</name>
    <admin_state_up quantum:type="bool">True</admin_state_up>
    <tenant_id>4fd44f30292945e481c7b8a0c8908869</tenant_id>
    <shared quantum:type="bool">True</shared>
    <id>d32019d3-bc6e-4319-9c1d-6722fc136a22</id>
</network>
```

1.2.4. Update network

Method	URI	Description
PUT	/v2.0/networks/{network_id}	Updates a specified network.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), forbidden (403), itemNotFound (404)

1.2.4.1. Request

This table shows the URI parameters for the update network request:

Name	Type	Description
{network_id}	UUID	The UUID for the network of interest to you.

Example 1.15. Update network: JSON request

```
{
    "network": {
        "name": "sample_network_5_updated"
    }
}
```

Example 1.16. Update network: XML request

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:provider="http://docs.openstack.org/ext/provider/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:router="http://docs.openstack.org/ext/quantum/router/api/v1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <name>sample-network-4-updated</name>
</network>
```

1.2.4.2. Response

Example 1.17. Update network: JSON response

```
{
    "network": {
        "status": "ACTIVE",
        "subnets": [],
        "name": "sample_network_5_updated",
        "provider:physical_network": null,
        "admin_state_up": true,
        "tenant_id": "4fd44f30292945e481c7b8a0c8908869",
        "provider:network_type": "local",
        "router:external": false,
        "mtu": 0,
        "shared": false,
        "port_security_enabled": true,
        "id": "1f370095-98f6-4079-be64-6d3d4a6adcc6",
        "provider:segmentation_id": null
    }
}
```

Example 1.18. Update network: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<network xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:provider="http://docs.openstack.org/ext/provider/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:router="http://docs.openstack.org/ext/neutron/router/api/v1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <status>ACTIVE</status>
    <subnets quantum:type="list"/>
    <name>sample-network-4-updated</name>
    <provider:physical_network xsi:nil="true"/>
    <admin_state_up quantum:type="bool">True</admin_state_up>
    <tenant_id>4fd44f30292945e481c7b8a0c8908869</tenant_id>
    <provider:network_type>local</provider:network_type>
    <router:external quantum:type="bool">False</router:external>
    <shared quantum:type="bool">False</shared>
    <id>af374017-c9ae-4a1d-b799-ab73111476e2</id>
    <provider:segmentation_id xsi:nil="true"/>
</network>
```

1.2.5. Delete network

Method	URI	Description
DELETE	`/v2.0/networks/{network_id}`	Deletes a specified network and its associated resources.

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404), conflict (409)

1.2.5.1. Request

This table shows the URI parameters for the delete network request:

Name	Type	Description
`{network_id}`	UUID	The UUID for the network of interest to you.

This operation does not accept a request body.

1.3. Subnets

Lists, shows information for, creates, updates, and deletes subnet resources.

Method	URI	Description
GET	`/v2.0/subnets{?display_name, network_id,gateway_ip,ip_version, cidr,id,enable_dhcp,ipv6_ra_mode, ipv6_address_mode}`	Lists subnets to which the specified tenant has access.
POST	`/v2.0/subnets`	Creates a subnet on a specified network.
POST	`/v2.0/subnets`	Creates multiple subnets in a single request. Specify a list of subnets in the request body.
GET	`/v2.0/subnets/{subnet_id}`	Shows information for a specified subnet.
PUT	`/v2.0/subnets/{subnet_id}`	Updates a specified subnet.
DELETE	`/v2.0/subnets/{subnet_id}`	Deletes a specified subnet.

1.3.1. List subnets

Method	URI	Description
GET	`/v2.0/subnets{?display_name, network_id,gateway_ip,ip_version, cidr,id,enable_dhcp,ipv6_ra_mode, ipv6_address_mode}`	Lists subnets to which the specified tenant has access.

Default policy settings returns exclusively subnets owned by the tenant submitting the request, unless the request is submitted by an user with administrative rights. You can control which attributes are returned by using the fields query parameter. You can filter results by using query string parameters.

Normal response codes: 200

Error response codes: unauthorized (401)

1.3.1.1. Request

This table shows the query parameters for the list subnets request:

Name	Type	Description
`display_name`	String *(Optional)*	Name of the network.
`network_id`	Uuid *(Optional)*	The ID of the attached network.
`gateway_ip`	String *(Optional)*	The gateway IP address.
`ip_version`	String *(Optional)*	The IP version, which is 4 or 6.
`cidr`	Bool *(Optional)*	The CIDR.
`id`	Uuid *(Optional)*	The ID of the subnet.
`enable_dhcp`	Boolean *(Optional)*	If true, DHCP is enabled and If false, DHCP is disabled.
`ipv6_ra_mode`	String *(Optional)*	Choose from (constants.IPV6_SLAAC,constants.DHCPV6_STATEFUL,constants.DHCPV6_STATELESS,name=
`ipv6_address_mode`	String *(Optional)*	Choose from (constants.IPV6_SLAAC,constants.DHCPV6_STATEFUL,constants.DHCPV6_STATELESS,name=

1.3.1.2. Response

Example 1.19. List subnets: JSON response

```
{
    "subnets": [
        {
            "name": "private-subnet",
            "enable_dhcp": true,
            "network_id": "db193ab3-96e3-4cb3-8fc5-05f4296d0324",
            "tenant_id": "26a7980765d0414dbc1fc1f88cdb7e6e",
            "dns_nameservers": [],
            "allocation_pools": [
                {
                    "start": "10.0.0.2",
                    "end": "10.0.0.254"
                }
            ],
            "host_routes": [],
            "ip_version": 4,
            "gateway_ip": "10.0.0.1",
            "cidr": "10.0.0.0/24",
            "id": "08eae331-0402-425a-923c-34f7cfe39c1b"
        },
        {
            "name": "my_subnet",
            "enable_dhcp": true,
            "network_id": "d32019d3-bc6e-4319-9c1d-6722fc136a22",
            "tenant_id": "4fd44f30292945e481c7b8a0c8908869",
            "dns_nameservers": [],
            "allocation_pools": [
                {
                    "start": "192.0.0.2",
                    "end": "192.255.255.254"
                }
            ],
            "host_routes": [],
            "ip_version": 4,
            "gateway_ip": "192.0.0.1",
            "cidr": "192.0.0.0/8",
            "id": "54d6f61d-db07-451c-9ab3-b9609b6b6f0b"
        }
    ]
}
```

Example 1.20. List subnets: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<subnets>
    <subnet>
        <name>private-subnet</name>
        <enable_dhcp>True</enable_dhcp>
        <network_id>db193ab3-96e3-4cb3-8fc5-05f4296d0324</network_id>
        <tenant_id>26a7980765d0414dbc1fc1f88cdb7e6e</tenant_id>
        <dns_nameservers/>
        <allocation_pools>
            <allocation_pool>
                <start>10.0.0.2</start>
                <end>10.0.0.254</end>
            </allocation_pool>
        </allocation_pools>
        <host_routes/>
        <ip_version>4</ip_version>
        <gateway_ip>10.0.0.1</gateway_ip>
        <cidr>10.0.0.0/24</cidr>
        <id>08eae331-0402-425a-923c-34f7cfe39c1b</id>
    </subnet>
    <subnet>
        <name>my_subnet</name>
        <enable_dhcp>True</enable_dhcp>
        <network_id>d32019d3-bc6e-4319-9c1d-6722fc136a22</network_id>
        <tenant_id>4fd44f30292945e481c7b8a0c8908869</tenant_id>
        <dns_nameservers/>
        <allocation_pools>
            <allocation_pool>
                <start>192.0.0.2</start>
                <end>192.255.255.254</end>
            </allocation_pool>
        </allocation_pools>
        <host_routes/>
        <ip_version>4</ip_version>
        <gateway_ip>192.0.0.1</gateway_ip>
        <cidr>192.0.0.0/8</cidr>
        <id>54d6f61d-db07-451c-9ab3-b9609b6b6f0b</id>
    </subnet>
</subnets>
```

This operation does not return a response body.

1.3.2. Create subnet

Method	URI	Description
POST	`/v2.0/subnets`	Creates a subnet on a specified network.

OpenStack Networking does not try to derive the correct IP version from the specified CIDR. If you do not specify the `gateway_ip` attribute, OpenStack Networking allocates an address from the CIDR for the gateway for the subnet.

To specify a subnet without a gateway, set the `gateway_ip` attribute to `null` in the request body. If you do not specify the `allocation_pools` attribute, OpenStack Networking automatically allocates pools for covering all IP addresses in the CIDR, excluding the address reserved for the subnet gateway. Otherwise, you can explicitly specify allocation pools as shown in the following example.

When you specify both the `allocation_pools` and `gateway_ip` attributes, you must ensure that the gateway IP does not overlap with the specified allocation pools; otherwise a 409 Conflict error occurs.

A subnet can have one or more name servers and host routes. Hosts in this subnet use the specified name servers. Devices with IP addresses from this subnet, not including the local subnet route, use the specified host routes.

Specify the `ipv6_ra_mode` and `ipv6_address_mode` attributes to create subnets that support IPv6 configurations, such as Stateless Address Autoconfiguration (SLAAC), DHCPv6 Stateful, and DHCPv6 Stateless configurations.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401), forbidden (403), itemNotFound (404), conflict (409)

1.3.2.1. Request

Example 1.21. Create subnet: JSON request

```
{
    "subnet": {
        "network_id": "d32019d3-bc6e-4319-9c1d-6722fc136a22",
        "ip_version": 4,
        "cidr": "10.0.0.1"
    }
}
```

Example 1.22. Create subnet: XML request

```
<?xml version='1.0' encoding='UTF-8'?>
<subnet>
    <network_id>d32019d3-bc6e-4319-9c1d-6722fc136a22</network_id>
    <ip_version>4</ip_version>
    <cidr>10.0.0.1</cidr>
</subnet>
```

This operation does not accept a request body.

1.3.2.2. Response

Example 1.23. Create subnet: JSON response

```
{
    "subnet": {
        "name": "",
        "enable_dhcp": true,
        "network_id": "d32019d3-bc6e-4319-9c1d-6722fc136a22",
        "tenant_id": "4fd44f30292945e481c7b8a0c8908869",
        "dns_nameservers": [],
        "allocation_pools": [
            {
                "start": "192.168.199.2",
                "end": "192.168.199.254"
            }
        ],
        "host_routes": [],
        "ip_version": 4,
        "gateway_ip": "192.168.199.1",
        "cidr": "192.168.199.0/24",
        "id": "3b80198d-4f7b-4f77-9ef5-774d54e17126"
    }
}
```

Example 1.24. Create subnet: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<subnet>
    <name>test_subnet_1</name>
    <enable_dhcp>True</enable_dhcp>
    <network_id>d32019d3-bc6e-4319-9c1d-6722fc136a22</network_id>
    <tenant_id>4fd44f30292945e481c7b8a0c8908869</tenant_id>
    <dns_nameservers/>
    <allocation_pools>
        <allocation_pool>
            <start>192.0.0.2</start>
            <end>192.255.255.254</end>
        </allocation_pool>
    </allocation_pools>
    <host_routes/>
    <ip_version>4</ip_version>
    <gateway_ip>192.0.0.1</gateway_ip>
    <cidr>192.0.0.0/8</cidr>
    <id>54d6f61d-db07-451c-9ab3-b9609b6b6f0b</id>
</subnet>
```

This operation does not return a response body.

1.3.3. Bulk create subnet

Method	URI	Description
POST	`/v2.0/subnets`	Creates multiple subnets in a single request. Specify a list of subnets in the request body.

The bulk create operation is always atomic. Either all or no subnets in the request body are created.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401), forbidden (403), itemNotFound (404), conflict (409)

1.3.3.1. Request

Example 1.25. Bulk create subnet: JSON request

```
{
    "subnets": [
        {
            "cidr": "192.168.199.0/24",
            "ip_version": 4,
            "network_id": "e6031bc2-901a-4c66-82da-f4c32ed89406"
        },
        {
            "cidr": "10.56.4.0/22",
            "ip_version": 4,
            "network_id": "64239a54-dcc4-4b39-920b-b37c2144effa"
        }
    ]
}
```

Example 1.26. Bulk create subnet: XML request

```
<?xml version="1.0" encoding="UTF-8"?>
<subnets>
    <subnet>
        <name>test_subnet_1</name>
        <network_id>a3775a7d-9f8b-4148-be81-c84bbd0837ce</network_id>
        <cidr>10.0.0.0/8</cidr>
        <ip_version>4</ip_version>
    </subnet>
    <subnet>
        <name>test_subnet_2</name>
        <network_id>a3775a7d-9f8b-4148-be81-c84bbd0837ce</network_id>
        <cidr>192.168.0.0/16</cidr>
        <ip_version>4</ip_version>
    </subnet>
</subnets>
```

This operation does not accept a request body.

1.3.3.2. Response

Example 1.27. Bulk create subnet: JSON response

```
{
    "subnets": [
        {
            "allocation_pools": [
                {
                    "end": "192.168.199.254",
                    "start": "192.168.199.2"
                }
            ],
            "cidr": "192.168.199.0/24",
            "dns_nameservers": [],
            "enable_dhcp": true,
            "gateway_ip": "192.168.199.1",
            "host_routes": [],
            "id": "0468a7a7-290d-4127-aedd-6c9449775a24",
            "ip_version": 4,
            "name": "",
            "network_id": "e6031bc2-901a-4c66-82da-f4c32ed89406",
            "tenant_id": "d19231fc08ec4bc4829b668040d34512"
        },
        {
            "allocation_pools": [
                {
                    "end": "10.56.7.254",
                    "start": "10.56.4.2"
                }
            ],
            "cidr": "10.56.4.0/22",
            "dns_nameservers": [],
            "enable_dhcp": true,
            "gateway_ip": "10.56.4.1",
            "host_routes": [],
            "id": "b0e7435c-1512-45fb-aa9e-9a7c5932fb30",
            "ip_version": 4,
            "name": "",
            "network_id": "64239a54-dcc4-4b39-920b-b37c2144effa",
            "tenant_id": "d19231fc08ec4bc4829b668040d34512"
        }
    ]
}
```

Example 1.28. Bulk create subnet: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<subnets>
    <subnet>
        <name>test_subnet_1</name>
        <enable_dhcp>True</enable_dhcp>
        <network_id>a3775a7d-9f8b-4148-be81-c84bbd0837ce</network_id>
        <tenant_id>60cd4f6dbc2f491982a284e7b83b5be3</tenant_id>
        <dns_nameservers/>
        <allocation_pools>
            <allocation_pool>
                <start>10.0.0.2</start>
                <end>10.255.255.254</end>
            </allocation_pool>
        </allocation_pools>
        <host_routes/>
        <ip_version>4</ip_version>
        <gateway_ip>10.0.0.1</gateway_ip>
        <cidr>10.0.0.0/8</cidr>
        <id>bd3fd365-fe19-431a-be63-07017a09316c</id>
    </subnet>
    <subnet>
        <name>test_subnet_2</name>
        <enable_dhcp>True</enable_dhcp>
        <network_id>a3775a7d-9f8b-4148-be81-c84bbd0837ce</network_id>
        <tenant_id>60cd4f6dbc2f491982a284e7b83b5be3</tenant_id>
        <dns_nameservers/>
        <allocation_pools>
            <allocation_pool>
                <start>192.168.0.2</start>
                <end>192.168.255.254</end>
            </allocation_pool>
        </allocation_pools>
        <host_routes/>
        <ip_version>4</ip_version>
        <gateway_ip>192.168.0.1</gateway_ip>
        <cidr>192.168.0.0/16</cidr>
        <id>86e7c838-fb75-402b-9dbf-d68166e3f5fe</id>
    </subnet>
</subnets>
```

This operation does not return a response body.

1.3.4. Show subnet

Method	URI	Description
GET	`/v2.0/subnets/{subnet_id}`	Shows information for a specified subnet.

Use the fields query parameter to filter the results.

Normal response codes: 200

Error response codes: unauthorized (401), itemNotFound (404)

1.3.4.1. Request

This table shows the URI parameters for the show subnet request:

Name	Type	Description
`{subnet_id}`	UUID	The UUID for the subnet of interest to you.

This operation does not accept a request body.

1.3.4.2. Response

Example 1.29. Show subnet: JSON response

```
{
    "subnet": {
        "name": "my_subnet",
        "enable_dhcp": true,
        "network_id": "d32019d3-bc6e-4319-9c1d-6722fc136a22",
        "tenant_id": "4fd44f30292945e481c7b8a0c8908869",
        "dns_nameservers": [],
        "allocation_pools": [
            {
                "start": "192.0.0.2",
                "end": "192.255.255.254"
            }
        ],
        "host_routes": [],
        "ip_version": 4,
        "gateway_ip": "192.0.0.1",
        "cidr": "192.0.0.0/8",
        "id": "54d6f61d-db07-451c-9ab3-b9609b6b6f0b"
    }
}
```

Example 1.30. Show subnet: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<subnet>
    <name>test_subnet_1</name>
    <enable_dhcp>True</enable_dhcp>
    <network_id>d32019d3-bc6e-4319-9c1d-6722fc136a22</network_id>
    <tenant_id>4fd44f30292945e481c7b8a0c8908869</tenant_id>
    <dns_nameservers/>
    <allocation_pools>
```

```
        <allocation_pool>
            <start>192.0.0.2</start>
            <end>192.255.255.254</end>
        </allocation_pool>
    </allocation_pools>
    <host_routes/>
    <ip_version>4</ip_version>
    <gateway_ip>192.0.0.1</gateway_ip>
    <cidr>192.0.0.0/8</cidr>
    <id>54d6f61d-db07-451c-9ab3-b9609b6b6f0b</id>
</subnet>
```

This operation does not return a response body.

1.3.5. Update subnet

Method	URI	Description
PUT	`/v2.0/subnets/{subnet_id}`	Updates a specified subnet.

Some attributes, such as IP version (ip_version), and CIDR (cidr) cannot be updated. Attempting to update these attributes results in a `400 Bad Request` error.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), forbidden (403), itemNotFound (404)

1.3.5.1. Request

This table shows the URI parameters for the update subnet request:

Name	Type	Description
`{subnet_id}`	UUID	The UUID for the subnet of interest to you.

Example 1.31. Update subnet: JSON request

```
{
    "subnet": {
        "name": "my_subnet"
    }
}
```

Example 1.32. Update subnet: XML request

```
<?xml version="1.0" encoding="UTF-8"?>
<subnet>
    <name>my_subnet</name>
</subnet>
```

This operation does not accept a request body.

1.3.5.2. Response

Example 1.33. Update subnet: JSON response

```
{
    "subnet": {
        "name": "my_subnet",
        "enable_dhcp": true,
        "network_id": "db193ab3-96e3-4cb3-8fc5-05f4296d0324",
        "tenant_id": "26a7980765d0414dbc1fc1f88cdb7e6e",
        "dns_nameservers": [],
        "allocation_pools": [
            {
                "start": "10.0.0.2",
                "end": "10.0.0.254"
            }
        ],
        "host_routes": [],
        "ip_version": 4,
        "gateway_ip": "10.0.0.1",
        "cidr": "10.0.0.0/24",
        "id": "08eae331-0402-425a-923c-34f7cfe39c1b"
    }
}
```

Example 1.34. Update subnet: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<subnet>
    <name>my_subnet</name>
    <enable_dhcp>True</enable_dhcp>
    <network_id>d32019d3-bc6e-4319-9c1d-6722fc136a22</network_id>
    <tenant_id>4fd44f30292945e481c7b8a0c8908869</tenant_id>
    <dns_nameservers/>
    <allocation_pools>
        <allocation_pool>
            <start>192.0.0.2</start>
            <end>192.255.255.254</end>
        </allocation_pool>
    </allocation_pools>
    <host_routes/>
    <ip_version>4</ip_version>
    <gateway_ip>192.0.0.1</gateway_ip>
    <cidr>192.0.0.0/8</cidr>
    <id>54d6f61d-db07-451c-9ab3-b9609b6b6f0b</id>
</subnet>
```

This operation does not return a response body.

1.3.6. Delete subnet

Method	URI	Description
DELETE	`/v2.0/subnets/{subnet_id}`	Deletes a specified subnet.

The operation fails if subnet IP addresses are still allocated.

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404), conflict (409)

1.3.6.1. Request

This table shows the URI parameters for the delete subnet request:

Name	Type	Description
`{subnet_id}`	UUID	The UUID for the subnet of interest to you.

This operation does not accept a request body.

1.4. Ports

Lists, shows information for, creates, updates, and deletes ports.

Method	URI	Description
GET	`/v2.0/ports{?status,display_name,admin_state,network_id,device_owner,mac_address,port_id,security_groups,device_id}`	Lists ports to which the tenant has access.
POST	`/v2.0/ports`	Creates a port on a specified network.
POST	`/v2.0/ports`	Creates multiple ports in a single request. Specify a list of ports in the request body.
GET	`/v2.0/ports/{port_id}`	Shows information for a specified port.
PUT	`/v2.0/ports/{port_id}`	Updates a specified port.
DELETE	`/v2.0/ports/{port_id}`	Deletes a specified port.

1.4.1. List ports

Method	URI	Description
GET	`/v2.0/ports{?status,display_name, admin_state,network_id, device_owner,mac_address,port_id, security_groups,device_id}`	Lists ports to which the tenant has access.

Default policy settings return only those ports that are owned by the tenant who submits the request, unless the request is submitted by an user with administrative rights. Users can control which attributes are returned by using the fields query parameter. Additionally, you can filter results by using query string parameters. For information, see Filtering and Column Selection.

Normal response codes: 200

Error response codes: unauthorized (401)

1.4.1.1. Request

This table shows the query parameters for the list ports request:

Name	Type	Description
`status`	String *(Optional)*	The port status. Value is ACTIVE or DOWN.
`display_name`	String *(Optional)*	The port name.
`admin_state`	Bool *(Optional)*	The administrative state of the router, which is up (true) or down (false).
`network_id`	Uuid *(Optional)*	The ID of the attached network.
`device_owner`	String *(Optional)*	The ID of the entity that uses this port. For example, a DHCP agent.
`mac_address`	String *(Optional)*	The MAC address of the port.
`port_id`	Uuid *(Optional)*	The ID of the port.
`security_groups`	Uuid *(Optional)*	The IDs of any attached security groups.
`device_id`	Uuid *(Optional)*	The ID of the device that uses this port. For example, a virtual server.

1.4.1.2. Response

Example 1.35. List ports: JSON response

```
{
    "ports": [
        {
            "status": "ACTIVE",
            "name": "",
            "allowed_address_pairs": [],
            "admin_state_up": true,
            "network_id": "70c1db1f-b701-45bd-96e0-a313ee3430b3",
            "tenant_id": "",
            "extra_dhcp_opts": [],
            "device_owner": "network:router_gateway",
            "mac_address": "fa:16:3e:58:42:ed",
            "fixed_ips": [
                {
                    "subnet_id": "008ba151-0b8c-4a67-98b5-0d2b87666062",
                    "ip_address": "172.24.4.2"
                }
            ],
            "id": "d80b1a3b-4fc1-49f3-952e-1e2ab7081d8b",
            "security_groups": [],
            "device_id": "9ae135f4-b6e0-4dad-9e91-3c223e385824"
        },
        {
            "status": "ACTIVE",
            "name": "",
            "allowed_address_pairs": [],
            "admin_state_up": true,
            "network_id": "f27aa545-cbdd-4907-b0c6-c9e8b039dcc2",
            "tenant_id": "d397de8a63f341818f198abb0966f6f3",
            "extra_dhcp_opts": [],
            "device_owner": "network:router_interface",
            "mac_address": "fa:16:3e:bb:3c:e4",
            "fixed_ips": [
                {
                    "subnet_id": "288bf4a1-51ba-43b6-9d0a-520e9005db17",
                    "ip_address": "10.0.0.1"
                }
            ],
            "id": "f71a6703-d6de-4be1-a91a-a570ede1d159",
            "security_groups": [],
            "device_id": "9ae135f4-b6e0-4dad-9e91-3c223e385824"
        }
    ]
}
```

Example 1.36. List ports: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<ports xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <port>
        <status>ACTIVE</status>
        <name/>
        <allowed_address_pairs quantum:type="list"/>
        <admin_state_up quantum:type="bool">True</admin_state_up>
        <network_id>70c1db1f-b701-45bd-96e0-a313ee3430b3</network_id>
        <tenant_id/>
        <extra_dhcp_opts quantum:type="list"/>
        <device_owner>network:router_gateway</device_owner>
        <mac_address>fa:16:3e:58:42:ed</mac_address>
        <fixed_ips>
            <fixed_ip>
                <subnet_id>008ba151-0b8c-4a67-98b5-0d2b87666062</subnet_id>
                <ip_address>172.24.4.2</ip_address>
            </fixed_ip>
        </fixed_ips>
        <id>d80b1a3b-4fc1-49f3-952e-1e2ab7081d8b</id>
        <security_groups quantum:type="list"/>
        <device_id>9ae135f4-b6e0-4dad-9e91-3c223e385824</device_id>
    </port>
    <port>
        <status>ACTIVE</status>
        <name/>
        <allowed_address_pairs quantum:type="list"/>
        <admin_state_up quantum:type="bool">True</admin_state_up>
        <network_id>f27aa545-cbdd-4907-b0c6-c9e8b039dcc2</network_id>
        <tenant_id>d397de8a63f341818f198abb0966f6f3</tenant_id>
        <extra_dhcp_opts quantum:type="list"/>
        <device_owner>network:router_interface</device_owner>
        <mac_address>fa:16:3e:bb:3c:e4</mac_address>
        <fixed_ips>
            <fixed_ip>
                <subnet_id>288bf4a1-51ba-43b6-9d0a-520e9005db17</subnet_id>
                <ip_address>10.0.0.1</ip_address>
            </fixed_ip>
        </fixed_ips>
        <id>f71a6703-d6de-4be1-a91a-a570ede1d159</id>
        <security_groups quantum:type="list"/>
        <device_id>9ae135f4-b6e0-4dad-9e91-3c223e385824</device_id>
    </port>
</ports>
```

This operation does not return a response body.

1.4.2. Create port

Method	URI	Description
POST	`/v2.0/ports`	Creates a port on a specified network.

You must specify the `network_id` attribute in the request body to define the network where the port is to be created.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401), forbidden (403), itemNotFound (404), macGenerationFailure (503), serviceUnavailable (503)

1.4.2.1. Request

Example 1.37. Create port: JSON request

```
{
    "port": {
        "network_id": "a87cc70a-3e15-4acf-8205-9b711a3531b7",
        "name": "private-port",
        "admin_state_up": true
    }
}
```

Example 1.38. Create port: XML request

```
<?xml version="1.0" encoding="UTF-8"?>
<port>
    <name>test_port_1</name>
    <network_id>a87cc70a-3e15-4acf-8205-9b711a3531b7</network_id>
</port>
```

This operation does not accept a request body.

1.4.2.2. Response

Example 1.39. Create port: JSON response

```
{
    "port": {
        "status": "DOWN",
        "name": "private-port",
        "allowed_address_pairs": [],
        "admin_state_up": true,
        "network_id": "a87cc70a-3e15-4acf-8205-9b711a3531b7",
        "tenant_id": "d6700c0c9ffa4f1cb322cd4a1f3906fa",
        "device_owner": "",
        "mac_address": "fa:16:3e:c9:cb:f0",
        "fixed_ips": [
            {
                "subnet_id": "a0304c3a-4f08-4c43-88af-d796509c97d2",
                "ip_address": "10.0.0.2"
            }
        ],
        "id": "65c0ee9f-d634-4522-8954-51021b570b0d",
        "security_groups": [
            "f0ac4394-7e4a-4409-9701-ba8be283dbc3"
        ],
        "device_id": ""
    }
}
```

Example 1.40. Create port: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<port xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <status>DOWN</status>
    <name>test_port_1</name>
    <allowed_address_pairs quantum:type="list"/>
    <admin_state_up quantum:type="bool">True</admin_state_up>
    <network_id>a87cc70a-3e15-4acf-8205-9b711a3531b7</network_id>
    <tenant_id>d6700c0c9ffa4f1cb322cd4a1f3906fa</tenant_id>
    <device_owner/>
    <mac_address>fa:16:3e:09:e3:47</mac_address>
    <fixed_ips>
        <fixed_ip>
            <subnet_id>a0304c3a-4f08-4c43-88af-d796509c97d2</subnet_id>
            <ip_address>10.0.0.4</ip_address>
        </fixed_ip>
    </fixed_ips>
    <id>8021790b-4bfd-46ab-bcc7-0ef2f73bff43</id>
    <security_groups>
        <security_group>f0ac4394-7e4a-4409-9701-ba8be283dbc3</security_group>
    </security_groups>
    <device_id/>
</port>
```

This operation does not return a response body.

1.4.3. Bulk create ports

Method	URI	Description
POST	`/v2.0/ports`	Creates multiple ports in a single request. Specify a list of ports in the request body.

Guarantees the atomic completion of the bulk operation.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401), forbidden (403), itemNotFound (404), conflict (409), macGenerationFailure (503)

1.4.3.1. Request

Example 1.41. Bulk create ports: JSON request

```
{
    "ports": [
        {
            "name": "sample_port_1",
            "admin_state_up": false,
            "network_id": "a87cc70a-3e15-4acf-8205-9b711a3531b7"
        },
        {
            "name": "sample_port_2",
            "admin_state_up": false,
            "network_id": "a87cc70a-3e15-4acf-8205-9b711a3531b7"
        }
    ]
}
```

Example 1.42. Bulk create ports: XML request

```
<?xml version="1.0" encoding="UTF-8"?>
<ports>
    <port>
        <name>test_port_1-xml</name>
        <network_id>a87cc70a-3e15-4acf-8205-9b711a3531b7</network_id>
    </port>
    <port>
        <name>test_port_2-xml</name>
        <network_id>a87cc70a-3e15-4acf-8205-9b711a3531b7</network_id>
    </port>
</ports>
```

This operation does not accept a request body.

1.4.3.2. Response

Example 1.43. Bulk create ports: JSON response

```
{
    "ports": [
        {
            "status": "DOWN",
            "name": "sample_port_1",
            "allowed_address_pairs": [],
            "admin_state_up": false,
            "network_id": "a87cc70a-3e15-4acf-8205-9b711a3531b7",
            "tenant_id": "d6700c0c9ffa4f1cb322cd4a1f3906fa",
            "device_owner": "",
            "mac_address": "fa:16:3e:48:b8:9f",
            "fixed_ips": [
                {
                    "subnet_id": "a0304c3a-4f08-4c43-88af-d796509c97d2",
                    "ip_address": "10.0.0.5"
                }
            ],
            "id": "94225baa-9d3f-4b93-bf12-b41e7ce49cdb",
            "security_groups": [
                "f0ac4394-7e4a-4409-9701-ba8be283dbc3"
            ],
            "device_id": ""
        },
        {
            "status": "DOWN",
            "name": "sample_port_2",
            "allowed_address_pairs": [],
            "admin_state_up": false,
            "network_id": "a87cc70a-3e15-4acf-8205-9b711a3531b7",
            "tenant_id": "d6700c0c9ffa4f1cb322cd4a1f3906fa",
            "device_owner": "",
            "mac_address": "fa:16:3e:f4:73:df",
            "fixed_ips": [
                {
                    "subnet_id": "a0304c3a-4f08-4c43-88af-d796509c97d2",
                    "ip_address": "10.0.0.6"
                }
            ],
            "id": "235b09e0-63c4-47f1-b221-66ba54c21760",
            "security_groups": [
                "f0ac4394-7e4a-4409-9701-ba8be283dbc3"
            ],
            "device_id": ""
        }
    ]
}
```

Example 1.44. Bulk create ports: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<ports xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <port>
        <status>DOWN</status>
        <name>test_port_1-xml</name>
        <allowed_address_pairs quantum:type="list"/>
        <admin_state_up quantum:type="bool">True</admin_state_up>
        <network_id>a87cc70a-3e15-4acf-8205-9b711a3531b7</network_id>
        <tenant_id>d6700c0c9ffa4f1cb322cd4a1f3906fa</tenant_id>
        <device_owner/>
        <mac_address>fa:16:3e:fa:e2:34</mac_address>
        <fixed_ips>
```

```
            <fixed_ip>
                <subnet_id>a0304c3a-4f08-4c43-88af-d796509c97d2</subnet_id>
                <ip_address>10.0.0.7</ip_address>
            </fixed_ip>
        </fixed_ips>
        <id>054e8f14-4082-400e-afcc-5d6e5b3bcc0c</id>
        <security_groups>
            <security_group>f0ac4394-7e4a-4409-9701-ba8be283dbc3</
security_group>
        </security_groups>
        <device_id/>
    </port>
    <port>
        <status>DOWN</status>
        <name>test_port_2-xml</name>
        <allowed_address_pairs quantum:type="list"/>
        <admin_state_up quantum:type="bool">True</admin_state_up>
        <network_id>a87cc70a-3e15-4acf-8205-9b711a3531b7</network_id>
        <tenant_id>d6700c0c9ffa4f1cb322cd4a1f3906fa</tenant_id>
        <device_owner/>
        <mac_address>fa:16:3e:e6:cf:d9</mac_address>
        <fixed_ips>
            <fixed_ip>
                <subnet_id>a0304c3a-4f08-4c43-88af-d796509c97d2</subnet_id>
                <ip_address>10.0.0.8</ip_address>
            </fixed_ip>
        </fixed_ips>
        <id>879e96f9-6dd5-4232-bd19-3f39d0ae463b</id>
        <security_groups>
            <security_group>f0ac4394-7e4a-4409-9701-ba8be283dbc3</
security_group>
        </security_groups>
        <device_id/>
    </port>
</ports>
```

This operation does not return a response body.

1.4.4. Show port

Method	URI	Description
GET	`/v2.0/ports/{port_id}`	Shows information for a specified port.

Normal response codes: 200

Error response codes: unauthorized (401), itemNotFound (404)

1.4.4.1. Request

This table shows the URI parameters for the show port request:

Name	Type	Description
`{port_id}`	UUID	The UUID for the port of interest to you.

This operation does not accept a request body.

1.4.4.2. Response

Example 1.45. Show port: JSON response

```
{
    "port": {
        "status": "ACTIVE",
        "name": "",
        "allowed_address_pairs": [],
        "admin_state_up": true,
        "network_id": "a87cc70a-3e15-4acf-8205-9b711a3531b7",
        "tenant_id": "7e02058126cc4950b75f9970368ba177",
        "extra_dhcp_opts": [],
        "device_owner": "network:router_interface",
        "mac_address": "fa:16:3e:23:fd:d7",
        "fixed_ips": [
            {
                "subnet_id": "a0304c3a-4f08-4c43-88af-d796509c97d2",
                "ip_address": "10.0.0.1"
            }
        ],
        "id": "46d4bfb9-b26e-41f3-bd2e-e6dcc1ccedb2",
        "security_groups": [],
        "device_id": "5e3898d7-11be-483e-9732-b2f5eccd2b2e"
    }
}
```

Example 1.46. Show port: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<port xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <status>ACTIVE</status>
    <name/>
    <allowed_address_pairs quantum:type="list"/>
    <admin_state_up quantum:type="bool">True</admin_state_up>
    <network_id>a87cc70a-3e15-4acf-8205-9b711a3531b7</network_id>
    <tenant_id>7e02058126cc4950b75f9970368ba177</tenant_id>
    <extra_dhcp_opts quantum:type="list"/>
    <device_owner>network:router_interface</device_owner>
    <mac_address>fa:16:3e:23:fd:d7</mac_address>
    <fixed_ips>
        <fixed_ip>
            <subnet_id>a0304c3a-4f08-4c43-88af-d796509c97d2</subnet_id>
            <ip_address>10.0.0.1</ip_address>
        </fixed_ip>
    </fixed_ips>
    <id>46d4bfb9-b26e-41f3-bd2e-e6dcc1ccedb2</id>
    <security_groups quantum:type="list"/>
    <device_id>5e3898d7-11be-483e-9732-b2f5eccd2b2e</device_id>
</port>
```

This operation does not return a response body.

1.4.5. Update port

Method	URI	Description
PUT	`/v2.0/ports/{port_id}`	Updates a specified port.

You can update information for a port, such as its symbolic name and associated IPs. When you update IPs for a port, any previously associated IPs are removed, returned to the respective subnets allocation pools, and replaced by the IPs specified in the body for the update request. Therefore, this operation replaces the `fixed_ip` attribute when it is specified in the request body. If the updated IP addresses are not valid or are already in use, the operation fails and the existing IP addresses are not removed from the port.

When you update security groups for a port and the operation succeeds, any associated security groups are removed and replaced by the security groups specified in the body for the update request. Therefore, this operation replaces the `security_groups` attribute when you specify it in the request body. However, if the specified security groups are not valid, the operation fails and the existing security groups are not removed from the port.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), forbidden (403), itemNotFound (404), conflict (409)

1.4.5.1. Request

This table shows the URI parameters for the update port request:

Name	Type	Description
`{port_id}`	UUID	The UUID for the port of interest to you.

Example 1.47. Update port: JSON request

```
{
    "port": {
        "name": "test-for-port-update",
        "admin_state_up": true,
        "device_owner": "compute:nova",
        "binding:host_id": "test_for_port_update_host"
    }
}
```

Example 1.48. Update port: JSON request

```
<?xml version="1.0" encoding="UTF-8"?>
<port>
    <name>test-for-port-update</name>
    <device_owner>compute:nova</device_owner>
</port>
```

This operation does not accept a request body.

1.4.5.2. Response

Example 1.49. Update port: JSON response

```
{
    "port": {
        "status": "DOWN",
        "binding:host_id": "test_for_port_update_host",
        "allowed_address_pairs": [],
        "extra_dhcp_opts": [],
        "device_owner": "compute:nova",
        "binding:profile": {},
        "fixed_ips": [
            {
                "subnet_id": "898dec4a-74df-4193-985f-c76721bcc746",
                "ip_address": "20.20.0.4"
            }
        ],
        "id": "43c831e0-19ce-4a76-9a49-57b57e69428b",
        "security_groups": [
            "ce0179d6-8a94-4f7c-91c2-f3038e2acbd0"
        ],
        "device_id": "",
        "name": "test-for-port-update",
        "admin_state_up": true,
        "network_id": "883fc383-5ea1-4c8b-8916-e1ddb0a9f365",
        "tenant_id": "522eda8d23124b25bf03fe44f1986b74",
        "binding:vif_details": {},
        "binding:vnic_type": "normal",
        "binding:vif_type": "binding_failed",
        "mac_address": "fa:16:3e:11:11:5e"
    }
}
```

Example 1.50. Update port: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<port xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <status>DOWN</status>
    <name>test-for-port-update</name>
    <allowed_address_pairs quantum:type="list"/>
    <admin_state_up quantum:type="bool">True</admin_state_up>
    <network_id>a87cc70a-3e15-4acf-8205-9b711a3531b7</network_id>
    <tenant_id>d6700c0c9ffa4f1cb322cd4a1f3906fa</tenant_id>
    <device_owner>compute:nova</device_owner>
    <mac_address>fa:16:3e:11:11:5e</mac_address>
    <fixed_ips>
        <fixed_ip>
            <subnet_id>898dec4a-74df-4193-985f-c76721bcc746</subnet_id>
            <ip_address>20.20.0.4</ip_address>
        </fixed_ip>
    </fixed_ips>
    <id>43c831e0-19ce-4a76-9a49-57b57e69428b</id>
    <security_groups>
        <security_group>ce0179d6-8a94-4f7c-91c2-f3038e2acbd0</security_group>
    </security_groups>
    <device_id/>
```

```
</port>
```

This operation does not return a response body.

1.4.6. Delete port

Method	URI	Description
DELETE	`/v2.0/ports/{port_id}`	Deletes a specified port.

Any IP addresses that are associated with the port are returned to the respective subnets allocation pools.

Normal response codes: 204

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404)

1.4.6.1. Request

This table shows the URI parameters for the delete port request:

Name	Type	Description
`{port_id}`	UUID	The UUID for the port of interest to you.

This operation does not accept a request body.

2. Networking API v2.0 extensions (CURRENT)

Method	URI	Description
Extensions		
GET	`/v2.0/extensions`	Lists available Networking API extensions.
GET	`/v2.0/extensions/{alias}`	Gets detailed information for a specified extension.
Quotas extension (quotas)		
GET	`/v2.0/quotas`	Lists quotas for tenants who have non-default quota values.
GET	`/v2.0/quotas/{tenant_id}`	Shows quotas for a specified tenant.
PUT	`/v2.0/quotas/{tenant_id}`	Updates quotas for a specified tenant. Use when non-default quotas are desired.
DELETE	`/v2.0/quotas/{tenant_id}`	Resets quotas to default values for a specified tenant.
Networks provider extended attributes (networks)		
GET	`/v2.0/networks`	Lists networks that are accessible to the tenant who submits the request.
POST	`/v2.0/networks`	Creates a network.
GET	`/v2.0/networks/{network_id}`	Shows details for a specified network.
PUT	`/v2.0/networks/{network_id}`	Updates a specified network.
DELETE	`/v2.0/networks/{network_id}`	Deletes a specified network.
Networks multiple provider extension (networks)		
GET	`/v2.0/networks`	Lists networks that are accessible to the tenant who submits the request. Networks with multiple segments include the `segments` list in the response.
POST	`/v2.0/networks`	Creates a network with multiple segment mappings.
GET	`/v2.0/networks/{network_id}`	Shows details for a specified network with multiple segments.
VLAN transparency extension (networks)		
GET	`/v2.0/networks`	Lists networks. The response shows the VLAN transparency attribute.
POST	`/v2.0/networks`	Creates a VLAN-transparent network.
GET	`/v2.0/networks/{network_id}`	Shows details for a specified VLAN-transparent network.
Ports binding extended attributes (ports)		
GET	`/v2.0/ports{?status,display_name, admin_state,network_id, device_owner,mac_address,port_id, security_groups,device_id}`	Lists ports to which the tenant has access.
POST	`/v2.0/ports`	Creates a port on the specified network.
GET	`/v2.0/ports/{port_id}`	Shows information for the specified port.
PUT	`/v2.0/ports/{port_id}`	Updates the specified port.
Security groups and rules (security-groups)		
GET	`/v2.0/security-groups`	Lists OpenStack Networking security groups to which the specified tenant has access.
POST	`/v2.0/security-groups`	Creates an OpenStack Networking security group.
GET	`/v2.0/security-groups/ {security_group_id}{?verbose, fields}`	Shows details for a specified security group.

Method	URI	Description
DELETE	`/v2.0/security-groups/{security_group_id}`	Deletes an OpenStack Networking security group.
GET	`/v2.0/security-group-rules`	Lists a summary of all OpenStack Networking security group rules that the specified tenant can access.
POST	`/v2.0/security-group-rules`	Creates an OpenStack Networking security group rule.
GET	`/v2.0/security-group-rules/{rules-security-groups-id}`	Shows detailed information for a specified security group rule.
DELETE	`/v2.0/security-group-rules/{rules-security-groups-id}`	Deletes a specified rule from a OpenStack Networking security group.
	Layer-3 networking	
GET	`/v2.0/routers`	Lists logical routers that are accessible to the tenant who submits the request.
POST	`/v2.0/routers`	Creates a logical router.
GET	`/v2.0/routers/{router_id}`	Shows details for a specified router.
PUT	`/v2.0/routers/{router_id}`	Updates a logical router.
DELETE	`/v2.0/routers/{router_id}`	Deletes a logical router and, if present, its external gateway interface.
PUT	`/v2.0/routers/{router_id}/add_router_interface`	Adds an internal interface to a logical router.
PUT	`/v2.0/routers/{router_id}/remove_router_interface`	Removes an internal interface from a logical router.
GET	`/v2.0/floatingips`	Lists floating IPs that are accessible to the tenant who submits the request.
POST	`/v2.0/floatingips`	Creates a floating IP, and, if you specify port information, associates the floating IP with an internal port.
GET	`/v2.0/floatingips/{floatingip_id}`	Shows details for a specified floating IP.
PUT	`/v2.0/floatingips/{floatingip_id}`	Updates a floating IP and its association with an internal port.
DELETE	`/v2.0/floatingips/{floatingip_id}`	Deletes a floating IP and, if present, its associated port.
	Metering labels and rules	
GET	`/v2.0/metering/metering-labels`	Lists all l3 metering labels that belong to the specified tenant.
POST	`/v2.0/metering/metering-labels`	Creates a l3 metering label.
GET	`/v2.0/metering/metering-labels/{metering_label_id}`	Shows informations for a specified metering label.
DELETE	`/v2.0/metering/metering-labels/{metering_label_id}`	Deletes a l3 metering label.
GET	`/v2.0/metering/metering-label-rules`	Lists a summary of all l3 metering label rules belonging to the specified tenant.
POST	`/v2.0/metering/metering-label-rules`	Creates a l3 metering label rule.
GET	`/v2.0/metering/metering-label-rules/{metering-label-rule-id}`	Shows detailed informations for a specified metering label rule.
DELETE	`/v2.0/metering/metering-label-rules/{metering-label-rule-id}`	Deletes a specified l3 metering label rule.
	Load-Balancer-as-a-Service (LBaaS) 1.0 (STABLE)	
GET	`/v2.0/lb/vips`	Lists VIPs.
POST	`/v2.0/lb/vips`	Creates a load balancer VIP.
GET	`/v2.0/lb/vips/{vip_id}`	Shows details for a specified VIP.
PUT	`/v2.0/lb/vips/{vip_id}`	Updates a specified load balancer VIP.

Method	URI	Description
DELETE	`/v2.0/lb/vips/{vip_id}`	Deletes a specified load balancer VIP.
GET	`/v2.0/lb/healthmonitors`	Lists health monitors.
POST	`/v2.0/lb/healthmonitors`	Creates a load balancer health monitor.
GET	`/v2.0/lb/healthmonitors/{health_monitor_id}`	Shows details for a specified health monitor.
PUT	`/v2.0/lb/healthmonitors/{health_monitor_id}`	Updates a specified load balancer health monitor.
DELETE	`/v2.0/lb/healthmonitors/{health_monitor_id}`	Deletes a specified load balancer health monitor.
GET	`/v2.0/lb/pools`	Lists pools.
POST	`/v2.0/lb/pools`	Creates a load balancer pool.
GET	`/v2.0/lb/pools/{pool_id}`	Shows details for a specified pool.
PUT	`/v2.0/lb/pools/{pool_id}`	Updates a specified load balancer pool.
DELETE	`/v2.0/lb/pools/{pool_id}`	Deletes a specified load balancer pool.
POST	`/v2.0/lb/pools/{pool_id}/health_monitors`	Associates a health monitor with a specified pool.
DELETE	`/v2.0/lb/pools/{pool_id}/health_monitors/{health_monitor_id}`	Disassociates a specified health monitor from a pool.
GET	`/v2.0/lb/members`	Lists members.
POST	`/v2.0/lb/members`	Creates a load balancer member.
GET	`/v2.0/lb/members/{member_id}`	Shows details for a specified member.
PUT	`/v2.0/lb/members/{member_id}`	Updates a specified load balancer member.
DELETE	`/v2.0/lb/members/{member_id}`	Deletes a specified load balancer member.
Load-Balancer-as-a-Service (LBaaS) 2.0 (EXPERIMENTAL)		
GET	`/v2.0/lbaas/loadbalancers`	Lists load balancers.
POST	`/v2.0/lbaas/loadbalancers`	Creates a load balancer.
GET	`/v2.0/lbaas/loadbalancers/{loadbalancer_id}`	Shows details for a specified load balancer.
PUT	`/v2.0/lbaas/loadbalancers/{loadbalancer_id}`	Updates a specified load balancer.
DELETE	`/v2.0/lbaas/loadbalancers/{loadbalancer_id}`	Removes a specified load balancer.
GET	`/v2.0/lbaas/listeners`	Lists listeners.
POST	`/v2.0/lbaas/listeners`	Creates a listener.
GET	`/v2.0/lbaas/listeners/{listener_id}`	Shows details for a specified listener.
PUT	`/v2.0/lbaas/listeners/{listener_id}`	Updates a specified listener.
DELETE	`/v2.0/lbaas/listeners/{listener_id}`	Removes a specified listener.
GET	`/v2.0/lbaas/pools`	Lists pools.
POST	`/v2.0/lbaas/pools`	Creates a pool.
GET	`/v2.0/lbaas/pools/{pool_id}`	Shows details for a specified pool.
PUT	`/v2.0/lbaas/pools/{pool_id}`	Updates a specified pool.
DELETE	`/v2.0/lbaas/pools/{pool_id}`	Removes a specified pool.
GET	`/v2.0/lbaas/pools/{pool_id}/members`	Lists members of a specified pool.

Method	URI	Description
POST	`/v2.0/lbaas/pools/{pool_id}/members`	Adds a member to a pool.
GET	`/v2.0/lbaas/pools/{pool_id}/members/{member_id}`	Shows details for a specified pool member.
PUT	`/v2.0/lbaas/pools/{pool_id}/members/{member_id}`	Updates a specified member of a pool.
DELETE	`/v2.0/lbaas/pools/{pool_id}/members/{member_id}`	Removes a member from a pool.
POST	`/v2.0/lbaas/healthmonitors`	Creates a health monitor.
GET	`/v2.0/lbaas/healthmonitors`	Lists health monitors.
GET	`/v2.0/lbaas/healthmonitors/{health_monitor_id}`	Shows details for a specified health monitor.
PUT	`/v2.0/lbaas/healthmonitors/{health_monitor_id}`	Updates a specified health monitor.
DELETE	`/v2.0/lbaas/healthmonitors/{health_monitor_id}`	Removes a specified health monitor.
Virtual-Private-Network-as-a-Service (VPNaaS) 2.0 (CURRENT)		
GET	`/v2.0/vpn/vpnservices`	Lists VPN services.
POST	`/v2.0/vpn/vpnservices`	Creates a VPN service.
GET	`/v2.0/vpn/vpnservices/{service_id}`	Shows details for a specified VPN service.
PUT	`/v2.0/vpn/vpnservices/{service_id}`	Updates a specified VPN service.
DELETE	`/v2.0/vpn/vpnservices/{service_id}`	Removes a specified VPN service.
GET	`/v2.0/vpn/ikepolicies`	Lists IKE policies.
POST	`/v2.0/vpn/ikepolicies`	Creates an IKE policy.
GET	`/v2.0/vpn/ikepolicies/{ikepolicy_id}`	Shows details for a specified IKE policy.
PUT	`/v2.0/vpn/ikepolicies/{ikepolicy_id}`	Updates policy settings in a specified IKE policy.
DELETE	`/v2.0/vpn/ikepolicies/{ikepolicy_id}`	Removes a specified IKE policy.
GET	`/v2.0/vpn/ipsecpolicies`	Lists IPSec policies.
POST	`/v2.0/vpn/ipsecpolicies`	Creates an IPSec policy.
GET	`/v2.0/vpn/ipsecpolicies/{ipsecpolicy_id}`	Shows details for a specified IPSec policy.
PUT	`/v2.0/vpn/ipsecpolicies/{ipsecpolicy_id}`	Updates policy settings in a specified IPSec policy.
DELETE	`/v2.0/vpn/ipsecpolicies/{ipsecpolicy_id}`	Removes a specified IPSec policy.
GET	`/v2.0/vpn/ipsecsiteconnections`	Lists IPSec connections.
POST	`/v2.0/vpn/ipsecsiteconnections`	Creates an IPSec connection.
GET	`/v2.0/vpn/ipsecsiteconnections/{connection_id}`	Shows details for a specified IPSec connection.
PUT	`/v2.0/vpn/ipsecsiteconnections/{connection_id}`	Updates connection settings for a specified IPSec connection.
DELETE	`/v2.0/vpn/ipsecsiteconnections/{connection_id}`	Removes a specified IPSec connection.
Extra routes		
PUT	`/v2.0/routers/{router_id}`	Configures extra routes on a specified router.

2.1. Extensions

Lists available Networking API v2.0 extensions and shows details for a specified extension.

Method	URI	Description
GET	`/v2.0/extensions`	Lists available Networking API extensions.
GET	`/v2.0/extensions/{alias}`	Gets detailed information for a specified extension.

2.1.1. List extensions

Method	URI	Description
GET	`/v2.0/extensions`	Lists available Networking API extensions.

Normal response codes: 200, 203

Error response codes: computeFault (400, 500, …)

2.1.1.1. Request

This operation does not accept a request body.

2.1.1.2. Response

Example 2.1. List extensions: JSON response

```
{
    "extensions": [
        {
            "updated": "2013-01-20T00:00:00-00:00",
            "name": "Neutron Service Type Management",
            "links": [],
            "alias": "service-type",
            "description": "API for retrieving service providers for Neutron
 advanced services"
        },
        {
            "updated": "2012-10-05T10:00:00-00:00",
            "name": "security-group",
            "links": [],
            "alias": "security-group",
            "description": "The security groups extension."
        },
        {
            "updated": "2013-02-07T10:00:00-00:00",
            "name": "L3 Agent Scheduler",
            "links": [],
            "alias": "l3_agent_scheduler",
            "description": "Schedule routers among l3 agents"
        },
        {
            "updated": "2013-02-07T10:00:00-00:00",
            "name": "Loadbalancer Agent Scheduler",
            "links": [],
            "alias": "lbaas_agent_scheduler",
            "description": "Schedule pools among lbaas agents"
        },
        {
            "updated": "2013-03-28T10:00:00-00:00",
            "name": "Neutron L3 Configurable external gateway mode",
            "links": [],
            "alias": "ext-gw-mode",
            "description": "Extension of the router abstraction for specifying
 whether SNAT should occur on the external gateway"
        },
        {
            "updated": "2014-02-03T10:00:00-00:00",
            "name": "Port Binding",
            "links": [],
            "alias": "binding",
            "description": "Expose port bindings of a virtual port to external
application"
        },
        {
            "updated": "2012-09-07T10:00:00-00:00",
            "name": "Provider Network",
            "links": [],
            "alias": "provider",
            "description": "Expose mapping of virtual networks to physical
networks"
        },
        {
            "updated": "2013-02-03T10:00:00-00:00",
            "name": "agent",
            "links": [],
            "alias": "agent",
            "description": "The agent management extension."
        },
        {
            "updated": "2012-07-29T10:00:00-00:00",
            "name": "Quota management support",
            "links": [],
            "alias": "quotas",
            "description": "Expose functions for quotas management per tenant"
        },
        {
            "updated": "2013-02-07T10:00:00-00:00",
            "name": "DHCP Agent Scheduler",
            "links": [],
            "alias": "dhcp_agent_scheduler",
            "description": "Schedule networks among dhcp agents"
        },
        {
            "updated": "2013-06-27T10:00:00-00:00",
            "name": "Multi Provider Network",
            "links": [],
            "alias": "multi-provider",
            "description": "Expose mapping of virtual networks to multiple
physical networks"
        },
        {
            "updated": "2013-01-14T10:00:00-00:00",
            "name": "Neutron external network",
            "links": [],
            "alias": "external-net",
            "description": "Adds external network attribute to network
resource."
        },
        {
            "updated": "2012-07-20T10:00:00-00:00",
            "name": "Neutron L3 Router",
            "links": [],
            "alias": "router",
            "description": "Router abstraction for basic L3 forwarding between
L2 Neutron networks and access to external networks via a NAT gateway."
        },
        {
            "updated": "2013-07-23T10:00:00-00:00",
            "name": "Allowed Address Pairs",
            "links": [],
            "alias": "allowed-address-pairs",
            "description": "Provides allowed address pairs"
        },
        {
            "updated": "2013-03-17T12:00:00-00:00",
            "name": "Neutron Extra DHCP opts",
            "links": [],
            "alias": "extra_dhcp_opt",
            "description": "Extra options configuration for DHCP. For example
 PXE boot options to DHCP clients can be specified (e.g. tftp-server, server-
ip-address, bootfile-name)"
        },
        {
            "updated": "2012-10-07T10:00:00-00:00",
            "name": "LoadBalancing service",
            "links": [],
            "alias": "lbaas",
            "description": "Extension for LoadBalancing service"
        },
        {
            "updated": "2013-02-01T10:00:00-00:00",
            "name": "Neutron Extra Route",
            "links": [],
            "alias": "extraroute",
            "description": "Extra routes configuration for L3 router"
        }
    ]
}
```

Example 2.2. List extensions: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<extensions xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <extension>
        <updated>2013-01-20T00:00:00-00:00</updated>
        <name>Neutron Service Type Management</name>
        <links quantum:type="list"/>
        <alias>service-type</alias>
        <description>API for retrieving service providers for Neutron
            advanced services</description>
    </extension>
    <extension>
        <updated>2012-10-05T10:00:00-00:00</updated>
        <name>security-group</name>
        <links quantum:type="list"/>
        <alias>security-group</alias>
        <description>The security groups extension.</description>
    </extension>
    <extension>
        <updated>2013-02-07T10:00:00-00:00</updated>
        <name>L3 Agent Scheduler</name>
        <links quantum:type="list"/>
        <alias>l3_agent_scheduler</alias>
        <description>Schedule routers among l3 agents</description>
    </extension>
    <extension>
        <updated>2013-02-07T10:00:00-00:00</updated>
        <name>Loadbalancer Agent Scheduler</name>
        <links quantum:type="list"/>
        <alias>lbaas_agent_scheduler</alias>
        <description>Schedule pools among lbaas agents</description>
    </extension>
    <extension>
        <updated>2013-03-28T10:00:00-00:00</updated>
        <name>Neutron L3 Configurable external gateway mode</name>
        <links quantum:type="list"/>
        <alias>ext-gw-mode</alias>
        <description>Extension of the router abstraction for
            specifying whether SNAT should occur on the external
            gateway</description>
    </extension>
    <extension>
        <updated>2014-02-03T10:00:00-00:00</updated>
        <name>Port Binding</name>
        <links quantum:type="list"/>
        <alias>binding</alias>
        <description>Expose port bindings of a virtual port to
            external application</description>
    </extension>
    <extension>
        <updated>2012-09-07T10:00:00-00:00</updated>
        <name>Provider Network</name>
        <links quantum:type="list"/>
        <alias>provider</alias>
        <description>Expose mapping of virtual networks to physical
            networks</description>
    </extension>
    <extension>
        <updated>2013-02-03T10:00:00-00:00</updated>
        <name>agent</name>
        <links quantum:type="list"/>
        <alias>agent</alias>
        <description>The agent management extension.</description>
    </extension>
    <extension>
        <updated>2012-07-29T10:00:00-00:00</updated>
        <name>Quota management support</name>
        <links quantum:type="list"/>
        <alias>quotas</alias>
        <description>Expose functions for quotas management per
            tenant</description>
    </extension>
    <extension>
        <updated>2013-02-07T10:00:00-00:00</updated>
        <name>DHCP Agent Scheduler</name>
        <links quantum:type="list"/>
        <alias>dhcp_agent_scheduler</alias>
        <description>Schedule networks among dhcp agents</description>
    </extension>
    <extension>
        <updated>2013-06-27T10:00:00-00:00</updated>
        <name>Multi Provider Network</name>
        <links quantum:type="list"/>
        <alias>multi-provider</alias>
        <description>Expose mapping of virtual networks to multiple
            physical networks</description>
    </extension>
    <extension>
        <updated>2013-01-14T10:00:00-00:00</updated>
        <name>Neutron external network</name>
        <links quantum:type="list"/>
        <alias>external-net</alias>
        <description>Adds external network attribute to network
            resource.</description>
    </extension>
    <extension>
        <updated>2012-07-20T10:00:00-00:00</updated>
        <name>Neutron L3 Router</name>
        <links quantum:type="list"/>
        <alias>router</alias>
        <description>Router abstraction for basic L3 forwarding
            between L2 Neutron networks and access to external
            networks via a NAT gateway.</description>
    </extension>
    <extension>
        <updated>2013-07-23T10:00:00-00:00</updated>
        <name>Allowed Address Pairs</name>
        <links quantum:type="list"/>
        <alias>allowed-address-pairs</alias>
        <description>Provides allowed address pairs</description>
    </extension>
    <extension>
        <updated>2013-03-17T12:00:00-00:00</updated>
        <name>Neutron Extra DHCP opts</name>
        <links quantum:type="list"/>
        <alias>extra_dhcp_opt</alias>
        <description>Extra options configuration for DHCP. For example
            PXE boot options to DHCP clients can be specified (e.g.
            tftp-server, server-ip-address,
            bootfile-name)</description>
    </extension>
    <extension>
        <updated>2012-10-07T10:00:00-00:00</updated>
        <name>LoadBalancing service</name>
        <links quantum:type="list"/>
        <alias>lbaas</alias>
        <description>Extension for LoadBalancing service</description>
    </extension>
    <extension>
        <updated>2013-02-01T10:00:00-00:00</updated>
        <name>Neutron Extra Route</name>
        <links quantum:type="list"/>
        <alias>extraroute</alias>
        <description>Extra routes configuration for L3
            router</description>
    </extension>
</extensions>
```

This operation does not return a response body.

2.1.2. Get extension details

Method	URI	Description
GET	`/v2.0/extensions/{alias}`	Gets detailed information for a specified extension.

Normal response codes: 200, 203

Error response codes: computeFault (400, 500, …)

2.1.2.1. Request

This table shows the URI parameters for the get extension details request:

Name	Type	Description
`{alias}`	String	

This operation does not accept a request body.

2.1.2.2. Response

Example 2.3. Get extension details: JSON response

```
{
    "extension": {
        "updated": "2013-02-03T10:00:00-00:00",
        "name": "agent",
        "links": [],
        "alias": "agent",
        "description": "The agent management extension."
    }
}
```

Example 2.4. Get extension details: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<extension xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <updated>2013-02-03T10:00:00-00:00</updated>
    <name>agent</name>
    <links quantum:type="list"/>
    <alias>agent</alias>
    <description>The agent management extension.</description>
</extension>
```

This operation does not return a response body.

2.2. Quotas extension (quotas)

Lists, shows information for, updates, and resets quotas.

Method	URI	Description
GET	`/v2.0/quotas`	Lists quotas for tenants who have non-default quota values.

Method	URI	Description
GET	`/v2.0/quotas/{tenant_id}`	Shows quotas for a specified tenant.
PUT	`/v2.0/quotas/{tenant_id}`	Updates quotas for a specified tenant. Use when non-default quotas are desired.
DELETE	`/v2.0/quotas/{tenant_id}`	Resets quotas to default values for a specified tenant.

2.2.1. List quotas

Method	URI	Description
GET	`/v2.0/quotas`	Lists quotas for tenants who have non-default quota values.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403)

2.2.1.1. Request

This operation does not accept a request body.

2.2.1.2. Response

Example 2.5. List quotas: JSON response

```
{
    "quotas": [
        {
            "subnet": 10,
            "network": 10,
            "floatingip": 50,
            "tenant_id": "b7445f221cda4f4a8ac7db6b218b1339",
            "subnetpool": -1,
            "security_group_rule": 100,
            "security_group": 10,
            "router": 10,
            "port": 30
        }
    ]
}
```

2.2.2. Show quota

Method	URI	Description
GET	`/v2.0/quotas/{tenant_id}`	Shows quotas for a specified tenant.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403)

2.2.2.1. Request

This table shows the URI parameters for the show quota request:

Name	Type	Description
`{tenant_id}`	Uuid	The tenant ID.

This operation does not accept a request body.

2.2.2.2. Response

Example 2.6. Show quota: JSON response

```
{
    "quota": {
        "subnet": 10,
        "router": 10,
        "port": 50,
        "network": 10,
        "floatingip": 50,
        "subnetpool": -1,
        "security_group_rule": 100,
        "security_group": 1
    }
}
```

2.2.3. Update quota

Method	URI	Description
PUT	`/v2.0/quotas/{tenant_id}`	Updates quotas for a specified tenant. Use when non-default quotas are desired.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403)

2.2.3.1. Request

This table shows the URI parameters for the update quota request:

Name	Type	Description
`{tenant_id}`	Uuid	The tenant ID.

Example 2.7. Update quota: JSON request

```
{
    "quota": {
        "subnet": 40,
        "router": 50,
        "network": 10,
        "floatingip": 30,
        "port": 30
    }
}
```

2.2.3.2. Response

Example 2.8. Update quota: JSON response

```
{
    "quota": {
        "subnet": 40,
        "router": 50,
        "port": 30,
        "network": 10,
        "floatingip": 30
    }
}
```

2.2.4. Reset quota

Method	URI	Description
DELETE	`/v2.0/quotas/{tenant_id}`	Resets quotas to default values for a specified tenant.

Normal response codes: 204

Error response codes: unauthorized (401), forbidden (403)

2.2.4.1. Request

This table shows the URI parameters for the reset quota request:

Name	Type	Description
`{tenant_id}`	Uuid	The tenant ID.

This operation does not accept a request body.

2.3. Networks provider extended attributes (networks)

Lists, creates, shows information for, updates, and deletes networks.

Method	URI	Description
GET	`/v2.0/networks`	Lists networks that are accessible to the tenant who submits the request.
POST	`/v2.0/networks`	Creates a network.
GET	`/v2.0/networks/{network_id}`	Shows details for a specified network.
PUT	`/v2.0/networks/{network_id}`	Updates a specified network.
DELETE	`/v2.0/networks/{network_id}`	Deletes a specified network.

2.3.1. List networks

Method	URI	Description
GET	`/v2.0/networks`	Lists networks that are accessible to the tenant who submits the request.

Normal response codes: 200

2.3.1.1. Request

This operation does not accept a request body.

2.3.1.2. Response

Example 2.9. List networks: JSON response

```
{
    "network": {
        "status": "ACTIVE",
        "subnets": [
            "54d6f61d-db07-451c-9ab3-b9609b6b6f0b"
        ],
        "name": "private-network",
        "router:external": false,
        "admin_state_up": true,
        "tenant_id": "4fd44f30292945e481c7b8a0c8908869",
        "mtu": 0,
        "shared": true,
        "port_security_enabled": true,
        "id": "d32019d3-bc6e-4319-9c1d-6722fc136a22"
    }
}
```

Example 2.10. List networks: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<network xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:provider="http://docs.openstack.org/ext/provider/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:router="http://docs.openstack.org/ext/neutron/router/api/v1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <status>ACTIVE</status>
    <subnets>
        <subnet>54d6f61d-db07-451c-9ab3-b9609b6b6f0b</subnet>
    </subnets>
    <name>private-network</name>
    <admin_state_up quantum:type="bool">True</admin_state_up>
    <tenant_id>4fd44f30292945e481c7b8a0c8908869</tenant_id>
    <shared quantum:type="bool">True</shared>
    <id>d32019d3-bc6e-4319-9c1d-6722fc136a22</id>
</network>
```

This operation does not return a response body.

2.3.2. Create network

Method	URI	Description
POST	`/v2.0/networks`	Creates a network.

Normal response codes: 201

2.3.2.1. Request

Example 2.11. Create network: JSON request

```
{
    "network": {
        "name": "sample_network",
        "admin_state_up": true
    }
}
```

Example 2.12. Create network: XML request

```
<?xml version="1.0" encoding="UTF-8"?>
<network>
    <name>sample_network2</name>
</network>
```

This operation does not accept a request body.

2.3.2.2. Response

Example 2.13. Create network: JSON response

```
{
    "network": {
        "status": "ACTIVE",
        "subnets": [],
        "name": "net1",
        "admin_state_up": true,
        "tenant_id": "9bacb3c5d39d41a79512987f338cf177",
        "router:external": false,
        "segments": [
            {
                "provider:segmentation_id": 2,
                "provider:physical_network": "8bab8453-1bc9-45af-8c70-
f83aa9b50453",
                "provider:network_type": "vlan"
            },
            {
                "provider:segmentation_id": null,
                "provider:physical_network": "8bab8453-1bc9-45af-8c70-
f83aa9b50453",
                "provider:network_type": "stt"
            }
        ],
        "shared": false,
        "id": "4e8e5957-649f-477b-9e5b-f1f75b21c03c"
    }
}
```

Example 2.14. Create network: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<network xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:provider="http://docs.openstack.org/ext/provider/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <status>ACTIVE</status>
    <subnets quantum:type="list"/>
    <name>sample_network2</name>
    <provider:physical_network xsi:nil="true"/>
    <admin_state_up quantum:type="bool">True</admin_state_up>
    <tenant_id>4fd44f30292945e481c7b8a0c8908869</tenant_id>
    <provider:network_type>local</provider:network_type>
    <shared quantum:type="bool">False</shared>
    <id>c220b026-ece1-4ead-873f-83537f4c9f92</id>
    <provider:segmentation_id xsi:nil="true"/>
</network>
```

This operation does not return a response body.

2.3.3. Show network details

Method	URI	Description
GET	`/v2.0/networks/{network_id}`	Shows details for a specified network.

Normal response codes: 200

2.3.3.1. Request

This table shows the URI parameters for the show network details request:

Name	Type	Description
`{network_id}`	UUID	The UUID for the network of interest to you.

This operation does not accept a request body.

2.3.3.2. Response

Example 2.15. Show network details: JSON response

```
{
    "network": {
        "status": "ACTIVE",
        "subnets": [
            "54d6f61d-db07-451c-9ab3-b9609b6b6f0b"
        ],
        "name": "private-network",
        "router:external": false,
        "admin_state_up": true,
        "tenant_id": "4fd44f30292945e481c7b8a0c8908869",
        "mtu": 0,
        "shared": true,
        "port_security_enabled": true,
        "id": "d32019d3-bc6e-4319-9c1d-6722fc136a22"
    }
}
```

Example 2.16. Show network details: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<network xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:provider="http://docs.openstack.org/ext/provider/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:router="http://docs.openstack.org/ext/neutron/router/api/v1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <status>ACTIVE</status>
    <subnets>
        <subnet>54d6f61d-db07-451c-9ab3-b9609b6b6f0b</subnet>
    </subnets>
    <name>private-network</name>
    <admin_state_up quantum:type="bool">True</admin_state_up>
    <tenant_id>4fd44f30292945e481c7b8a0c8908869</tenant_id>
    <shared quantum:type="bool">True</shared>
    <id>d32019d3-bc6e-4319-9c1d-6722fc136a22</id>
</network>
```

This operation does not return a response body.

2.3.4. Update network

Method	URI	Description
PUT	`/v2.0/networks/{network_id}`	Updates a specified network.

Normal response codes: 201

2.3.4.1. Request

This table shows the URI parameters for the update network request:

Name	Type	Description
`{network_id}`	UUID	The UUID for the network of interest to you.

Example 2.17. Update network: JSON request

```
{
    "network": {
        "name": "sample_network_5_updated"
    }
}
```

Example 2.18. Update network: XML request

```
<?xml version="1.0" encoding="UTF-8"?>
<network xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:provider="http://docs.openstack.org/ext/provider/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:router="http://docs.openstack.org/ext/quantum/router/api/v1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <name>sample-network-4-updated</name>
</network>
```

This operation does not accept a request body.

2.3.4.2. Response

Example 2.19. Update network: JSON response

```
{
    "network": {
        "status": "ACTIVE",
        "subnets": [],
        "name": "sample_network_5_updated",
        "provider:physical_network": null,
        "admin_state_up": true,
        "tenant_id": "4fd44f30292945e481c7b8a0c8908869",
        "provider:network_type": "local",
        "router:external": false,
        "mtu": 0,
        "shared": false,
        "port_security_enabled": true,
        "id": "1f370095-98f6-4079-be64-6d3d4a6adcc6",
        "provider:segmentation_id": null
    }
}
```

Example 2.20. Update network: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<network xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:provider="http://docs.openstack.org/ext/provider/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:router="http://docs.openstack.org/ext/neutron/router/api/v1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <status>ACTIVE</status>
    <subnets quantum:type="list"/>
    <name>sample-network-4-updated</name>
    <provider:physical_network xsi:nil="true"/>
    <admin_state_up quantum:type="bool">True</admin_state_up>
    <tenant_id>4fd44f30292945e481c7b8a0c8908869</tenant_id>
    <provider:network_type>local</provider:network_type>
    <router:external quantum:type="bool">False</router:external>
    <shared quantum:type="bool">False</shared>
    <id>af374017-c9ae-4a1d-b799-ab73111476e2</id>
    <provider:segmentation_id xsi:nil="true"/>
</network>
```

This operation does not return a response body.

2.3.5. Delete network

Method	URI	Description
DELETE	`/v2.0/networks/{network_id}`	Deletes a specified network.

2.3.5.1. Request

This table shows the URI parameters for the delete network request:

Name	Type	Description
`{network_id}`	UUID	The UUID for the network of interest to you.

This operation does not accept a request body.

2.4. Networks multiple provider extension (networks)

Enables administrative users to define multiple physical bindings for an OpenStack Networking network and list or show details for networks with multiple physical bindings.

You cannot update any `provider` attributes. If you try to do so, an error occurs.

To delete a network with multiple physical bindings, issue a normal delete network request.

To define multiple physical bindings for a network, include a `segments` list in the request body of a `POST /v2.0/networks` request. Each element in the `segments` list has the same structure as the provider network attributes. These attributes are `provider:network_type`, `provider:physical_network`, and `provider:segmentation_id`. The validation rules for these attributes are the same as for the Networks provider extended attributes. You cannot use both extensions at the same time.

The NSX and ML2 plug-ins support this extension. With the ML2 plug-in, you can specify multiple VLANs for a specified network, a VXLAN tunnel ID, and a VLAN.

Method	URI	Description
GET	`/v2.0/networks`	Lists networks that are accessible to the tenant who submits the request. Networks with multiple segments include the `segments` list in the response.
POST	`/v2.0/networks`	Creates a network with multiple segment mappings.
GET	`/v2.0/networks/{network_id}`	Shows details for a specified network with multiple segments.

2.4.1. List networks

Method	URI	Description
GET	`/v2.0/networks`	Lists networks that are accessible to the tenant who submits the request. Networks with multiple segments include the `segments` list in the response.

Normal response codes: 200

2.4.1.1. Request

This operation does not accept a request body.

2.4.1.2. Response

Example 2.21. List networks: JSON response

```
{
    "networks": [
        {
            "status": "ACTIVE",
            "subnets": [],
            "name": "net1",
            "admin_state_up": true,
            "tenant_id": "9bacb3c5d39d41a79512987f338cf177",
            "segments": [
                {
                    "provider:segmentation_id": 2,
                    "provider:physical_network": "8bab8453-1bc9-45af-8c70-
f83aa9b50453",
                    "provider:network_type": "vlan"
                },
                {
                    "provider:segmentation_id": 0,
                    "provider:physical_network": "8bab8453-1bc9-45af-8c70-
f83aa9b50453",
                    "provider:network_type": "stt"
                }
            ],
            "router:external": false,
            "shared": false,
            "id": "4e8e5957-649f-477b-9e5b-f1f75b21c03c"
        },
        {
            "status": "ACTIVE",
            "subnets": [
                "08eae331-0402-425a-923c-34f7cfe39c1b"
            ],
            "name": "private",
            "provider:physical_network": null,
            "router:external": true,
            "admin_state_up": true,
            "tenant_id": "26a7980765d0414dbc1fc1f88cdb7e6e",
            "provider:network_type": "local",
            "shared": true,
            "id": "db193ab3-96e3-4cb3-8fc5-05f4296d0324",
            "provider:segmentation_id": null
        }
    ]
}
```

2.4.2. Create network with multiple segment mappings

Method	URI	Description
POST	`/v2.0/networks`	Creates a network with multiple segment mappings.

Normal response codes: 201

2.4.2.1. Request

Example 2.22. Create network with multiple segment mappings: JSON request

```
{
    "network": {
        "segments": [
            {
                "provider:segmentation_id": "2",
                "provider:physical_network": "8bab8453-1bc9-45af-8c70-
f83aa9b50453",
                "provider:network_type": "vlan"
            },
            {
                "provider:physical_network": "8bab8453-1bc9-45af-8c70-
f83aa9b50453",
                "provider:network_type": "stt"
            }
        ],
        "name": "net1",
        "admin_state_up": true
    }
}
```

2.4.2.2. Response

Example 2.23. Create network with multiple segment mappings: JSON response

```
{
    "network": {
        "status": "ACTIVE",
        "subnets": [],
        "name": "net1",
        "admin_state_up": true,
        "tenant_id": "9bacb3c5d39d41a79512987f338cf177",
        "segments": [
            {
                "provider:segmentation_id": 2,
                "provider:physical_network": "8bab8453-1bc9-45af-8c70-
f83aa9b50453",
                "provider:network_type": "vlan"
            },
            {
                "provider:segmentation_id": null,
                "provider:physical_network": "8bab8453-1bc9-45af-8c70-
f83aa9b50453",
                "provider:network_type": "stt"
            }
        ],
        "shared": false,
        "id": "4e8e5957-649f-477b-9e5b-f1f75b21c03c"
    }
}
```

2.4.3. Show details for a network with multiple segments

Method	URI	Description
GET	`/v2.0/networks/{network_id}`	Shows details for a specified network with multiple segments.

Normal response codes: 200

2.4.3.1. Request

This table shows the URI parameters for the show details for a network with multiple segments request:

Name	Type	Description
`{network_id}`	UUID	The UUID for the network of interest to you.

This operation does not accept a request body.

2.4.3.2. Response

Example 2.24. Show details for a network with multiple segments: JSON response

```
{
    "network": {
        "status": "ACTIVE",
        "subnets": [],
        "name": "net1",
        "admin_state_up": true,
        "tenant_id": "9bacb3c5d39d41a79512987f338cf177",
        "segments": [
            {
                "provider:segmentation_id": 2,
                "provider:physical_network": "8bab8453-1bc9-45af-8c70-
f83aa9b50453",
                "provider:network_type": "vlan"
            },
            {
                "provider:segmentation_id": 0,
                "provider:physical_network": "8bab8453-1bc9-45af-8c70-
f83aa9b50453",
                "provider:network_type": "stt"
            }
        ],
        "router:external": false,
        "shared": false,
        "id": "4e8e5957-649f-477b-9e5b-f1f75b21c03c"
    }
}
```

2.5. VLAN transparency extension (networks)

Enables plug-ins that support VLAN transparency to deliver VLAN-transparent trunk networks. If the service does not support VLAN transparency and a user requests a VLAN-transparent network, the plug-in refuses to create one and returns an appropriate error to the user.

You cannot update the `vlan-transparent` attribute. If you try to do so, an error occurs.

To delete a VLAN-transparent network, issue a normal delete network request.

The ML2 plug-in currently supports this extension. With the ML2 plug-in, you can set the `vlan-transparent` attribute to either `true` or `false`.

Method	URI	Description
GET	`/v2.0/networks`	Lists networks. The response shows the VLAN transparency attribute.
POST	`/v2.0/networks`	Creates a VLAN-transparent network.
GET	`/v2.0/networks/{network_id}`	Shows details for a specified VLAN-transparent network.

2.5.1. List networks with VLAN transparency attribute

Method	URI	Description
GET	`/v2.0/networks`	Lists networks. The response shows the VLAN transparency attribute.

Normal response codes: 200

2.5.1.1. Request

This operation does not accept a request body.

2.5.1.2. Response

Example 2.25. List networks with VLAN transparency attribute: JSON response

```
{
    "networks": [
        {
            "status": "ACTIVE",
            "subnets": [],
            "name": "net1",
            "admin_state_up": true,
            "tenant_id": "e252a863-92ee-480f-8bd8-71be77089499",
            "shared": false,
            "router:external": false,
            "vlan_transparent": true,
            "id": "f5e6d63c-04a4-4b2c-8b27-a9854412d5a7"
        },
        {
            "status": "ACTIVE",
            "subnets": [
                "3daba37a-bced-4153-a4bb-d83dcc0552d9"
            ],
            "name": "private",
            "admin_state_up": true,
            "tenant_id": "109e5fae-d976-4791-84c7-6ae0bb3896c3",
            "shared": true,
            "router:external": false,
            "vlan_transparent": false,
            "id": "37e11503-3244-49f1-b92a-9f21bab017d9"
        }
    ]
}
```

2.5.2. Create VLAN-transparent network

Method	URI	Description
POST	`/v2.0/networks`	Creates a VLAN-transparent network.

Normal response codes: 201

2.5.2.1. Request

Example 2.26. Create VLAN-transparent network: JSON request

```
{
    "network": {
        "name": "net1",
        "admin_state_up": true,
        "vlan_transparent": true
    }
}
```

2.5.2.2. Response

Example 2.27. Create VLAN-transparent network: JSON response

```
{
    "network": {
        "status": "ACTIVE",
        "subnets": [],
        "name": "net1",
        "admin_state_up": true,
        "vlan_transparent": true,
        "tenant_id": "5831268f-1f52-49a7-88d5-bc0d7a74d523",
        "router:external": false,
        "shared": false,
        "id": "3114f6e9-f9bc-4570-a941-7329b3b9759f"
    }
}
```

2.5.3. Show VLAN-transparent network details

Method	URI	Description
GET	`/v2.0/networks/{network_id}`	Shows details for a specified VLAN-transparent network.

Normal response codes: 200

2.5.3.1. Request

This table shows the URI parameters for the show vlan-transparent network details request:

Name	Type	Description
`{network_id}`	UUID	The UUID for the network of interest to you.

This operation does not accept a request body.

2.5.3.2. Response

Example 2.28. Show VLAN-transparent network details: JSON response

```
{
    "network": {
        "status": "ACTIVE",
        "subnets": [],
        "name": "net1",
        "admin_state_up": true,
        "tenant_id": "e926fd5a-e9f6-4dc8-8043-a352d974ceaf",
        "router:external": false,
        "vlan_transparent": true,
        "shared": false,
        "id": "20403fe9-6c9c-48e5-9edb-c3426a955068"
    }
}
```

2.6. Ports binding extended attributes (ports)

Lists, creates, shows information for, and updates ports.

Method	URI	Description
GET	`/v2.0/ports{?status,display_name, admin_state,network_id, device_owner,mac_address,port_id, security_groups,device_id}`	Lists ports to which the tenant has access.
POST	`/v2.0/ports`	Creates a port on the specified network.
GET	`/v2.0/ports/{port_id}`	Shows information for the specified port.
PUT	`/v2.0/ports/{port_id}`	Updates the specified port.

2.6.1. List ports

Method	URI	Description
GET	`/v2.0/ports{?status,display_name, admin_state,network_id, device_owner,mac_address,port_id, security_groups,device_id}`	Lists ports to which the tenant has access.

Normal response codes: 200

2.6.1.1. Request

This table shows the query parameters for the list ports request:

Name	Type	Description
`status`	String *(Optional)*	The port status. Value is ACTIVE or DOWN.
`display_name`	String *(Optional)*	The port name.
`admin_state`	Bool *(Optional)*	The administrative state of the router, which is up (true) or down (false).
`network_id`	Uuid *(Optional)*	The ID of the attached network.
`device_owner`	String *(Optional)*	The ID of the entity that uses this port. For example, a DHCP agent.
`mac_address`	String *(Optional)*	The MAC address of the port.
`port_id`	Uuid *(Optional)*	The ID of the port.
`security_groups`	Uuid *(Optional)*	The IDs of any attached security groups.
`device_id`	Uuid *(Optional)*	The ID of the device that uses this port. For example, a virtual server.

2.6.1.2. Response

Example 2.29. List ports: JSON response

```
{
    "ports": [
        {
            "status": "ACTIVE",
            "binding:host_id": "devstack",
            "name": "",
            "allowed_address_pairs": [],
            "admin_state_up": true,
            "network_id": "70c1db1f-b701-45bd-96e0-a313ee3430b3",
            "tenant_id": "",
            "extra_dhcp_opts": [],
            "binding:vif_details": {
                "port_filter": true,
                "ovs_hybrid_plug": true
            },
            "binding:vif_type": "ovs",
            "device_owner": "network:router_gateway",
            "port_security_enabled": true,
            "mac_address": "fa:16:3e:58:42:ed",
            "binding:profile": {},
            "binding:vnic_type": "normal",
            "fixed_ips": [
                {
                    "subnet_id": "008ba151-0b8c-4a67-98b5-0d2b87666062",
                    "ip_address": "172.24.4.2"
                }
            ],
            "id": "d80b1a3b-4fc1-49f3-952e-1e2ab7081d8b",
            "security_groups": [],
            "device_id": "9ae135f4-b6e0-4dad-9e91-3c223e385824"
        },
        {
            "status": "ACTIVE",
            "binding:host_id": "devstack",
            "name": "",
            "allowed_address_pairs": [],
            "admin_state_up": true,
            "network_id": "f27aa545-cbdd-4907-b0c6-c9e8b039dcc2",
            "tenant_id": "d397de8a63f341818f198abb0966f6f3",
            "extra_dhcp_opts": [],
            "binding:vif_details": {
                "port_filter": true,
                "ovs_hybrid_plug": true
            },
            "binding:vif_type": "ovs",
            "device_owner": "network:router_interface",
            "port_security_enabled": true,
            "mac_address": "fa:16:3e:bb:3c:e4",
            "binding:profile": {},
            "binding:vnic_type": "normal",
            "fixed_ips": [
                {
                    "subnet_id": "288bf4a1-51ba-43b6-9d0a-520e9005db17",
                    "ip_address": "10.0.0.1"
                }
            ],
            "id": "f71a6703-d6de-4be1-a91a-a570ede1d159",
            "security_groups": [],
            "device_id": "9ae135f4-b6e0-4dad-9e91-3c223e385824"
        }
    ]
}
```

Example 2.30. List ports: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<ports xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:binding="http://docs.openstack.org/ext/binding/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <port>
```

```
        <status>ACTIVE</status>
        <binding:host_id>devstack</binding:host_id>
        <name/>
        <allowed_address_pairs quantum:type="list"/>
        <admin_state_up quantum:type="bool">True</admin_state_up>
        <network_id>70c1db1f-b701-45bd-96e0-a313ee3430b3</network_id>
        <tenant_id/>
        <extra_dhcp_opts quantum:type="list"/>
        <binding:vif_details>
            <port_filter quantum:type="bool">True</port_filter>
            <ovs_hybrid_plug quantum:type="bool"
                >True</ovs_hybrid_plug>
        </binding:vif_details>
        <binding:vif_type>ovs</binding:vif_type>
        <device_owner>network:router_gateway</device_owner>
        <mac_address>fa:16:3e:58:42:ed</mac_address>
        <binding:profile quantum:type="dict"/>
        <binding:vnic_type>normal</binding:vnic_type>
        <fixed_ips>
            <fixed_ip>
                <subnet_id>008ba151-0b8c-4a67-98b5-0d2b87666062</subnet_id>
                <ip_address>172.24.4.2</ip_address>
            </fixed_ip>
        </fixed_ips>
        <id>d80b1a3b-4fc1-49f3-952e-1e2ab7081d8b</id>
        <security_groups quantum:type="list"/>
        <device_id>9ae135f4-b6e0-4dad-9e91-3c223e385824</device_id>
    </port>
    <port>
        <status>ACTIVE</status>
        <binding:host_id>devstack</binding:host_id>
        <name/>
        <allowed_address_pairs quantum:type="list"/>
        <admin_state_up quantum:type="bool">True</admin_state_up>
        <network_id>f27aa545-cbdd-4907-b0c6-c9e8b039dcc2</network_id>
        <tenant_id>d397de8a63f341818f198abb0966f6f3</tenant_id>
        <extra_dhcp_opts quantum:type="list"/>
        <binding:vif_details>
            <port_filter quantum:type="bool">True</port_filter>
            <ovs_hybrid_plug quantum:type="bool"
                >True</ovs_hybrid_plug>
        </binding:vif_details>
        <binding:vif_type>ovs</binding:vif_type>
        <device_owner>network:router_interface</device_owner>
        <mac_address>fa:16:3e:bb:3c:e4</mac_address>
        <binding:profile quantum:type="dict"/>
        <binding:vnic_type>normal</binding:vnic_type>
        <fixed_ips>
            <fixed_ip>
                <subnet_id>288bf4a1-51ba-43b6-9d0a-520e9005db17</subnet_id>
                <ip_address>10.0.0.1</ip_address>
            </fixed_ip>
        </fixed_ips>
        <id>f71a6703-d6de-4be1-a91a-a570ede1d159</id>
        <security_groups quantum:type="list"/>
        <device_id>9ae135f4-b6e0-4dad-9e91-3c223e385824</device_id>
    </port>
</ports>
```

This operation does not return a response body.

2.6.2. Create port

Method	URI	Description
POST	`/v2.0/ports`	Creates a port on the specified network.

Normal response codes: 200

2.6.2.1. Request

Example 2.31. Create port: JSON request

```
{
    "port": {
        "network_id": "ee2d3158-3e80-4fb3-ba87-c99f515d85e7",
        "admin_state_up": true
    }
}
```

2.6.2.2. Response

Example 2.32. Create port: JSON response

```
{
    "port": {
        "status": "DOWN",
        "binding:host_id": "",
        "name": "private-port",
        "allowed_address_pairs": [],
        "admin_state_up": true,
        "network_id": "a87cc70a-3e15-4acf-8205-9b711a3531b7",
        "tenant_id": "d6700c0c9ffa4f1cb322cd4a1f3906fa",
        "binding:vif_details": {},
        "binding:vnic_type": "normal",
        "binding:vif_type": "unbound",
        "device_owner": "",
        "mac_address": "fa:16:3e:c9:cb:f0",
        "binding:profile": {},
        "fixed_ips": [
            {
                "subnet_id": "a0304c3a-4f08-4c43-88af-d796509c97d2",
                "ip_address": "10.0.0.2"
            }
        ],
        "id": "65c0ee9f-d634-4522-8954-51021b570b0d",
        "security_groups": [
            "f0ac4394-7e4a-4409-9701-ba8be283dbc3"
        ],
        "device_id": ""
    }
}
```

Example 2.33. Create port: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
<port xmlns="http://openstack.org/quantum/api/v2.0"
```

```
    xmlns:binding="http://docs.openstack.org/ext/binding/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <status>DOWN</status>
    <binding:host_id/>
    <name>test_port_1</name>
    <allowed_address_pairs quantum:type="list"/>
    <admin_state_up quantum:type="bool">True</admin_state_up>
    <network_id>a87cc70a-3e15-4acf-8205-9b711a3531b7</network_id>
    <tenant_id>d6700c0c9ffa4f1cb322cd4a1f3906fa</tenant_id>
    <binding:vif_details quantum:type="dict"/>
    <binding:vnic_type>normal</binding:vnic_type>
    <binding:vif_type>unbound</binding:vif_type>
    <device_owner/>
    <mac_address>fa:16:3e:09:e3:47</mac_address>
    <binding:profile quantum:type="dict"/>
    <fixed_ips>
        <fixed_ip>
            <subnet_id>a0304c3a-4f08-4c43-88af-d796509c97d2</subnet_id>
            <ip_address>10.0.0.4</ip_address>
        </fixed_ip>
    </fixed_ips>
    <id>8021790b-4bfd-46ab-bcc7-0ef2f73bff43</id>
    <security_groups>
        <security_group>f0ac4394-7e4a-4409-9701-ba8be283dbc3</security_group>
    </security_groups>
    <device_id/>
</port>
```

This operation does not return a response body.

2.6.3. Show port

Method	URI	Description
GET	`/v2.0/ports/{port_id}`	Shows information for the specified port.

Normal response codes: 200

2.6.3.1. Request

This table shows the URI parameters for the show port request:

Name	Type	Description
`{port_id}`	UUID	The UUID for the port of interest to you.

This operation does not accept a request body.

2.6.3.2. Response

Example 2.34. Show port: JSON response

```
{
    "port": {
        "status": "ACTIVE",
        "binding:host_id": "devstack",
        "name": "",
        "allowed_address_pairs": [],
        "admin_state_up": true,
        "network_id": "a87cc70a-3e15-4acf-8205-9b711a3531b7",
        "tenant_id": "7e02058126cc4950b75f9970368ba177",
        "extra_dhcp_opts": [],
        "binding:vif_details": {
            "port_filter": true,
            "ovs_hybrid_plug": true
        },
        "binding:vif_type": "ovs",
        "device_owner": "network:router_interface",
        "port_security_enabled": false,
        "mac_address": "fa:16:3e:23:fd:d7",
        "binding:profile": {},
        "binding:vnic_type": "normal",
        "fixed_ips": [
            {
                "subnet_id": "a0304c3a-4f08-4c43-88af-d796509c97d2",
                "ip_address": "10.0.0.1"
            }
        ],
        "id": "46d4bfb9-b26e-41f3-bd2e-e6dcc1ccedb2",
        "security_groups": [],
        "device_id": "5e3898d7-11be-483e-9732-b2f5eccd2b2e"
    }
}
```

Example 2.35. Show port: XML response

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<port xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:binding="http://docs.openstack.org/ext/binding/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <status>ACTIVE</status>
    <binding:host_id>devstack</binding:host_id>
    <name/>
    <allowed_address_pairs quantum:type="list"/>
    <admin_state_up quantum:type="bool">True</admin_state_up>
    <network_id>a87cc70a-3e15-4acf-8205-9b711a3531b7</network_id>
    <tenant_id>7e02058126cc4950b75f9970368ba177</tenant_id>
    <extra_dhcp_opts quantum:type="list"/>
    <binding:vif_details>
        <port_filter quantum:type="bool">True</port_filter>
        <ovs_hybrid_plug quantum:type="bool">True</ovs_hybrid_plug>
    </binding:vif_details>
    <binding:vif_type>ovs</binding:vif_type>
    <device_owner>network:router_interface</device_owner>
    <mac_address>fa:16:3e:23:fd:d7</mac_address>
    <binding:profile quantum:type="dict"/>
    <binding:vnic_type>normal</binding:vnic_type>
    <fixed_ips>
        <fixed_ip>
            <subnet_id>a0304c3a-4f08-4c43-88af-d796509c97d2</subnet_id>
            <ip_address>10.0.0.1</ip_address>
        </fixed_ip>
    </fixed_ips>
    <id>46d4bfb9-b26e-41f3-bd2e-e6dcc1ccedb2</id>
    <security_groups quantum:type="list"/>
    <device_id>5e3898d7-11be-483e-9732-b2f5eccd2b2e</device_id>
</port>
```

This operation does not return a response body.

2.6.4. Update port

Method	URI	Description
PUT	`/v2.0/ports/{port_id}`	Updates the specified port.

Normal response codes: 200

2.6.4.1. Request

This table shows the URI parameters for the update port request:

Name	Type	Description
`{port_id}`	UUID	The UUID for the port of interest to you.

Example 2.36. Update port: JSON request

```
{
    "port": {
        "network_id": "ee2d3158-3e80-4fb3-ba87-c99f515d85e7",
        "admin_state_up": true
    }
}
```

2.6.4.2. Response

Example 2.37. Update port: JSON response

```
{
    "port": {
        "status": "DOWN",
        "binding:host_id": "",
        "name": "private-port",
        "allowed_address_pairs": [],
        "admin_state_up": true,
        "network_id": "a87cc70a-3e15-4acf-8205-9b711a3531b7",
        "tenant_id": "d6700c0c9ffa4f1cb322cd4a1f3906fa",
        "binding:vif_details": {},
        "binding:vnic_type": "normal",
        "binding:vif_type": "unbound",
        "device_owner": "",
        "mac_address": "fa:16:3e:c9:cb:f0",
        "binding:profile": {},
        "fixed_ips": [
            {
                "subnet_id": "a0304c3a-4f08-4c43-88af-d796509c97d2",
                "ip_address": "10.0.0.2"
            }
        ],
        "id": "65c0ee9f-d634-4522-8954-51021b570b0d",
        "security_groups": [
            "f0ac4394-7e4a-4409-9701-ba8be283dbc3"
        ],
        "device_id": ""
    }
}
```

Example 2.38. Update port: XML response

```xml
<?xml version='1.0' encoding='UTF-8'?>
<port xmlns="http://openstack.org/quantum/api/v2.0"
    xmlns:binding="http://docs.openstack.org/ext/binding/api/v1.0"
    xmlns:quantum="http://openstack.org/quantum/api/v2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <status>DOWN</status>
    <binding:host_id/>
    <name>test_port_1</name>
    <allowed_address_pairs quantum:type="list"/>
    <admin_state_up quantum:type="bool">True</admin_state_up>
    <network_id>a87cc70a-3e15-4acf-8205-9b711a3531b7</network_id>
    <tenant_id>d6700c0c9ffa4f1cb322cd4a1f3906fa</tenant_id>
    <binding:vif_details quantum:type="dict"/>
    <binding:vnic_type>normal</binding:vnic_type>
    <binding:vif_type>unbound</binding:vif_type>
    <device_owner/>
    <mac_address>fa:16:3e:09:e3:47</mac_address>
    <binding:profile quantum:type="dict"/>
    <fixed_ips>
        <fixed_ip>
            <subnet_id>a0304c3a-4f08-4c43-88af-d796509c97d2</subnet_id>
            <ip_address>10.0.0.4</ip_address>
        </fixed_ip>
    </fixed_ips>
    <id>8021790b-4bfd-46ab-bcc7-0ef2f73bff43</id>
    <security_groups>
        <security_group>f0ac4394-7e4a-4409-9701-ba8be283dbc3</security_group>
    </security_groups>
    <device_id/>
</port>
```

This operation does not return a response body.

2.7. Security groups and rules (security-groups)

Lists, creates, shows information for, and deletes security groups and security group rules.

Method	URI	Description
GET	`/v2.0/security-groups`	Lists OpenStack Networking security groups to which the specified tenant has access.
POST	`/v2.0/security-groups`	Creates an OpenStack Networking security group.
GET	`/v2.0/security-groups/{security_group_id}{?verbose, fields}`	Shows details for a specified security group.
DELETE	`/v2.0/security-groups/{security_group_id}`	Deletes an OpenStack Networking security group.
GET	`/v2.0/security-group-rules`	Lists a summary of all OpenStack Networking security group rules that the specified tenant can access.
POST	`/v2.0/security-group-rules`	Creates an OpenStack Networking security group rule.
GET	`/v2.0/security-group-rules/{rules-security-groups-id}`	Shows detailed information for a specified security group rule.
DELETE	`/v2.0/security-group-rules/{rules-security-groups-id}`	Deletes a specified rule from a OpenStack Networking security group.

2.7.1. List security groups

Method	URI	Description
GET	`/v2.0/security-groups`	Lists OpenStack Networking security groups to which the specified tenant has access.

The list shows the unique ID for and the rules that are associated with each security group.

Normal response codes: 200

Error response codes: unauthorized (401)

2.7.1.1. Request

Example 2.39. List security groups: JSON request

```
GET /v2.0/security-groups
Accept: application/json
```

This operation does not accept a request body.

2.7.1.2. Response

Example 2.40. List security groups: JSON response

```
{
    "security_groups": [
        {
            "description": "default",
            "id": "85cc3048-abc3-43cc-89b3-377341426ac5",
            "name": "default",
            "security_group_rules": [
                {
                    "direction": "egress",
                    "ethertype": "IPv6",
                    "id": "3c0e45ff-adaf-4124-b083-bf390e5482ff",
                    "port_range_max": null,
                    "port_range_min": null,
                    "protocol": null,
                    "remote_group_id": null,
                    "remote_ip_prefix": null,
                    "security_group_id": "85cc3048-
abc3-43cc-89b3-377341426ac5",
                    "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
                },
                {
                    "direction": "egress",
                    "ethertype": "IPv4",
                    "id": "93aa42e5-80db-4581-9391-3a608bd0e448",
                    "port_range_max": null,
                    "port_range_min": null,
                    "protocol": null,
                    "remote_group_id": null,
                    "remote_ip_prefix": null,
                    "security_group_id": "85cc3048-
abc3-43cc-89b3-377341426ac5",
                    "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
                },
                {
                    "direction": "ingress",
                    "ethertype": "IPv6",
                    "id": "c0b09f00-1d49-4e64-a0a7-8a186d928138",
                    "port_range_max": null,
                    "port_range_min": null,
                    "protocol": null,
                    "remote_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
                    "remote_ip_prefix": null,
                    "security_group_id": "85cc3048-
abc3-43cc-89b3-377341426ac5",
                    "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
                },
                {
                    "direction": "ingress",
                    "ethertype": "IPv4",
                    "id": "f7d45c89-008e-4bab-88ad-d6811724c51c",
                    "port_range_max": null,
                    "port_range_min": null,
                    "protocol": null,
                    "remote_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
                    "remote_ip_prefix": null,
                    "security_group_id": "85cc3048-
abc3-43cc-89b3-377341426ac5",
                    "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
                }
            ],
            "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
        }
    ]
}
```

2.7.2. Create security group

Method	URI	Description
POST	`/v2.0/security-groups`	Creates an OpenStack Networking security group.

This operation creates a security group with default security group rules for the IPv4 and IPv6 ether types.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

2.7.2.1. Request

Example 2.41. Create security group: JSON request

```
{
    "security_group": {
        "name": "new-webservers",
        "description": "security group for webservers"
    }
}
```

2.7.2.2. Response

Example 2.42. Create security group: JSON response

```
{
    "security_group": {
        "description": "security group for webservers",
        "id": "2076db17-a522-4506-91de-c6dd8e837028",
        "name": "new-webservers",
        "security_group_rules": [
            {
                "direction": "egress",
                "ethertype": "IPv4",
                "id": "38ce2d8e-e8f1-48bd-83c2-d33cb9f50c3d",
                "port_range_max": null,
                "port_range_min": null,
                "protocol": null,
                "remote_group_id": null,
                "remote_ip_prefix": null,
                "security_group_id": "2076db17-a522-4506-91de-c6dd8e837028",
                "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
            },
            {
                "direction": "egress",
                "ethertype": "IPv6",
                "id": "565b9502-12de-4ffd-91e9-68885cff6ae1",
                "port_range_max": null,
                "port_range_min": null,
                "protocol": null,
                "remote_group_id": null,
                "remote_ip_prefix": null,
                "security_group_id": "2076db17-a522-4506-91de-c6dd8e837028",
                "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
            }
        ],
        "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
    }
}
```

2.7.3. Show security group

Method	URI	Description
GET	`/v2.0/security-groups/{security_group_id}{?verbose, fields}`	Shows details for a specified security group.

This operation returns a response body that contains the description, name, ID, and security group rules associated with the specified security group and tenant ID.

Normal response codes: 200

Error response codes: unauthorized (401), itemNotFound (404)

2.7.3.1. Request

This table shows the URI parameters for the show security group request:

Name	Type	Description
`{security_group_id}`	Uuid	The unique identifier of the security group.

This table shows the query parameters for the show security group request:

Name	Type	Description
`verbose`	Bool *(Optional)*	Show detailed information.
`fields`	String *(Optional)*	The fields to be returned by server.

Example 2.43. Show security group: JSON request

```
GET /v2.0/security-groups/85cc3048-abc3-43cc-89b3-377341426ac5
Accept: application/json
```

This operation does not accept a request body.

2.7.3.2. Response

Example 2.44. Show security group: JSON response

```
{
    "security_group": {
        "description": "default",
        "id": "85cc3048-abc3-43cc-89b3-377341426ac5",
        "name": "default",
        "security_group_rules": [
            {
                "direction": "egress",
                "ethertype": "IPv6",
                "id": "3c0e45ff-adaf-4124-b083-bf390e5482ff",
                "port_range_max": null,
                "port_range_min": null,
                "protocol": null,
                "remote_group_id": null,
                "remote_ip_prefix": null,
                "security_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
                "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
            },
            {
                "direction": "egress",
                "ethertype": "IPv4",
                "id": "93aa42e5-80db-4581-9391-3a608bd0e448",
                "port_range_max": null,
                "port_range_min": null,
                "protocol": null,
                "remote_group_id": null,
                "remote_ip_prefix": null,
                "security_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
                "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
            },
            {
                "direction": "ingress",
                "ethertype": "IPv6",
                "id": "c0b09f00-1d49-4e64-a0a7-8a186d928138",
                "port_range_max": null,
                "port_range_min": null,
                "protocol": null,
                "remote_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
                "remote_ip_prefix": null,
                "security_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
                "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
            },
            {
                "direction": "ingress",
                "ethertype": "IPv4",
                "id": "f7d45c89-008e-4bab-88ad-d6811724c51c",
                "port_range_max": null,
                "port_range_min": null,
                "protocol": null,
                "remote_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
                "remote_ip_prefix": null,
                "security_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
                "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
            }
        ],
        "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
    }
}
```

2.7.4. Delete security group

Method	URI	Description
DELETE	`/v2.0/security-groups/{security_group_id}`	Deletes an OpenStack Networking security group.

This operation deletes an OpenStack Networking security group and its associated security group rules, provided that a port is not associated with the security group.

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

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404)

2.7.4.1. Request

This table shows the URI parameters for the delete security group request:

Name	Type	Description
`{security_group_id}`	Uuid	The unique identifier of the security group.

Example 2.45. Delete security group: JSON request

```
DELETE /v2.0/security-groups/e470bdfc-4869-459b-a561-cb3377efae59
Content-Type: application/json
Accept: application/json
```

This operation does not accept a request body.

2.7.4.2. Response

Example 2.46. Delete security group: JSON response

```
status: 204
```

This operation does not return a response body.

2.7.5. List security group rules

Method	URI	Description
GET	`/v2.0/security-group-rules`	Lists a summary of all OpenStack Networking security group rules that the specified tenant can access.

The list provides the unique ID for each security group rule.

Normal response codes: 200

Error response codes: unauthorized (401)

2.7.5.1. Request

Example 2.47. List security group rules: JSON request

```
GET /v2.0/security-group-rules/
Accept: application/json
```

This operation does not accept a request body.

2.7.5.2. Response

Example 2.48. List security group rules: JSON response

```
{
    "security_group_rules": [
        {
            "direction": "egress",
            "ethertype": "IPv6",
            "id": "3c0e45ff-adaf-4124-b083-bf390e5482ff",
            "port_range_max": null,
            "port_range_min": null,
            "protocol": null,
            "remote_group_id": null,
            "remote_ip_prefix": null,
            "security_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
            "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
        },
        {
            "direction": "egress",
            "ethertype": "IPv4",
            "id": "93aa42e5-80db-4581-9391-3a608bd0e448",
            "port_range_max": null,
            "port_range_min": null,
            "protocol": null,
            "remote_group_id": null,
            "remote_ip_prefix": null,
            "security_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
            "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
        },
        {
            "direction": "ingress",
            "ethertype": "IPv6",
            "id": "c0b09f00-1d49-4e64-a0a7-8a186d928138",
            "port_range_max": null,
            "port_range_min": null,
            "protocol": null,
            "remote_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
            "remote_ip_prefix": null,
            "security_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
            "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
        },
        {
            "direction": "ingress",
            "ethertype": "IPv4",
            "id": "f7d45c89-008e-4bab-88ad-d6811724c51c",
            "port_range_max": null,
            "port_range_min": null,
            "protocol": null,
            "remote_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
            "remote_ip_prefix": null,
            "security_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
            "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
        }
    ]
}
```

2.7.6. Create security group rule

Method	URI	Description
POST	`/v2.0/security-group-rules`	Creates an OpenStack Networking security group rule.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401), itemNotFound (404), buildInProgress (409)

2.7.6.1. Request

Example 2.49. Create security group rule: JSON request

```
{
    "security_group_rule": {
        "direction": "ingress",
        "port_range_min": "80",
        "ethertype": "IPv4",
        "port_range_max": "80",
        "protocol": "tcp",
        "remote_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
        "security_group_id": "a7734e61-b545-452d-a3cd-0189cbd9747a"
    }
}
```

2.7.6.2. Response

Example 2.50. Create security group rule: JSON response

```
{
    "security_group_rule": {
        "direction": "ingress",
        "ethertype": "IPv4",
        "id": "2bc0accf-312e-429a-956e-e4407625eb62",
        "port_range_max": 80,
        "port_range_min": 80,
        "protocol": "tcp",
        "remote_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
        "remote_ip_prefix": null,
        "security_group_id": "a7734e61-b545-452d-a3cd-0189cbd9747a",
        "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
    }
}
```

2.7.7. Show security group rule

Method	URI	Description
GET	`/v2.0/security-group-rules/{rules-security-groups-id}`	Shows detailed information for a specified security group rule.

The response body contains the following information about the security group rule:

Normal response codes: 200

Error response codes: unauthorized (401), itemNotFound (404)

2.7.7.1. Request

This table shows the URI parameters for the show security group rule request:

Name	Type	Description
`{rules-security-groups-id}`	Uuid	The unique identifier of the security group rule.

Example 2.51. Show security group rule: JSON request

```
GET /v2.0/security-group-rules/ 3c0e45ff-adaf-4124-b083-bf390e5482ff
Accept: application/json
```

This operation does not accept a request body.

2.7.7.2. Response

Example 2.52. Show security group rule: JSON response

```
{
    "security_group_rule": {
        "direction": "egress",
        "ethertype": "IPv6",
        "id": "3c0e45ff-adaf-4124-b083-bf390e5482ff",
        "port_range_max": null,
        "port_range_min": null,
        "protocol": null,
        "remote_group_id": null,
        "remote_ip_prefix": null,
        "security_group_id": "85cc3048-abc3-43cc-89b3-377341426ac5",
        "tenant_id": "e4f50856753b4dc6afee5fa6b9b6c550"
    }
}
```

2.7.8. Delete security group rule

Method	URI	Description
DELETE	`/v2.0/security-group-rules/{rules-security-groups-id}`	Deletes a specified rule from a OpenStack Networking security group.

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404)

2.7.8.1. Request

This table shows the URI parameters for the delete security group rule request:

Name	Type	Description
`{rules-security-groups-id}`	Uuid	The unique identifier of the security group rule.

Example 2.53. Delete security group rule: JSON request

```
DELETE /v2.0/security-group-rules/fc3c327a-b5b5-4cd3-9577-52893289ce08
Content-Type: application/json
Accept: application/json
```

This operation does not accept a request body.

2.7.8.2. Response

Example 2.54. Delete security group rule: JSON response

```
status: 204
```

This operation does not return a response body.

2.8. Layer-3 networking

Routes packets between subnets, forwards packets from internal networks to external ones, and accesses instances from external networks through floating IPs.

This extension introduces these resources:

- **router**. A logical entity for forwarding packets across internal subnets and NATting them on external networks through an appropriate external gateway.
- **floatingip**. An external IP address that is mapped to a port that is attached to an internal network.

Method	URI	Description
GET	`/v2.0/routers`	Lists logical routers that are accessible to the tenant who submits the request.
POST	`/v2.0/routers`	Creates a logical router.

Method	URI	Description
GET	`/v2.0/routers/{router_id}`	Shows details for a specified router.
PUT	`/v2.0/routers/{router_id}`	Updates a logical router.
DELETE	`/v2.0/routers/{router_id}`	Deletes a logical router and, if present, its external gateway interface.
PUT	`/v2.0/routers/{router_id}/add_router_interface`	Adds an internal interface to a logical router.
PUT	`/v2.0/routers/{router_id}/remove_router_interface`	Removes an internal interface from a logical router.
GET	`/v2.0/floatingips`	Lists floating IPs that are accessible to the tenant who submits the request.
POST	`/v2.0/floatingips`	Creates a floating IP, and, if you specify port information, associates the floating IP with an internal port.
GET	`/v2.0/floatingips/{floatingip_id}`	Shows details for a specified floating IP.
PUT	`/v2.0/floatingips/{floatingip_id}`	Updates a floating IP and its association with an internal port.
DELETE	`/v2.0/floatingips/{floatingip_id}`	Deletes a floating IP and, if present, its associated port.

2.8.1. List routers

Method	URI	Description
GET	`/v2.0/routers`	Lists logical routers that are accessible to the tenant who submits the request.

Default policy settings return only those routers that are owned by the tenant who submits the request, unless an admin user submits the request.

This example request lists routers in JSON format:

```
GET /v2.0/routers
Accept: application/json
```

Use the `fields` query parameter to control which fields are returned in the response body. Additionally, you can filter results by using query string parameters. For information, see Filtering and Column Selection.

Normal response codes: 200

Error response codes: unauthorized (401)

2.8.1.1. Request

This operation does not accept a request body.

2.8.1.2. Response

Example 2.55. List routers: JSON response

```
{
    "routers": [
        {
            "status": "ACTIVE",
            "external_gateway_info": null,
            "name": "second_routers",
            "admin_state_up": true,
            "tenant_id": "6b96ff0cb17a4b859e1e575d221683d3",
            "routes": [],
            "id": "7177abc4-5ae9-4bb7-b0d4-89e94a4abf3b"
        },
        {
            "status": "ACTIVE",
            "external_gateway_info": {
                "network_id": "3c5bcddd-6af9-4e6b-9c3e-c153e521cab8"
            },
            "name": "router1",
            "admin_state_up": true,
            "tenant_id": "33a40233088643acb66ff6eb0ebea679",
            "routes": [],
            "id": "a9254bdb-2613-4a13-ac4c-adc581fba50d"
        }
    ]
}
```

2.8.2. Create router

Method	URI	Description
POST	`/v2.0/routers`	Creates a logical router.

This operation creates a new logical router. When it is created, a logical router does not have any internal interface; it is not associated to any subnet. You can optionally specify an external gateway for a router at create time. The external gateway for the router must be plugged into an external network. An external network has its extended field `router:external` set to true. To specify an external gateway, the identifier of the external network must be passed in the `external_gateway_info` attribute in the request body, as follows:

```
{
   "router": {
      "external_gateway_info": {
         "network_id": "8ca37218-28ff-41cb-9b10-039601ea7e6b"
      }
   }
}
```

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

2.8.2.1. Request

Example 2.56. Create router: JSON request

```
{
    "router": {
        "name": "another_router",
        "external_gateway_info": {
            "network_id": "8ca37218-28ff-41cb-9b10-039601ea7e6b"
        },
        "admin_state_up": true
    }
}
```

2.8.2.2. Response

Example 2.57. Create router: JSON response

```
{
    "router": {
        "status": "ACTIVE",
        "external_gateway_info": {
            "network_id": "8ca37218-28ff-41cb-9b10-039601ea7e6b"
        },
        "name": "another_router",
        "admin_state_up": true,
        "tenant_id": "6b96ff0cb17a4b859e1e575d221683d3",
        "id": "8604a0de-7f6b-409a-a47c-a1cc7bc77b2e"
    }
}
```

2.8.3. Show router details

Method	URI	Description
GET	`/v2.0/routers/{router_id}`	Shows details for a specified router.

This example request shows details for a router in JSON format:

```
GET /v2.0/routers/{router_id}
Accept: application/json
```

Use the `fields` query parameter to control which fields are returned in the response body. For information, see Filtering and Column Selection.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404)

2.8.3.1. Request

This table shows the URI parameters for the show router details request:

Name	Type	Description
`{router_id}`	UUID	The UUID of the router.

This operation does not accept a request body.

2.8.3.2. Response

Example 2.58. Show router details: JSON response

```
{
    "router": {
        "status": "ACTIVE",
        "external_gateway_info": {
            "network_id": "85d76829-6415-48ff-9c63-5c5ca8c61ac6"
        },
        "name": "router1",
        "admin_state_up": true,
        "tenant_id": "d6554fe62e2f41efbb6e026fad5c1542",
        "routes": [],
        "id": "a07eea83-7710-4860-931b-5fe220fae533"
    }
}
```

2.8.4. Update router

Method	URI	Description
PUT	`/v2.0/routers/{router_id}`	Updates a logical router.

You can update the name, administrative state, and the external gateway. For more information about how to set the external gateway for a router, see the create router operation. This operation does not enable the update of router interfaces. To update a router, use the add router interface and remove router interface operations.

This example updates the external gateway information for a router:

```
PUT /v2.0/routers/{router_id}
Accept: application/json
```

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), itemNotFound (404)

2.8.4.1. Request

This table shows the URI parameters for the update router request:

Name	Type	Description
`{router_id}`	UUID	The UUID of the router.

Example 2.59. Update router: JSON request

```
{
    "router": {
        "external_gateway_info": {
            "network_id": "8ca37218-28ff-41cb-9b10-039601ea7e6b"
        }
    }
}
```

2.8.4.2. Response

Example 2.60. Update router: JSON response

```
{
    "router": {
        "status": "ACTIVE",
        "external_gateway_info": {
            "network_id": "8ca37218-28ff-41cb-9b10-039601ea7e6b"
        },
        "name": "another_router",
        "admin_state_up": true,
        "tenant_id": "6b96ff0cb17a4b859e1e575d221683d3",
        "id": "8604a0de-7f6b-409a-a47c-a1cc7bc77b2e"
    }
}
```

2.8.5. Delete router

Method	URI	Description
DELETE	`/v2.0/routers/{router_id}`	Deletes a logical router and, if present, its external gateway interface.

This operation fails if the router has attached interfaces.

Use the remove router interface operation to remove all router interfaces before you delete the router.

This example deletes a router:

```
DELETE /v2.0/routers/{router_id}
Accept: application/json
```

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404), conflict (409)

2.8.5.1. Request

This table shows the URI parameters for the delete router request:

Name	Type	Description
`{router_id}`	UUID	The UUID of the router.

This operation does not accept a request body.

2.8.6. Add interface to router

Method	URI	Description
PUT	/v2.0/routers/{router_id}/ add_router_interface	Adds an internal interface to a logical router.

Attaches a subnet to an internal router interface.

Specify a subnet ID or port ID in the request body:

- Subnet ID. The gateway IP address for the subnet is used to create the router interface.
- Port ID. The IP address associated with the port is used to create the router interface.

If you specify both IDs, the operation returns a `400 Bad Request` error.

If the port is already used, the operation returns a `409 Conflict` error.

The port ID that is returned by this operation can be either:

- The same ID that is passed in the request body.
- The ID of a port that is created by this operation to attach the specified subnet to the router.

After you run this operation:

- The device ID of this port is set to the router ID.
- The `device_owner` attribute is set to `network:router_interface`.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), itemNotFound (404), conflict (409)

2.8.6.1. Request

This table shows the URI parameters for the add interface to router request:

Name	Type	Description
{router_id}	UUID	The UUID of the router.

Example 2.61. Add interface to router: JSON request

```
{
    "subnet_id": "a2f1f29d-571b-4533-907f-5803ab96ead1"
}
```

2.8.6.2. Response

Example 2.62. Add interface to router: JSON response

```
{
    "subnet_id": "a2f1f29d-571b-4533-907f-5803ab96ead1",
    "tenant_id": "6ba032e4730d42e2ad928f430f5da33e",
    "port_id": "3a44f4e5-1694-493a-a1fb-393881c673a4",
    "id": "b0294d7e-7da4-4202-9882-2ab1de9dabc0"
}
```

2.8.7. Remove interface from router

Method	URI	Description
PUT	`/v2.0/routers/{router_id}/remove_router_interface`	Removes an internal interface from a logical router.

This operation removes an internal router interface, which detaches a subnet from the router. You must specify either a subnet ID or port ID in the request body; this value is used to identify the router interface to remove.

You can also specify both a subnet ID and port ID. If you specify both IDs, the subnet ID must correspond to the subnet ID of the first IP address on the port specified by the port ID. Otherwise, the operation returns a `409 Conflict` error. The response contains information about the affected router and interface.

The operation returns a `404 Not Found` if the router or the subnet and port do not exist or are not visible to you. As a consequence of this operation, the port connecting the router with the subnet is removed from the subnet for the network.

This example removes an interface from a router:

```
PUT /v2.0/routers/{router_id}/remove_router_interface
Accept: application/json
```

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), itemNotFound (404), conflict (409)

2.8.7.1. Request

This table shows the URI parameters for the remove interface from router request:

Name	Type	Description
`{router_id}`	UUID	The UUID of the router.

Example 2.63. Remove interface from router: JSON request

```
{
    "subnet_id": "a2f1f29d-571b-4533-907f-5803ab96ead1"
}
```

2.8.7.2. Response

Example 2.64. Remove interface from router: JSON response

```
{
    "id": "8604a0de-7f6b-409a-a47c-a1cc7bc77b2e",
    "tenant_id": "2f245a7b-796b-4f26-9cf9-9e82d248fda7",
    "port_id": "3a44f4e5-1694-493a-a1fb-393881c673a4",
    "subnet_id": "a2f1f29d-571b-4533-907f-5803ab96ead1"
}
```

2.8.8. List floating IPs

Method	URI	Description
GET	`/v2.0/floatingips`	Lists floating IPs that are accessible to the tenant who submits the request.

Default policy settings return only those floating IPs that are owned by the tenant who submits the request, unless an admin user submits the request.

This example request lists floating IPs in JSON format:

```
GET /v2.0/floatingips
Accept: application/json
```

Use the `fields` query parameter to control which fields are returned in the response body. Additionally, you can filter results by using query string parameters. For information, see Filtering and Column Selection.

Normal response codes: 200

Error response codes: unauthorized (401)

2.8.8.1. Request

This operation does not accept a request body.

2.8.8.2. Response

Example 2.65. List floating IPs: JSON response

```
{
    "floatingips": [
        {
            "router_id": "d23abc8d-2991-4a55-ba98-2aaea84cc72f",
            "tenant_id": "4969c491a3c74ee4af974e6d800c62de",
            "floating_network_id": "376da547-b977-4cfe-9cba-275c80debf57",
            "fixed_ip_address": "10.0.0.3",
            "floating_ip_address": "172.24.4.228",
            "port_id": "ce705c24-c1ef-408a-bda3-7bbd946164ab",
            "id": "2f245a7b-796b-4f26-9cf9-9e82d248fda7",
            "status": "ACTIVE"
        },
        {
            "router_id": null,
            "tenant_id": "4969c491a3c74ee4af974e6d800c62de",
            "floating_network_id": "376da547-b977-4cfe-9cba-275c80debf57",
            "fixed_ip_address": null,
            "floating_ip_address": "172.24.4.227",
            "port_id": null,
            "id": "61cea855-49cb-4846-997d-801b70c71bdd",
            "status": "DOWN"
        }
    ]
}
```

2.8.9. Create floating IP

Method	URI	Description
POST	`/v2.0/floatingips`	Creates a floating IP, and, if you specify port information, associates the floating IP with an internal port.

To associate the floating IP with an internal port, specify the port ID attribute in the request body. If you do not specify a port ID in the request, you can issue a **PUT** request instead of a **POST** request.

Default policy settings enable only administrative users to set floating IP addresses and some non-administrative users might require a floating IP address. If you do not specify a floating IP address in the request, the API automatically allocates one.

By default, this operation associates the floating IP address with a single fixed IP address that is configured on an OpenStack Networking port. If a port has multiple IP addresses, you must specify the `fixed_ip_address` attribute in the request body to associate a specific fixed IP address with the floating IP address.

You can create floating IPs on external networks only.

You must configure an IP address with the internal OpenStack Networking port that is associated with the floating IP address.

Error codes:

- `400` The operation returns this error code for one of these reasons:
  - The specified network is not external, such as `router:external=False`.
  - The specified internal OpenStack Networking port is not associated with the floating IP address.
  - The requested floating IP address does not fall in the subnet range for the external network.
  - The specified fixed IP address is not valid.
- `404` The specified port ID is not valid.
- `409` The operation returns this error code for one of these reasons:
  - The requested floating IP address is already in use.
  - The internal OpenStack Networking port and specified fixed IP address are already associated with another floating IP.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401), conflict (409)

2.8.9.1. Request

Example 2.66. Create floating IP: JSON request

```
{
    "floatingip": {
        "floating_network_id": "376da547-b977-4cfe-9cba-275c80debf57",
        "port_id": "ce705c24-c1ef-408a-bda3-7bbd946164ab"
    }
}
```

2.8.9.2. Response

Example 2.67. Create floating IP: JSON response

```
{
    "floatingip": {
        "fixed_ip_address": "10.0.0.3",
        "floating_ip_address": "172.24.4.228",
        "floating_network_id": "376da547-b977-4cfe-9cba-275c80debf57",
        "id": "2f245a7b-796b-4f26-9cf9-9e82d248fda7",
        "port_id": "ce705c24-c1ef-408a-bda3-7bbd946164ab",
        "router_id": "d23abc8d-2991-4a55-ba98-2aaea84cc72f",
        "status": "ACTIVE",
        "tenant_id": "4969c491a3c74ee4af974e6d800c62de"
    }
}
```

2.8.10. Show floating IP details

Method	URI	Description
GET	`/v2.0/floatingips/{floatingip_id}`	Shows details for a specified floating IP.

Use the `fields` query parameter to control which fields are returned in the response body. For information, see Filtering and Column Selection.

This example request shows details for a floating IP in JSON format. This example also filters the result by the `fixed_ip_address` and `floating_ip_address` fields.

```
GET /v2.0/floatingips/{floatingip_id}?fields=fixed_ip_address&fields=
floating_ip_address
Accept: application/json
```

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404)

2.8.10.1. Request

This table shows the URI parameters for the show floating ip details request:

Name	Type	Description
`{floatingip_id}`	UUID	The UUID of the floating IP.

This operation does not accept a request body.

2.8.10.2. Response

Example 2.68. Show floating IP details: JSON response

```
{
    "floatingip": {
        "floating_network_id": "376da547-b977-4cfe-9cba-275c80debf57",
        "router_id": "d23abc8d-2991-4a55-ba98-2aaea84cc72f",
        "fixed_ip_address": "10.0.0.3",
        "floating_ip_address": "172.24.4.228",
        "tenant_id": "4969c491a3c74ee4af974e6d800c62de",
        "status": "ACTIVE",
        "port_id": "ce705c24-c1ef-408a-bda3-7bbd946164ab",
        "id": "2f245a7b-796b-4f26-9cf9-9e82d248fda7"
    }
}
```

2.8.11. Update floating IP

Method	URI	Description
PUT	`/v2.0/floatingips/{floatingip_id}`	Updates a floating IP and its association with an internal port.

The association process is the same as the process for the create floating IP operation.

To disassociate a floating IP from a port, set the `port_id` attribute to null or omit it from the request body.

This example updates a floating IP:

```
PUT /v2.0/floatingips/{floatingip_id}
Accept: application/json
```

Depending on the request body that you submit, this request associates a port with or disassociates a port from a floating IP.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), itemNotFound (404), conflict (409)

2.8.11.1. Request

This table shows the URI parameters for the update floating ip request:

Name	Type	Description
`{floatingip_id}`	UUID	The UUID of the floating IP.

Example 2.69. Update floating IP (associate port): JSON

```
{
    "floatingip": {
        "port_id": "fc861431-0e6c-4842-a0ed-e2363f9bc3a8"
    }
}
```

Example 2.70. Update floating IP (disassociate port): JSON

```
{
    "floatingip": {
        "port_id": null
    }
}
```

2.8.11.2. Response

Example 2.71. Update floating IP (associate port): JSON

```
{
    "floatingip": {
        "floating_network_id": "376da547-b977-4cfe-9cba-275c80debf57",
        "router_id": "d23abc8d-2991-4a55-ba98-2aaea84cc72f",
        "fixed_ip_address": "10.0.0.4",
        "floating_ip_address": "172.24.4.228",
        "tenant_id": "4969c491a3c74ee4af974e6d800c62de",
        "status": "ACTIVE",
        "port_id": "fc861431-0e6c-4842-a0ed-e2363f9bc3a8",
        "id": "2f245a7b-796b-4f26-9cf9-9e82d248fda7"
    }
}
```

Example 2.72. Update floating IP (disassociate port): JSON

```
{
    "floatingip": {
        "floating_network_id": "376da547-b977-4cfe-9cba-275c80debf57",
        "router_id": "d23abc8d-2991-4a55-ba98-2aaea84cc72f",
        "fixed_ip_address": null,
        "floating_ip_address": "172.24.4.228",
        "tenant_id": "4969c491a3c74ee4af974e6d800c62de",
        "status": "ACTIVE",
        "port_id": null,
        "id": "2f245a7b-796b-4f26-9cf9-9e82d248fda7"
    }
}
```

2.8.12. Delete floating IP

Method	URI	Description
DELETE	`/v2.0/floatingips/{floatingip_id}`	Deletes a floating IP and, if present, its associated port.

This example deletes a floating IP:

```
DELETE /v2.0/floatingips/{floatingip_id}
Accept: application/json
```

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404)

2.8.12.1. Request

This table shows the URI parameters for the delete floating ip request:

Name	Type	Description
`{floatingip_id}`	UUID	The UUID of the floating IP.

This operation does not accept a request body.

2.9. Metering labels and rules

Creates, modifies, and deletes OpenStack Layer3 metering labels and rules.

Method	URI	Description
GET	`/v2.0/metering/metering-labels`	Lists all l3 metering labels that belong to the specified tenant.
POST	`/v2.0/metering/metering-labels`	Creates a l3 metering label.
GET	`/v2.0/metering/metering-labels/{metering_label_id}`	Shows informations for a specified metering label.
DELETE	`/v2.0/metering/metering-labels/{metering_label_id}`	Deletes a l3 metering label.
GET	`/v2.0/metering/metering-label-rules`	Lists a summary of all l3 metering label rules belonging to the specified tenant.
POST	`/v2.0/metering/metering-label-rules`	Creates a l3 metering label rule.
GET	`/v2.0/metering/metering-label-rules/{metering-label-rule-id}`	Shows detailed informations for a specified metering label rule.
DELETE	`/v2.0/metering/metering-label-rules/{metering-label-rule-id}`	Deletes a specified l3 metering label rule.

2.9.1. List metering labels

Method	URI	Description
GET	`/v2.0/metering/metering-labels`	Lists all l3 metering labels that belong to the specified tenant.

The list includes the unique ID for each metering labels.

This operation does not require a request body.

This operation returns a response body.

Normal response codes: 200

Error response codes: unauthorized (401)

2.9.1.1. Request

Example 2.73. List metering labels: JSON request

```
GET /v2.0/metering/metering-labels HTTP/1.1
Host: controlnode:9696
User-Agent: python-neutronclient
Content-Type: application/json
Accept: application/json
X-Auth-Token: c52a1b304fec4ca0ac85dc1741eec6e2
```

This operation does not accept a request body.

2.9.1.2. Response

Example 2.74. List metering labels: JSON response

```
{
    "metering_labels": [
        {
            "tenant_id": "45345b0ee1ea477fac0f541b2cb79cd4",
            "description": "label1 description",
            "name": "label1",
            "id": "a6700594-5b7a-4105-8bfe-723b346ce866"
        },
        {
            "tenant_id": "45345b0ee1ea477fac0f541b2cb79cd4",
            "description": "label2 description",
            "name": "label2",
            "id": "e131d186-b02d-4c0b-83d5-0c0725c4f812"
        }
    ]
}
```

2.9.2. Create metering label

Method	URI	Description
POST	`/v2.0/metering/metering-labels`	Creates a l3 metering label.

This operation requires a request body.

The following table describes the required and optional attributes in the request body:

Table 2.1. Create Metering label rule Request Attributes

Attribute	Required	Description
name	Required	The name of the metering label.
description	Optional	Description for the metering label.

This operation returns a response body, which contains the following informations about the metering label:

- `name`. Name of the metering label.
- `description`. Description of the metering label.
- `tenant_id`. The tenant ID for the specified metering label.
- `id`. The metering label ID

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

2.9.2.1. Request

Example 2.75. Create metering label: JSON request

```
{
    "metering_label": {
        "name": "label1",
        "description": "description of label1"
    }
}
```

2.9.2.2. Response

Example 2.76. Create metering label: JSON response

```
{
    "metering_label": {
        "tenant_id": "45345b0ee1ea477fac0f541b2cb79cd4",
        "description": "description of label1",
        "name": "label1",
        "id": "bc91b832-8465-40a7-a5d8-ba87de442266"
    }
}
```

2.9.3. Show metering label

Method	URI	Description
GET	`/v2.0/metering/metering-labels/{metering_label_id}`	Shows informations for a specified metering label.

This operation does not require a request body.

This operation returns a response body that contains the description, name, ID.

Normal response codes: 200

Error response codes: unauthorized (401), itemNotFound (404)

2.9.3.1. Request

This table shows the URI parameters for the show metering label request:

Name	Type	Description
`{metering_label_id}`	Uuid	The unique identifier of the metering label.

Example 2.77. Show metering label: JSON request

```
GET /v2.0/metering/metering-labels/a6700594-5b7a-4105-8bfe-723b346ce866 HTTP/1.1
Host: controlnode:9696
User-Agent: python-neutronclient
Content-Type: application/json
Accept: application/json
X-Auth-Token: c52a1b304fec4ca0ac85dc1741eec6e2
```

This operation does not accept a request body.

2.9.3.2. Response

Example 2.78. Show metering label: JSON response

```
{
    "metering_label": {
        "tenant_id": "45345b0ee1ea477fac0f541b2cb79cd4",
        "description": "label1 description",
        "name": "label1",
        "id": "a6700594-5b7a-4105-8bfe-723b346ce866"
    }
}
```

2.9.4. Delete metering label

Method	URI	Description
DELETE	`/v2.0/metering/metering-labels/{metering_label_id}`	Deletes a l3 metering label.

This operation deletes a l3 metering label.

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

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404)

2.9.4.1. Request

This table shows the URI parameters for the delete metering label request:

Name	Type	Description
`{metering_label_id}`	Uuid	The unique identifier of the metering label.

Example 2.79. Delete metering label: JSON request

```
DELETE /v2.0/metering/metering-labels/a6700594-5b7a-4105-8bfe-723b346ce866
 HTTP/1.1
Host: controlnode:9696
User-Agent: python-neutronclient
Content-Type: application/json
Accept: application/json
X-Auth-Token: c52a1b304fec4ca0ac85dc1741eec6e2
```

This operation does not accept a request body.

2.9.4.2. Response

Example 2.80. Delete metering label: JSON response

```
status: 204
```

This operation does not return a response body.

2.9.5. List metering label rules

Method	URI	Description
GET	`/v2.0/metering/metering-label-rules`	Lists a summary of all l3 metering label rules belonging to the specified tenant.

The list provides the unique ID for each metering label rule.

This operation does not require a request body. This operation returns a response body.

Normal response codes: 200

Error response codes: unauthorized (401)

2.9.5.1. Request

Example 2.81. List metering label rules: JSON request

```
GET /v2.0/metering/metering-label-rules HTTP/1.1
Host: controlnode:9696
User-Agent: python-neutronclient
Content-Type: application/json
Accept: application/json
X-Auth-Token: c52a1b304fec4ca0ac85dc1741eec6e2
```

This operation does not accept a request body.

2.9.5.2. Response

Example 2.82. List metering label rules: JSON response

```
{
    "metering_label_rules": [
        {
            "remote_ip_prefix": "20.0.0.0/24",
            "direction": "ingress",
            "metering_label_id": "e131d186-b02d-4c0b-83d5-0c0725c4f812",
            "id": "9536641a-7d14-4dc5-afaf-93a973ce0eb8",
            "excluded": false
        },
        {
            "remote_ip_prefix": "10.0.0.0/24",
            "direction": "ingress",
            "metering_label_id": "e131d186-b02d-4c0b-83d5-0c0725c4f812",
            "id": "ffc6fd15-40de-4e7d-b617-34d3f7a93aec",
            "excluded": false
        }
    ]
}
```

2.9.6. Create metering label rule

Method	URI	Description
POST	`/v2.0/metering/metering-label-rules`	Creates a l3 metering label rule.

This operation requires a request body.

The following table describes the required and optional attributes in the request body:

Table 2.2. Create Metering label rule Request Attributes

Attribute	Required	Description
direction	Optional	Ingress or egress: The direction in which metering rule is applied. Default: ingress
metering_label_id	Required	The meteting label ID to associate with this metering rule.
excluded	Optional	Specify whether the remote_ip_prefix will be excluded or not from traffic counters of the metering label, ie: to not count the traffic of a specific IP address of a range. Default: False
remote_ip_prefix	Required	The remote IP prefix to be associated with this metering rule. packet.

This operation returns a response body.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401), itemNotFound (404), buildInProgress (409)

2.9.6.1. Request

Example 2.83. Create metering label rule: JSON request

```
{
    "metering_label_rule": {
        "remote_ip_prefix": "10.0.1.0/24",
        "direction": "ingress",
        "metering_label_id": "e131d186-b02d-4c0b-83d5-0c0725c4f812"
    }
}
```

2.9.6.2. Response

Example 2.84. Create metering label rule: JSON response

```
{
    "metering_label_rule": {
        "remote_ip_prefix": "10.0.1.0/24",
        "direction": "ingress",
        "metering_label_id": "e131d186-b02d-4c0b-83d5-0c0725c4f812",
        "id": "00e13b58-b4f2-4579-9c9c-7ac94615f9ae",
        "excluded": false
    }
}
```

2.9.7. Show metering label rule

Method	URI	Description
GET	`/v2.0/metering/metering-label-rules/{metering-label-rule-id}`	Shows detailed informations for a specified metering label rule.

This operation does not require a request body.

This operation returns a response body, which contains the following informations about the metering label rule:

- `direction`. Either ingress or egress.
- `excluded`. Either True or False.
- The ID for the specified metering label rule
- The remote IP prefix
- The metering label ID for the metering label with which the rule is associated

Normal response codes: 200

Error response codes: unauthorized (401), itemNotFound (404)

2.9.7.1. Request

This table shows the URI parameters for the show metering label rule request:

Name	Type	Description
`{metering-label-rule-id}`	Uuid	The unique identifier of metering label rule.

Example 2.85. Show metering label rule: JSON request

```
GET /v2.0/metering/metering-label-rules/9536641a-7d14-4dc5-afaf-93a973ce0eb8
 HTTP/1.1
Host: controlnode:9696
User-Agent: python-neutronclient
Content-Type: application/json
Accept: application/json
X-Auth-Token: c52a1b304fec4ca0ac85dc1741eec6e2
```

This operation does not accept a request body.

2.9.7.2. Response

Example 2.86. Show metering label rule: JSON response

```
{
    "metering_label_rule": {
        "remote_ip_prefix": "20.0.0.0/24",
        "direction": "ingress",
        "metering_label_id": "e131d186-b02d-4c0b-83d5-0c0725c4f812",
        "id": "9536641a-7d14-4dc5-afaf-93a973ce0eb8",
        "excluded": false
    }
}
```

2.9.8. Delete metering label rule

Method	URI	Description
DELETE	`/v2.0/metering/metering-label-rules/{metering-label-rule-id}`	Deletes a specified l3 metering label rule.

This operation does not require a request body.

This operation does not return a response body.

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404)

2.9.8.1. Request

This table shows the URI parameters for the delete metering label rule request:

Name	Type	Description
`{metering-label-rule-id}`	Uuid	The unique identifier of metering label rule.

Example 2.87. Delete metering label rule: JSON request

```
DELETE /v2.0/metering/metering-labels/37b31179-71ee-4f0a-b130-0eeb28e7ede7
 HTTP/1.1
Host: controlnode:9696
User-Agent: python-neutronclient
Content-Type: application/json
Accept: application/json
X-Auth-Token: c52a1b304fec4ca0ac85dc1741eec6e2
```

This operation does not accept a request body.

2.9.8.2. Response

Example 2.88. Delete metering label rule: JSON response

```
status: 204
```

This operation does not return a response body.

2.10. Load-Balancer-as-a-Service (LBaaS) 1.0 (STABLE)

The LBaaS version 1.0 extension pairs with the Networking 2.0 API to enable OpenStack tenants to manage load balancers for their VMs. With this extension, you can load-balance client traffic from one network to application services, such as VMs, on the same network.

Use this extension to create and manage virtual IP addresses (VIPs), pools, members of a pool, health monitors associated with a pool, and view status of a resource.

Table 2.3. Load balancer statuses

Status	Description
ACTIVE	Resource is ready and active.
PENDING_CREATE	Resource is being created.
PENDING_UPDATE	Resource is being updated.
PENDING_DELETE	Resource is pending deletion.
INACTIVE	Resource was created but is not active.
ERROR	Object within the service is not working. The `error_details` attribute provides an explanation for the error, its cause, and possibly a solution.

Method	URI	Description
GET	`/v2.0/lb/vips`	Lists VIPs.
POST	`/v2.0/lb/vips`	Creates a load balancer VIP.
GET	`/v2.0/lb/vips/{vip_id}`	Shows details for a specified VIP.
PUT	`/v2.0/lb/vips/{vip_id}`	Updates a specified load balancer VIP.
DELETE	`/v2.0/lb/vips/{vip_id}`	Deletes a specified load balancer VIP.
GET	`/v2.0/lb/healthmonitors`	Lists health monitors.
POST	`/v2.0/lb/healthmonitors`	Creates a load balancer health monitor.
GET	`/v2.0/lb/healthmonitors/{health_monitor_id}`	Shows details for a specified health monitor.
PUT	`/v2.0/lb/healthmonitors/{health_monitor_id}`	Updates a specified load balancer health monitor.
DELETE	`/v2.0/lb/healthmonitors/{health_monitor_id}`	Deletes a specified load balancer health monitor.
GET	`/v2.0/lb/pools`	Lists pools.
POST	`/v2.0/lb/pools`	Creates a load balancer pool.
GET	`/v2.0/lb/pools/{pool_id}`	Shows details for a specified pool.
PUT	`/v2.0/lb/pools/{pool_id}`	Updates a specified load balancer pool.
DELETE	`/v2.0/lb/pools/{pool_id}`	Deletes a specified load balancer pool.
POST	`/v2.0/lb/pools/{pool_id}/health_monitors`	Associates a health monitor with a specified pool.
DELETE	`/v2.0/lb/pools/{pool_id}/health_monitors/{health_monitor_id}`	Disassociates a specified health monitor from a pool.
GET	`/v2.0/lb/members`	Lists members.
POST	`/v2.0/lb/members`	Creates a load balancer member.
GET	`/v2.0/lb/members/{member_id}`	Shows details for a specified member.
PUT	`/v2.0/lb/members/{member_id}`	Updates a specified load balancer member.
DELETE	`/v2.0/lb/members/{member_id}`	Deletes a specified load balancer member.

2.10.1. List VIPs

Method	URI	Description
GET	`/v2.0/lb/vips`	Lists VIPs.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403)

2.10.1.1. Request

This operation does not accept a request body.

2.10.1.2. Response

Example 2.89. List VIPs: JSON response

```
{
    "vips": [
        {
            "status": "ACTIVE",
            "protocol": "HTTP",
            "description": "",
            "admin_state_up": true,
            "subnet_id": "8032909d-47a1-4715-90af-5153ffe39861",
            "tenant_id": "83657cfcdfe44cd5920adaf26c48ceea",
            "connection_limit": 1000,
            "pool_id": "72741b06-df4d-4715-b142-276b6bce75ab",
            "session_persistence": {
                "cookie_name": "MyAppCookie",
                "type": "APP_COOKIE"
            },
            "address": "10.0.0.10",
            "protocol_port": 80,
            "port_id": "b5a743d6-056b-468b-862d-fb13a9aa694e",
            "id": "4ec89087-d057-4e2c-911f-60a3b47ee304",
            "name": "my-vip"
        }
    ]
}
```

2.10.2. Create a load balancer VIP

Method	URI	Description
POST	`/v2.0/lb/vips`	Creates a load balancer VIP.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

2.10.2.1. Request

Example 2.90. Create a load balancer VIP: JSON request

```
{
    "vip": {
        "protocol": "HTTP",
        "name": "NewVip",
        "admin_state_up": true,
        "subnet_id": "8032909d-47a1-4715-90af-5153ffe39861",
        "pool_id": "61b1f87a-7a21-4ad3-9dda-7f81d249944f",
        "protocol_port": "80"
    }
}
```

2.10.2.2. Response

Example 2.91. Create a load balancer VIP: JSON response

```
{
    "vip": {
        "status": "PENDING_CREATE",
        "protocol": "HTTP",
        "description": "",
        "admin_state_up": true,
        "subnet_id": "8032909d-47a1-4715-90af-5153ffe39861",
        "tenant_id": "83657cfcdfe44cd5920adaf26c48ceea",
        "connection_limit": -1,
        "pool_id": "61b1f87a-7a21-4ad3-9dda-7f81d249944f",
        "address": "10.0.0.11",
        "protocol_port": 80,
        "port_id": "f7e6fe6a-b8b5-43a8-8215-73456b32e0f5",
        "id": "c987d2be-9a3c-4ac9-a046-e8716b1350e2",
        "name": "NewVip"
    }
}
```

2.10.3. Show VIP details

Method	URI	Description
GET	`/v2.0/lb/vips/{vip_id}`	Shows details for a specified VIP.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404)

2.10.3.1. Request

This table shows the URI parameters for the show vip details request:

Name	Type	Description
`{vip_id}`	UUID	The UUID for the VIP.

This operation does not accept a request body.

2.10.3.2. Response

Example 2.92. Show VIP details: JSON response

```
{
    "vip": {
        "status": "ACTIVE",
        "protocol": "HTTP",
        "description": "",
        "admin_state_up": true,
        "subnet_id": "8032909d-47a1-4715-90af-5153ffe39861",
        "tenant_id": "83657cfcdfe44cd5920adaf26c48ceea",
        "connection_limit": 1000,
        "pool_id": "72741b06-df4d-4715-b142-276b6bce75ab",
        "session_persistence": {
            "cookie_name": "MyAppCookie",
            "type": "APP_COOKIE"
        },
        "address": "10.0.0.10",
        "protocol_port": 80,
        "port_id": "b5a743d6-056b-468b-862d-fb13a9aa694e",
        "id": "4ec89087-d057-4e2c-911f-60a3b47ee304",
        "name": "my-vip"
    }
}
```

2.10.4. Update VIP

Method	URI	Description
PUT	`/v2.0/lb/vips/{vip_id}`	Updates a specified load balancer VIP.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), forbidden (403), itemNotFound (404)

2.10.4.1. Request

This table shows the URI parameters for the update vip request:

Name	Type	Description
`{vip_id}`	UUID	The UUID for the VIP.

Example 2.93. Update VIP: JSON request

```
{
    "vip": {
        "connection_limit": "1000"
    }
}
```

2.10.4.2. Response

Example 2.94. Update VIP: JSON response

```
{
    "vip": {
        "status": "PENDING_UPDATE",
        "protocol": "HTTP",
        "description": "",
        "admin_state_up": true,
        "subnet_id": "8032909d-47a1-4715-90af-5153ffe39861",
        "tenant_id": "83657cfcdfe44cd5920adaf26c48ceea",
        "connection_limit": 1000,
        "pool_id": "61b1f87a-7a21-4ad3-9dda-7f81d249944f",
        "address": "10.0.0.11",
        "protocol_port": 80,
        "port_id": "f7e6fe6a-b8b5-43a8-8215-73456b32e0f5",
        "id": "c987d2be-9a3c-4ac9-a046-e8716b1350e2",
        "name": "NewVip"
    }
}
```

2.10.5. Delete VIP

Method	URI	Description
DELETE	`/v2.0/lb/vips/{vip_id}`	Deletes a specified load balancer VIP.

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404), conflict (409)

2.10.5.1. Request

This table shows the URI parameters for the delete vip request:

Name	Type	Description
`{vip_id}`	UUID	The UUID for the VIP.

This operation does not accept a request body.

2.10.6. List health monitors

Method	URI	Description
GET	`/v2.0/lb/healthmonitors`	Lists health monitors.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403)

2.10.6.1. Request

This operation does not accept a request body.

2.10.6.2. Response

Example 2.95. List health monitors: JSON response

```
{
   "health_monitors":[
      {
         "admin_state_up":true,
         "tenant_id":"83657cfcdfe44cd5920adaf26c48ceea",
         "delay":10,
         "max_retries":1,
         "timeout":1,
         "type":"PING",
         "id":"466c8345-28d8-4f84-a246-e04380b0461d"
      },
      {
         "admin_state_up":true,
         "tenant_id":"83657cfcdfe44cd5920adaf26c48ceea",
         "delay":5,
         "expected_codes":"200",
         "max_retries":2,
         "http_method":"GET",
         "timeout":2,
         "url_path":"/",
         "type":"HTTP",
         "id":"5d4b5228-33b0-4e60-b225-9b727c1a20e7"
      }
   ]
}
```

2.10.7. Create a load balancer health monitor

Method	URI	Description
POST	`/v2.0/lb/healthmonitors`	Creates a load balancer health monitor.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

2.10.7.1. Request

Example 2.96. Create a load balancer health monitor: JSON request

```
{
    "healthmonitor": {
        "admin_state_up": true,
        "delay": "1",
        "expected_codes": "200,201,202",
        "http_method": "GET",
        "max_retries": 5,
        "pool_id": "74aa2010-a59f-4d35-a436-60a6da882819",
        "timeout": 1,
        "type": "HTTP",
        "url_path": "/index.html"
    }
}
```

2.10.7.2. Response

Example 2.97. Create a load balancer health monitor: JSON response

```
{
    "healthmonitor": {
        "admin_state_up": true,
        "delay": 1,
        "expected_codes": "200,201,202",
        "http_method": "GET",
        "id": "0a9ac99d-0a09-4b18-8499-a0796850279a",
        "max_retries": 5,
        "pools": [
            {
                "id": "74aa2010-a59f-4d35-a436-60a6da882819"
            }
        ],
        "tenant_id": "6f3584d5754048a18e30685362b88411",
        "timeout": 1,
        "type": "HTTP",
        "url_path": "/index.html"
    }
}
```

2.10.8. Show health monitor details

Method	URI	Description
GET	`/v2.0/lb/healthmonitors/{health_monitor_id}`	Shows details for a specified health monitor.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404)

2.10.8.1. Request

This operation does not accept a request body.

2.10.8.2. Response

Example 2.98. Show health monitor details: JSON response

```
{
    "healthmonitor": {
        "admin_state_up": true,
        "delay": 1,
        "expected_codes": "200,201,202",
        "http_method": "GET",
        "id": "0a9ac99d-0a09-4b18-8499-a0796850279a",
        "max_retries": 5,
        "pools": [
            {
                "id": "74aa2010-a59f-4d35-a436-60a6da882819"
            }
        ],
        "tenant_id": "6f3584d5754048a18e30685362b88411",
        "timeout": 1,
        "type": "HTTP",
        "url_path": "/index.html"
    }
}
```

2.10.9. Update health monitor

Method	URI	Description
PUT	/v2.0/lb/healthmonitors/{health_monitor_id}	Updates a specified load balancer health monitor.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), forbidden (403), itemNotFound (404)

2.10.9.1. Request

Example 2.99. Update health monitor: JSON request

```
{
   "health_monitor":{
      "delay":"3"
   }
}
```

2.10.9.2. Response

Example 2.100. Update health monitor: JSON response

```
{
    "health_monitor": {
        "admin_state_up": true,
        "tenant_id": "4fd44f30292945e481c7b8a0c8908869",
        "delay": 5,
        "max_retries": 5,
        "http_method": "GET",
        "timeout": 1,
        "pools": [
            {
                "status": "PENDING_CREATE",
                "status_description": null,
                "pool_id": "6e55751f-6ad4-4e53-b8d4-02e442cd21df"
            }
        ],
        "type": "PING",
        "id": "b05e44b5-81f9-4551-b474-711a722698f7"
    }
}
```

2.10.10. Delete health monitor

Method	URI	Description
DELETE	`/v2.0/lb/healthmonitors/{health_monitor_id}`	Deletes a specified load balancer health monitor.

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404), conflict (409)

2.10.11. List pools

Method	URI	Description
GET	`/v2.0/lb/pools`	Lists pools.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403)

2.10.11.1. Request

This operation does not accept a request body.

2.10.11.2. Response

Example 2.101. List pools: JSON response

```
{
   "pools":[
      {
         "status":"ACTIVE",
         "lb_method":"ROUND_ROBIN",
         "protocol":"HTTP",
         "description":"",
         "health_monitors":[
            "466c8345-28d8-4f84-a246-e04380b0461d",
            "5d4b5228-33b0-4e60-b225-9b727c1a20e7"
         ],
         "subnet_id":"8032909d-47a1-4715-90af-5153ffe39861",
         "tenant_id":"83657cfcdfe44cd5920adaf26c48ceea",
         "admin_state_up":true,
         "name":"app_pool",
         "members":[
            "701b531b-111a-4f21-ad85-4795b7b12af6",
            "beb53b4d-230b-4abd-8118-575b8fa006ef"
         ],
         "id":"72741b06-df4d-4715-b142-276b6bce75ab",
         "vip_id":"4ec89087-d057-4e2c-911f-60a3b47ee304"
      }
   ]
}
```

2.10.12. Create a load balancer pool

Method	URI	Description
POST	`/v2.0/lb/pools`	Creates a load balancer pool.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

2.10.12.1. Request

Example 2.102. Create a load balancer pool: JSON request

```
{
    "pool": {
        "admin_state_up": true,
        "description": "simple pool",
        "lb_algorithm": "ROUND_ROBIN",
        "listener_id": "39de4d56-d663-46e5-85a1-5b9d5fa17829",
        "name": "pool1",
        "protocol": "HTTP",
        "session_persistence": {
            "cookie_name": "my_cookie",
            "type": "APP_COOKIE"
        }
    }
}
```

2.10.12.2. Response

Example 2.103. Create a load balancer pool: JSON response

```
{
    "pool": {
        "admin_state_up": true,
        "description": "simple pool",
        "healthmonitor_id": null,
        "id": "12ff63af-4127-4074-a251-bcb2ecc53ebe",
        "lb_algorithm": "ROUND_ROBIN",
        "listeners": [
            {
                "id": "39de4d56-d663-46e5-85a1-5b9d5fa17829"
            }
        ],
        "members": [],
        "name": "pool1",
        "protocol": "HTTP",
        "session_persistence": {
            "cookie_name": "my_cookie",
            "type": "APP_COOKIE"
        },
        "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c"
    }
}
```

2.10.13. Show pool details

Method	URI	Description
GET	`/v2.0/lb/pools/{pool_id}`	Shows details for a specified pool.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404)

2.10.13.1. Request

This table shows the URI parameters for the show pool details request:

Name	Type	Description
`{pool_id}`	UUID	The UUID for the pool.

This operation does not accept a request body.

2.10.13.2. Response

Example 2.104. Show pool details: JSON response

```
{
    "pool": {
        "admin_state_up": true,
        "description": "simple pool",
        "healthmonitor_id": null,
        "id": "4c0a0a5f-cf8f-44b7-b912-957daa8ce5e5",
        "lb_algorithm": "ROUND_ROBIN",
        "listeners": [
            {
                "id": "35cb8516-1173-4035-8dae-0dae3453f37f"
            }
        ],
        "members": [],
        "name": "pool1",
        "protocol": "HTTP",
        "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c"
    }
}
```

2.10.14. Update pool

Method	URI	Description
PUT	`/v2.0/lb/pools/{pool_id}`	Updates a specified load balancer pool.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), forbidden (403), itemNotFound (404)

2.10.14.1. Request

This table shows the URI parameters for the update pool request:

Name	Type	Description
`{pool_id}`	UUID	The UUID for the pool.

Example 2.105. Update pool: JSON request

```
{
   "pool":{
      "name":"SuperPool"
   }
}
```

2.10.14.2. Response

Example 2.106. Update pool: JSON response

```
{
   "pool":{
      "status":"PENDING_UPDATE",
      "lb_method":"ROUND_ROBIN",
      "protocol":"TCP",
      "description":"",
      "health_monitors":[

      ],
      "subnet_id":"8032909d-47a1-4715-90af-5153ffe39861",
      "tenant_id":"83657cfcdfe44cd5920adaf26c48ceea",
      "admin_state_up":true,
      "name":"SuperPool",
      "members":[

      ],
      "id":"61b1f87a-7a21-4ad3-9dda-7f81d249944f",
      "vip_id":null
   }
}
```

2.10.15. Delete pool

Method	URI	Description
DELETE	`/v2.0/lb/pools/{pool_id}`	Deletes a specified load balancer pool.

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404), conflict (409)

2.10.15.1. Request

This table shows the URI parameters for the delete pool request:

Name	Type	Description
`{pool_id}`	UUID	The UUID for the pool.

This operation does not accept a request body.

2.10.16. Associate health monitor with pool

Method	URI	Description
POST	`/v2.0/lb/pools/{pool_id}/health_monitors`	Associates a health monitor with a specified pool.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

2.10.16.1. Request

This table shows the URI parameters for the associate health monitor with pool request:

Name	Type	Description
`{pool_id}`	UUID	The UUID for the pool.

Example 2.107. Associate health monitor with pool: JSON request

```
{
   "health_monitor":{
      "id":"b624decf-d5d3-4c66-9a3d-f047e7786181"
   }
}
```

2.10.16.2. Response

Example 2.108. Associate health monitor with pool: JSON response

```
{
   "health_monitor":{

   }
}
```

2.10.17. Disassociate health monitor from pool

Method	URI	Description
DELETE	`/v2.0/lb/pools/{pool_id}/health_monitors/{health_monitor_id}`	Disassociates a specified health monitor from a pool.

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404), conflict (409)

2.10.17.1. Request

This table shows the URI parameters for the disassociate health monitor from pool request:

Name	Type	Description
`{pool_id}`	UUID	The UUID for the pool.
`{health_monitor_id}`	UUID	The UUID for the health monitor.

This operation does not accept a request body.

2.10.18. List members

Method	URI	Description
GET	`/v2.0/lb/members`	Lists members.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403)

2.10.18.1. Request

This operation does not accept a request body.

2.10.18.2. Response

Example 2.109. List members: JSON response

```
{
   "members":[
      {
         "status":"ACTIVE",
         "weight":1,
         "admin_state_up":true,
         "tenant_id":"83657cfcdfe44cd5920adaf26c48ceea",
         "pool_id":"72741b06-df4d-4715-b142-276b6bce75ab",
         "address":"10.0.0.4",
         "protocol_port":80,
         "id":"701b531b-111a-4f21-ad85-4795b7b12af6"
      },
      {
         "status":"ACTIVE",
         "weight":1,
         "admin_state_up":true,
         "tenant_id":"83657cfcdfe44cd5920adaf26c48ceea",
         "pool_id":"72741b06-df4d-4715-b142-276b6bce75ab",
         "address":"10.0.0.3",
         "protocol_port":80,
         "id":"beb53b4d-230b-4abd-8118-575b8fa006ef"
      }
   ]
}
```

2.10.19. Create a load balancer member

Method	URI	Description
POST	`/v2.0/lb/members`	Creates a load balancer member.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

2.10.19.1. Request

Example 2.110. Create a load balancer member: JSON request

```
{
    "member": {
        "address": "10.0.0.8",
        "admin_state_up": true,
        "protocol_port": "80",
        "subnet_id": "013d3059-87a4-45a5-91e9-d721068ae0b2",
        "weight": "1"
    }
}
```

2.10.19.2. Response

Example 2.111. Create a load balancer member: JSON response

```
{
    "member": {
        "address": "10.0.0.8",
        "admin_state_up": true,
        "id": "9a7aff27-fd41-4ec1-ba4c-3eb92c629313",
        "protocol_port": 80,
        "subnet_id": "013d3059-87a4-45a5-91e9-d721068ae0b2",
        "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c",
        "weight": 1
    }
}
```

2.10.20. Show member details

Method	URI	Description
GET	`/v2.0/lb/members/{member_id}`	Shows details for a specified member.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404)

2.10.20.1. Request

This table shows the URI parameters for the show member details request:

Name	Type	Description
`{member_id}`	UUID	The UUID for the member.

This operation does not accept a request body.

2.10.20.2. Response

Example 2.112. Show member details: JSON response

```
{
    "member": {
        "address": "10.0.0.8",
        "admin_state_up": true,
        "id": "9a7aff27-fd41-4ec1-ba4c-3eb92c629313",
        "protocol_port": 80,
        "subnet_id": "013d3059-87a4-45a5-91e9-d721068ae0b2",
        "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c",
        "weight": 1
    }
}
```

2.10.21. Update member

Method	URI	Description
PUT	`/v2.0/lb/members/{member_id}`	Updates a specified load balancer member.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), forbidden (403), itemNotFound (404)

2.10.21.1. Request

This table shows the URI parameters for the update member request:

Name	Type	Description
`{member_id}`	UUID	The UUID for the member.

Example 2.113. Update member: JSON request

```
{
   "member":{
      "admin_state_up":false
   }
}
```

2.10.21.2. Response

Example 2.114. Update member: JSON response

```
{
   "member":{
      "status":"PENDING_UPDATE",
      "protocol_port":8080,
      "weight":1,
      "admin_state_up":false,
      "tenant_id":"4fd44f30292945e481c7b8a0c8908869",
      "pool_id":"7803631d-f181-4500-b3a2-1b68ba2a75fd",
      "address":"10.0.0.5",
      "status_description":null,
      "id":"48a471ea-64f1-4eb6-9be7-dae6bbe40a0f"
   }
}
```

2.10.22. Delete member

Method	URI	Description
DELETE	`/v2.0/lb/members/{member_id}`	Deletes a specified load balancer member.

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404), conflict (409)

2.10.22.1. Request

This table shows the URI parameters for the delete member request:

Name	Type	Description
`{member_id}`	UUID	The UUID for the member.

This operation does not accept a request body.

2.11. Load-Balancer-as-a-Service (LBaaS) 2.0 (EXPERIMENTAL)

The LBaaS version 2.0 extension pairs with the Networking 2.0 API to enable OpenStack tenants to manage load balancers for their VMs. With this extension you can load-balance client traffic from one network to application services, such as VMs, on the same network.

Use this extension to create and manage load balancers, listeners, pools, members of a pool, and health monitors associated with a pool and view status of a resource.

Table 2.4. Load balancer statuses

Status	Description
ACTIVE	Resource is ready and active.
PENDING_CREATE	Resource is being created.
PENDING_UPDATE	Resource is being updated.
PENDING_DELETE	Resource is pending deletion.
INACTIVE	Resource was created but is not active.
ERROR	Object within the service is not working. The `error_details` attribute provides an explanation for the error, its cause, and possibly a solution.

Method	URI	Description
GET	`/v2.0/lbaas/loadbalancers`	Lists load balancers.
POST	`/v2.0/lbaas/loadbalancers`	Creates a load balancer.
GET	`/v2.0/lbaas/loadbalancers/{loadbalancer_id}`	Shows details for a specified load balancer.
PUT	`/v2.0/lbaas/loadbalancers/{loadbalancer_id}`	Updates a specified load balancer.
DELETE	`/v2.0/lbaas/loadbalancers/{loadbalancer_id}`	Removes a specified load balancer.

Method	URI	Description
GET	`/v2.0/lbaas/listeners`	Lists listeners.
POST	`/v2.0/lbaas/listeners`	Creates a listener.
GET	`/v2.0/lbaas/listeners/{listener_id}`	Shows details for a specified listener.
PUT	`/v2.0/lbaas/listeners/{listener_id}`	Updates a specified listener.
DELETE	`/v2.0/lbaas/listeners/{listener_id}`	Removes a specified listener.
GET	`/v2.0/lbaas/pools`	Lists pools.
POST	`/v2.0/lbaas/pools`	Creates a pool.
GET	`/v2.0/lbaas/pools/{pool_id}`	Shows details for a specified pool.
PUT	`/v2.0/lbaas/pools/{pool_id}`	Updates a specified pool.
DELETE	`/v2.0/lbaas/pools/{pool_id}`	Removes a specified pool.
GET	`/v2.0/lbaas/pools/{pool_id}/members`	Lists members of a specified pool.
POST	`/v2.0/lbaas/pools/{pool_id}/members`	Adds a member to a pool.
GET	`/v2.0/lbaas/pools/{pool_id}/members/{member_id}`	Shows details for a specified pool member.
PUT	`/v2.0/lbaas/pools/{pool_id}/members/{member_id}`	Updates a specified member of a pool.
DELETE	`/v2.0/lbaas/pools/{pool_id}/members/{member_id}`	Removes a member from a pool.
POST	`/v2.0/lbaas/healthmonitors`	Creates a health monitor.
GET	`/v2.0/lbaas/healthmonitors`	Lists health monitors.
GET	`/v2.0/lbaas/healthmonitors/{health_monitor_id}`	Shows details for a specified health monitor.
PUT	`/v2.0/lbaas/healthmonitors/{health_monitor_id}`	Updates a specified health monitor.
DELETE	`/v2.0/lbaas/healthmonitors/{health_monitor_id}`	Removes a specified health monitor.

2.11.1. List load balancers

Method	URI	Description
GET	`/v2.0/lbaas/loadbalancers`	Lists load balancers.

Lists all load balancers that are associated with your tenant account.

This operation does not require a request body.

This operation returns a response body. It returns a (potentially empty) list. Each element in the list is a load balancer that can contain the following attributes:

- `id`
- `tenant_id`
- `name`
- `description`
- `vip_subnet_id`
- `vip_address`
- `admin_state_up`
- `listeners`
- `provisioning_status`
- `operating_status`

Normal response codes: 200

Error response codes: unauthorized (401), Internal-server-error (500), serviceUnavailable (503)

2.11.1.1. Request

This operation does not accept a request body.

2.11.1.2. Response

Example 2.115. List load balancers: JSON response

```
{
    "loadbalancers": [
        {
            "id": "3b98602c-3cfe-4f91-bfa4-c3a11c9e7fe0",
            "name": "Example LB",
            "description": "A very simple example load balancer.",
            "tenant_id": "783b31af-6635-48b2-a807-091d9973e3a9",
            "admin_state_up": true,
            "status": "ACTIVE"
        },
        {
            "id": "c617c538-daa5-4ead-be88-59521d8745a7",
            "name": "Example LB",
            "description": "A very simple example load balancer.",
            "tenant_id": "783b31af-6635-48b2-a807-091d9973e3a9",
            "admin_state_up": true,
            "status": "ACTIVE"
        }
    ]
}
```

2.11.2. Create load balancer

Method	URI	Description
POST	`/v2.0/lbaas/loadbalancers`	Creates a load balancer.

This operation provisions a new load balancer based on the configuration defined in the request object. After the request is validated and progress has started on the provisioning process, a response object is returned. The object contains a unique identifier and the status of provisioning the load balancer.

The `provisioning_status` of the load balancer in the response can have one of the following values: `ACTIVE`, `PENDING_CREATE`, or `ERROR`.

If the status is `PENDING_CREATE`, the caller can view the progress of the provisioning operation by performing a GET on `/lbaas/loadbalancers/loadbalancer_id`. When the status of the load balancer changes to `ACTIVE`, the load balancer was successfully provisioned and is operational for traffic handling.

If the request cannot be fulfilled due to insufficient or invalid data, an HTTP 400 (Bad Request) error response is returned with information about the nature of the failure in the response body. Failures in the validation process are non-recoverable and require the caller to correct the cause of the failure and POST the request again.

You can configure all documented features of the load balancer at creation time by specifying the additional elements or attributes in the request.

Users with an administrative role can create load balancers on behalf of other tenants by specifying a `tenant_id` attribute different than their own.

Example: Create a load balancer

- `tenant_id`: only required if the caller has an administrative role and wants to create a load balancer for another tenant.
- `vip_subnet_id`: The network on which to allocate the load balancer's vip address. A tenant can only create load balancer vips on networks authorized by policy (e.g. her own networks or shared/provider networks).

Some attributes receive default values if not specified in the request:

- `admin_state_up`: The default is true.
- `name`: The default is an empty string.
- `description`: The default is an empty string.

If the request cannot be fulfilled due to insufficient or invalid data, an HTTP 400 (Bad Request) error response is returned with information regarding the nature of the failure in the response body. Failures in the validation process are non-recoverable and require the caller to correct the cause of the failure and POST the request again.

You can configure all documented features of the load balancer at creation time by specifying the additional elements or attributes in the request.

Users with an administrative role can create load balancers on behalf of other tenants by specifying a `tenant_id` attribute that is different than their own.

A user can supply a `vip_address` field if she owns the subnet on which the load balancer's VIP will be created. If a `vip_address` is not specified in the payload, the LBaaS service allocates one from the load balancer VIP's subnet.

Example: Create a load balancer

Normal response codes: 201

Error response codes: unauthorized (401), itemNotFound (404), conflict (409), overLimit (413), Internal-server-error (500), serviceUnavailable (503)

2.11.2.1. Request

Example 2.116. Create load balancer: JSON request

```
{
    "loadbalancer": {
        "name": "loadbalancer1",
        "description": "simple lb",
        "tenant_id": "b7c1a69e88bf4b21a8148f787aef2081",
        "vip_subnet_id": "013d3059-87a4-45a5-91e9-d721068ae0b2",
        "vip_address": "10.0.0.4",
        "admin_state_up": true
    }
}
```

2.11.2.2. Response

Example 2.117. Create load balancer: JSON response

```
{
    "loadbalancer": {
        "admin_state_up": true,
        "description": "simple lb",
        "id": "a36c20d0-18e9-42ce-88fd-82a35977ee8c",
        "listeners": [],
        "name": "loadbalancer1",
        "operating_status": "ONLINE",
        "provisioning_status": "ACTIVE",
        "tenant_id": "b7c1a69e88bf4b21a8148f787aef2081",
        "vip_address": "10.0.0.4",
        "vip_subnet_id": "013d3059-87a4-45a5-91e9-d721068ae0b2"
    }
}
```

2.11.3. Show load balancer details

Method	URI	Description
GET	`/v2.0/lbaas/loadbalancers/{loadbalancer_id}`	Shows details for a specified load balancer.

This operation returns a load balancer object identified by `loadbalancer_id`. If the user is not an administrative user and the load balancer object does not belong to her tenant account, she would receive a 403 (Forbidden) error.

This operation does not require a request body.

This operation returns a response body. On success, the returned element is a load balancer that can contain the following attributes:

- `id`
- `tenant_id`
- `name`
- `description`
- `vip_subnet_id`
- `vip_address`
- `admin_state_up`
- `listeners`
- `provisioning_status`
- `operating_status`

Example: Show load balancer details

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404), overLimit (413), Internal-server-error (500), serviceUnavailable (503)

2.11.3.1. Request

This operation does not accept a request body.

2.11.3.2. Response

Example 2.118. Show load balancer details: JSON response

```
{
   "loadbalancer":{
      "id":"8992a43f-83af-4b49-9afd-c2bfbd82d7d7",
      "name":"Example LB",
      "description":"A very simple example load balancer.",
      "vip_address":"1.2.3.4",
      "vip_subnet_id":"SUBNET_ID",
      "tenant_id":"7725fe12-1c14-4f45-ba8e-44bf01763578",
      "admin_state_up":true,
      "status":"ACTIVE"
   }
}
```

2.11.4. Update load balancer

Method	URI	Description
PUT	`/v2.0/lbaas/loadbalancers/{loadbalancer_id}`	Updates a specified load balancer.

This operation updates the attributes of the specified load balancer. Upon successful validation of the request, the service returns a 202 (Accepted) response code. A caller should check that the load balancer provisioning_status has changed to ACTIVE to confirm that the update has taken effect. If the load balancer `provisioning_status` is `PENDING_UPDATE`, the caller can poll the load balancer object by using a GET operation to wait for the changes to be applied.

The update operation allows the caller to change one or more of the following load balancer attributes:

- `name`
- `description`
- `admin_state_up`

This operation returns the updated load balancer object. The `provisioning_status` of the load balancer in the response can take one of the following values: `ACTIVE`, `PENDING_UPDATE`, or `ERROR`.

Normal response codes: 200

Error response codes: Internal-server-error (500), serviceUnavailable (503), badRequest (400), unauthorized (401), overLimit (413)

2.11.4.1. Request

Example 2.119. Update load balancer: JSON request

```
{
    "loadbalancer": {
        "admin_state_up": false,
        "description": "simple lb2",
        "name": "loadbalancer2"
    }
}
```

2.11.4.2. Response

Example 2.120. Update load balancer: JSON response

```
{
    "loadbalancer": {
        "admin_state_up": false,
        "description": "simple lb2",
        "id": "a36c20d0-18e9-42ce-88fd-82a35977ee8c",
        "listeners": [],
        "name": "loadbalancer2",
        "operating_status": "ONLINE",
        "provisioning_status": "PENDING_UPDATE",
        "tenant_id": "b7c1a69e88bf4b21a8148f787aef2081",
        "vip_address": "10.0.0.4",
        "vip_subnet_id": "013d3059-87a4-45a5-91e9-d721068ae0b2"
    }
}
```

2.11.5. Remove load balancer

Method	URI	Description
DELETE	`/v2.0/lbaas/loadbalancers/{loadbalancer_id}`	Removes a specified load balancer.

This operation removes the specified load balancer and its associated configuration from the tenant account. Any and all configuration data is immediately purged and cannot be recovered.

This operation does not require a request body.

This operation does not return a response body.

Example: Delete a load balancer

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404), conflict (409)

2.11.5.1. Request

This operation does not accept a request body.

2.11.6. List listeners

Method	URI	Description
GET	`/v2.0/lbaas/listeners`	Lists listeners.

This operation lists all listeners associated with your tenant account.

This operation does not require a request body.

This operation returns a response body. It returns a (potentially empty) list. Each element in the list is a listener that can contain the following attributes:

- `id`
- `tenant_id`
- `name`
- `description`
- `protocol`
- `protocol_port`
- `connection_limit`
- `default_pool_id`
- `admin_state_up`
- `loadbalancers`

Example: List listeners

Normal response codes: 200

Error response codes: unauthorized (401), serviceUnavailable (503), Internal-server-error (500)

2.11.6.1. Request

This operation does not accept a request body.

2.11.6.2. Response

Example 2.121. List listeners: JSON response

```
{
    "listeners": [
        {
            "admin_state_up": true,
            "connection_limit": 100,
            "default_pool_id": null,
            "description": "",
            "id": "35cb8516-1173-4035-8dae-0dae3453f37f",
            "loadbalancers": [
                {
                    "id": "a9729389-6147-41a3-ab22-a24aed8692b2"
                }
            ],
            "name": "",
            "protocol": "HTTP",
            "protocol_port": 80,
            "tenant_id": "3e4d8bec50a845fcb09e03a4375c691d"
        }
    ]
}
```

2.11.7. Create listener

Method	URI	Description
POST	`/v2.0/lbaas/listeners`	Creates a listener.

This operation provisions a new listener based on the configuration defined in the request object. After the request is validated and progress has started on the provisioning process, a response object is returned. The object contains a unique identifier.

The caller of this operation must specify these listener attributes, at a minimum:

- `tenant_id`: Only required if the caller has an administrative role and wants to create a listener for another tenant.
- `loadbalancer_id`: The load balancer on which this listener is provisioned. A tenant can only create listeners on load balancers authorized by policy (for example, her own load balancers).
- `description`
- `protocol`: The protocol the front end listens for. Must be TCP, HTTP, or HTTPS.
- `protocol_port`: The port on which the front end listens. Must be an integer in the range from 1 to 65535.

Some attributes receive default values if not specified in the request:

- `admin_state_up`: The default is true.
- `name`: The default is an empty string.
- `description`: The default is an empty string.
- `connection_limit`: The default is -1, indicating an infinite limit.

If the request cannot be fulfilled due to insufficient or invalid data, an HTTP 400 (Bad Request) error response is returned with information regarding the nature of the failure in the response body. Failures in the validation process are non-recoverable and require the caller to correct the cause of the failure and POST the request again.

You can configure all documented features of the listener at creation time by specifying the additional elements or attributes in the request.

Users with an administrative role can create listeners on behalf of other tenants by specifying a `tenant_id` attribute different than their own.

A listener cannot be updated if the load balancer that it is attempting to be attached to does not have a `provisioning_status` of `ACTIVE`.

Example: Create a listener

Normal response codes: 201

Error response codes: itemNotFound (404), unauthorized (401), conflict (409), overLimit (413), Internal-server-error (500), serviceUnavailable (503)

2.11.7.1. Request

Example 2.122. Create listener: JSON request

```
{
    "listener": {
        "admin_state_up": true,
        "connection_limit": 100,
        "description": "listener one",
        "loadbalancer_id": "a36c20d0-18e9-42ce-88fd-82a35977ee8c",
        "name": "listener1",
        "protocol": "HTTP",
        "protocol_port": "80"
    }
}
```

2.11.7.2. Response

Example 2.123. Create listener: JSON response

```
{
    "listener": {
        "admin_state_up": true,
        "connection_limit": 100,
        "default_pool_id": null,
        "description": "listener one",
        "id": "39de4d56-d663-46e5-85a1-5b9d5fa17829",
        "loadbalancers": [
            {
                "id": "a36c20d0-18e9-42ce-88fd-82a35977ee8c"
            }
        ],
        "name": "listener1",
        "protocol": "HTTP",
        "protocol_port": 80,
        "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c"
    }
}
```

2.11.8. Show listener details

Method	URI	Description
GET	`/v2.0/lbaas/listeners/{listener_id}`	Shows details for a specified listener.

This operation returns a listener object identified by `listener_id`. If the user is not an administrative user, and the listener object does not belong to her tenant account, she receives a 403 (Forbidden) error.

This operation does not require a request body.

This operation returns a response body. On success, the returned element is a listener that can contain the following attributes:

- `id`
- `tenant_id`
- `name`
- `description`
- `protocol`
- `protocol_port`
- `connection_limit`
- `default_pool_id`
- `admin_state_up`
- `loadbalancers`

Example: Show listener details

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404), conflict (409), overLimit (413), Internal-server-error (500), serviceUnavailable (503)

2.11.8.1. Request

This operation does not accept a request body.

2.11.8.2. Response

Example 2.124. Show listener details: JSON response

```
{
    "listener": {
        "admin_state_up": true,
        "connection_limit": 100,
        "default_pool_id": null,
        "description": "",
        "id": "35cb8516-1173-4035-8dae-0dae3453f37f",
        "loadbalancers": [
            {
                "id": "a9729389-6147-41a3-ab22-a24aed8692b2"
            }
        ],
        "name": "",
        "protocol": "HTTP",
        "protocol_port": 80,
        "tenant_id": "3e4d8bec50a845fcb09e03a4375c691d"
    }
}
```

2.11.9. Update listener

Method	URI	Description
PUT	`/v2.0/lbaas/listeners/{listener_id}`	Updates a specified listener.

This operation updates the attributes of a specified listener. Upon successful validation of the request, the service returns a 202 (Accepted) response code.

The update operation allows the caller to change one or more of the following listener attributes:

- `name`
- `description`
- `admin_state_up`
- `connection_limit`

Example: Update a listener

Note: You cannot update these listener attributes: `listener_id`, `tenant_id`, `loadbalancer_id`, `loadbalancers`, `default_pool_id`, `protocol`, and `protocol_port`. Attempting to update an imutable attribute results in a 422 (Immutable) fault.

Note: You cannot update a listener if the load balancer to which the listener is attached does not have a `provisioning_status` of `ACTIVE`.

Normal response codes: 200

Error response codes: Internal-server-error (500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

2.11.9.1. Request

Example 2.125. Update listener: JSON request

```
{
    "listener": {
        "admin_state_up": false,
        "connection_limit": 200,
        "description": "listener two",
        "name": "listener2"
    }
}
```

2.11.9.2. Response

Example 2.126. Update listener: JSON response

```
{
    "listener": {
        "admin_state_up": false,
        "connection_limit": 200,
        "default_pool_id": null,
        "description": "listener two",
        "id": "39de4d56-d663-46e5-85a1-5b9d5fa17829",
        "loadbalancers": [
            {
                "id": "a36c20d0-18e9-42ce-88fd-82a35977ee8c"
            }
        ],
        "name": "listener2",
        "protocol": "HTTP",
        "protocol_port": 80,
        "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c"
    }
}
```

2.11.10. Remove listener

Method	URI	Description
DELETE	`/v2.0/lbaas/listeners/{listener_id}`	Removes a specified listener.

This operation removes a specified listener and its associated configuration from the tenant account. Any and all configuration data is immediately purged and is not recoverable.

This operation does not require a request body.

This operation does not return a response body.

You cannot delete a listener if the load balancer to which it is attached does not have a `provisioning_status` of `ACTIVE`.

Example: Delete a listener

Normal response codes: 204

Error response codes: Internal-server-error (500), serviceUnavailable (503), unauthorized (401), badRequest (400), conflict (409), overLimit (413)

2.11.10.1. Request

This operation does not accept a request body.

2.11.11. List pools

Method	URI	Description
GET	`/v2.0/lbaas/pools`	Lists pools.

This operation lists all pools that are associated with your tenant account.

This operation does not require a request body.

This operation returns a response body. It returns a (potentially empty) list. Each element in the list is a pool that can contain the following attributes:

- `id`
- `tenant_id`
- `name`
- `description`
- `protocol`
- `lb_algorithm`
- `session_persistence`
- `admin_state_up`
- `listeners`
- `members`
- `healthmonitor_id`

Example: List pools

Normal response codes: 200

Error response codes: unauthorized (401), Internal-server-error (500), serviceUnavailable (503)

2.11.11.1. Request

This operation does not accept a request body.

2.11.11.2. Response

Example 2.127. List pools: JSON response

```
{
    "pools": [
        {
            "admin_state_up": true,
            "description": "simple pool",
            "healthmonitor_id": null,
            "id": "4c0a0a5f-cf8f-44b7-b912-957daa8ce5e5",
            "lb_algorithm": "ROUND_ROBIN",
            "listeners": [
                {
                    "id": "35cb8516-1173-4035-8dae-0dae3453f37f"
                }
            ],
            "members": [],
            "name": "pool1",
            "protocol": "HTTP",
            "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c"
        }
    ]
}
```

2.11.12. Create pool

Method	URI	Description
POST	`/v2.0/lbaas/pools`	Creates a pool.

This operation provisions a new pool based on the configuration defined in the request object. After the request is validated and progress has started on the provisioning process, a response object is returned. The object contains a unique identifier.

The caller of this operation must specify these pool attributes, at a minimum:

- `tenant_id`: Only required if the caller has an administrative role and wants to create a pool for another tenant.
- `protocol`: The protocol this pool and its members listen for. Must be one of TCP, HTTP, or HTTPS
- `lb_algorithm`: The load balancing algorithm to distribute traffic to the pool's members. Must be one of ROUND_ROBIN, LEAST_CONNECTIONS, or SOURCE_IP.
- `protocol_port`: The port on which the front end listens. Must be an integer in the range from 1 to 65535.
- `listener_id`: The listener in which this pool becomes the default pool. There can only be on default pool for a listener.

Some attributes receive default values if not specified in the request:

- `admin_state_up`: The default is true.
- `name`: The default is an empty string.
- `description`: The default is an empty string.
- `session_persistence`: The default is an empty dictionary.

If the request cannot be fulfilled due to insufficient or invalid data, an HTTP 400 (Bad Request) error response is returned with information about the nature of the failure in the response body. Failures in the validation process are non-recoverable and require the caller to correct the cause of the failure and POST the request again.

Users can configure all documented features at creation time by providing the additional elements or attributes in the request.

Users with an administrative role can create pools on behalf of other tenants by specifying a `tenant_id` attribute that is different than their own.

You cannot update a pool if the load balancer to which it is attempting to be attached does not have a `provisioning_status` of `ACTIVE`.

Example: Create a pool

Normal response codes: 201

Error response codes: unauthorized (401), itemNotFound (404), conflict (409), overLimit (413), Internal-server-error (500), serviceUnavailable (503)

2.11.12.1. Request

Example 2.128. Create pool: JSON request

```
{
    "pool": {
        "admin_state_up": true,
        "description": "simple pool",
        "lb_algorithm": "ROUND_ROBIN",
        "listener_id": "39de4d56-d663-46e5-85a1-5b9d5fa17829",
        "name": "pool1",
        "protocol": "HTTP",
        "session_persistence": {
            "cookie_name": "my_cookie",
            "type": "APP_COOKIE"
        }
    }
}
```

2.11.12.2. Response

Example 2.129. Create pool: JSON response

```
{
    "pool": {
        "admin_state_up": true,
        "description": "simple pool",
        "healthmonitor_id": null,
        "id": "12ff63af-4127-4074-a251-bcb2ecc53ebe",
        "lb_algorithm": "ROUND_ROBIN",
        "listeners": [
            {
                "id": "39de4d56-d663-46e5-85a1-5b9d5fa17829"
            }
        ],
        "members": [],
        "name": "pool1",
        "protocol": "HTTP",
        "session_persistence": {
            "cookie_name": "my_cookie",
            "type": "APP_COOKIE"
        },
        "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c"
    }
}
```

2.11.13. Show pool details

Method	URI	Description
GET	`/v2.0/lbaas/pools/{pool_id}`	Shows details for a specified pool.

This operation returns a pool object identified by `pool_id`. If the user is not an administrative user and the pool object does not belong to her tenant account, she receives a 403 (Forbidden) error.

This operation does not require a request body.

This operation returns a response body. On success, the returned element is a pool that can contain the following attributes:

- `id`
- `tenant_id`
- `name`
- `description`
- `protocol`
- `lb_algorithm`
- `session_persistence`
- `admin_state_up`
- `listeners`
- `members`
- `healthmonitor_id`

Example: Show pool details

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404), conflict (409), overLimit (413), Internal-server-error (500), serviceUnavailable (503)

2.11.13.1. Request

This operation does not accept a request body.

2.11.13.2. Response

Example 2.130. Show pool details: JSON response

```
{
    "pool": {
        "admin_state_up": true,
        "description": "simple pool",
        "healthmonitor_id": null,
        "id": "4c0a0a5f-cf8f-44b7-b912-957daa8ce5e5",
        "lb_algorithm": "ROUND_ROBIN",
        "listeners": [
            {
                "id": "35cb8516-1173-4035-8dae-0dae3453f37f"
            }
        ],
        "members": [],
        "name": "pool1",
        "protocol": "HTTP",
        "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c"
    }
}
```

2.11.14. Update pool

Method	URI	Description
PUT	/v2.0/lbaas/pools/{pool_id}	Updates a specified pool.

This operation updates the attributes of the specified pool. Upon successful validation of the request, the service returns a 202 (Accepted) response code.

The update operation allows the caller to change one or more of the following pool attributes:

- name
- description
- admin_state_up
- lb_algorithm
- session_persistence

Note: You cannot update these attributes: pool ID, tenant_id, listener_id, listeners, healthmonitor_id, protocol, and members are immutable attributes. If you try to update any of these attributes, a 422 (Immutable) fault is returned.

Note: You cannot update a pool if the load balancer to which it is attached does not have a provisioning_status of ACTIVE.

Example: Update a pool

Normal response codes: 200

Error response codes: Internal-server-error (500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

2.11.14.1. Request

Example 2.131. Update pool: JSON request

```
{
    "pool": {
        "admin_state_up": false,
        "description": "pool two",
        "lb_algorithm": "LEAST_CONNECTIONS",
        "name": "pool2",
        "session_persistence": {
            "type": "HTTP_COOKIE"
        }
    }
}
```

2.11.14.2. Response

Example 2.132. Update pool: JSON response

```
{
    "pool": {
        "admin_state_up": false,
        "description": "pool two",
        "healthmonitor_id": null,
        "id": "12ff63af-4127-4074-a251-bcb2ecc53ebe",
        "lb_algorithm": "LEAST_CONNECTIONS",
        "listeners": [
            {
                "id": "39de4d56-d663-46e5-85a1-5b9d5fa17829"
            }
        ],
        "members": [],
        "name": "pool2",
        "protocol": "HTTP",
        "session_persistence": {
            "cookie_name": null,
            "type": "HTTP_COOKIE"
        },
        "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c"
    }
}
```

2.11.15. Remove pool

Method	URI	Description
DELETE	`/v2.0/lbaas/pools/{pool_id}`	Removes a specified pool.

This operation removes a specified pool and its associated configuration from the tenant account. Any and all configuration data is immediately purged and is not recoverable.

This operation does not require a request body.

This operation does not return a response body.

You cannot delete a pool if the load balancer to which it is attached does not have a `provisioning_status` of `ACTIVE`.

Example: Delete a pool

Normal response codes: 204

Error response codes: Internal-server-error (500), serviceUnavailable (503), unauthorized (401), badRequest (400), conflict (409), overLimit (413)

2.11.15.1. Request

This operation does not accept a request body.

2.11.16. List pool members

Method	URI	Description
GET	`/v2.0/lbaas/pools/{pool_id}/members`	Lists members of a specified pool.

Lists all members that are associated with a pool that is associated with your tenant account. The list of members includes only members that belong to the pool object identified by `pool_id`.

This operation does not require a request body.

This operation returns a response body. It returns a (potentially empty) list. Each element in the list is a member that can contain the following attributes:

- `id`
- `tenant_id`
- `address`
- `protocol_port`
- `weight`
- `subnet_id`
- `admin_state_up`

Example: List pool members

Normal response codes: 200

Error response codes: unauthorized (401), serviceUnavailable (503), Internal-server-error (500)

2.11.16.1. Request

This operation does not accept a request body.

2.11.16.2. Response

Example 2.133. List pool members: JSON response

```
{
    "members": [
        {
            "address": "10.0.0.8",
            "admin_state_up": true,
            "id": "9a7aff27-fd41-4ec1-ba4c-3eb92c629313",
            "protocol_port": 80,
            "subnet_id": "013d3059-87a4-45a5-91e9-d721068ae0b2",
            "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c",
            "weight": 1
        }
    ]
}
```

2.11.17. Add member to pool

Method	URI	Description
POST	`/v2.0/lbaas/pools/{pool_id}/members`	Adds a member to a pool.

This operation provisions a new member and adds it to a pool based on the configuration defined in the request object. After the request is validated and progress has started on the provisioning process, a response object is returned. The object contains a unique identifier.

The caller of this operation must specify the following pool attributes, at a minimum:

- `tenant_id`: Only required if the caller has an administrative role and wants to create a pool for another tenant.
- `address`: The IP Address of the member to receive traffic from the load balancer.
- `protocol_port` The port that the member is listening to receive traffic.

Some attributes receive default values if not specified in the request:

- `admin_state_up`: The default is true.
- `weight`: The default is 1.

If you omit the `subnet_id` parameter, LBaaS uses the `vip_subnet_id` parameter value for the subnet ID.

If the request fails due to incorrect data, the response returns an HTTP 400 (Bad Request) error with information about reason for the failure. Validation errors require that you correct the error and submit the request again.

To configure all documented member features at creation time, specify additional elements or attributes in the request.

Users with an administrative role can create members on behalf of other tenants by specifying a `tenant_id` attribute that is different than their own.

To update a member, the load balancer must have a `provisioning_status` of ACTIVE.

Normal response codes: 201

Error response codes: unauthorized (401), itemNotFound (404), conflict (409), overLimit (413), Internal-server-error (500), serviceUnavailable (503)

2.11.17.1. Request

Example 2.134. Add member to a pool: JSON request

```
{
    "member": {
        "address": "10.0.0.8",
        "admin_state_up": true,
        "protocol_port": "80",
        "subnet_id": "013d3059-87a4-45a5-91e9-d721068ae0b2",
        "weight": "1"
    }
}
```

2.11.17.2. Response

Example 2.135. Add member to pool: JSON response

```
{
    "member": {
        "address": "10.0.0.8",
        "admin_state_up": true,
        "id": "9a7aff27-fd41-4ec1-ba4c-3eb92c629313",
        "protocol_port": 80,
        "subnet_id": "013d3059-87a4-45a5-91e9-d721068ae0b2",
        "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c",
        "weight": 1
    }
}
```

2.11.18. Show pool member details

Method	URI	Description
GET	/v2.0/lbaas/pools/{pool_id}/members/{member_id}	Shows details for a specified pool member.

This operation returns a member object identified by `member_id` that belongs to a pool object identified by `pool_id`. If the user is not an administrative user and the pool or member object does not belong to her tenant account, she receives a 403 (Forbidden) error.

This operation does not require a request body.

This operation returns a response body. On success, the returned element is a pool that can contain the following attributes:

- `id`
- `tenant_id`
- `address`
- `protocol_port`
- `weight`
- `subnet_id`
- `admin_state_up`

Example: Show pool member details

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404), conflict (409), overLimit (413), Internal-server-error (500), serviceUnavailable (503)

2.11.18.1. Request

This operation does not accept a request body.

2.11.18.2. Response

Example 2.136. Show pool member details: JSON response

```
{
    "member": {
        "address": "10.0.0.8",
        "admin_state_up": true,
        "id": "9a7aff27-fd41-4ec1-ba4c-3eb92c629313",
        "protocol_port": 80,
        "subnet_id": "013d3059-87a4-45a5-91e9-d721068ae0b2",
        "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c",
        "weight": 1
    }
}
```

2.11.19. Update pool member

Method	URI	Description
PUT	/v2.0/lbaas/pools/{pool_id}/members/{member_id}	Updates a specified member of a pool.

This operation updates the attributes of the specified pool. Upon successful validation of the request, the service returns a 200 (OK) response code.

The update operation allows the caller to change one or more of the following pool attributes:

- weight
- admin_state_up

Note: You cannot update these attributes: The member ID, tenant_id, address, protocol_port, and subnet_id. If you attempt to update any of these attributes, a 422 (Immutable) fault is returned.

Note: You cannot update a member if the attached load balancer does not have a provisioning_status of ACTIVE.

Normal response codes: 200

Error response codes: Internal-server-error (500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

2.11.19.1. Request

Example 2.137. Update pool member: JSON request

```
{
    "member": {
        "admin_state_up": false,
        "weight": 5
    }
}
```

2.11.19.2. Response

Example 2.138. Update pool member: JSON response

```
{
    "member": {
        "address": "10.0.0.8",
        "admin_state_up": false,
        "id": "9a7aff27-fd41-4ec1-ba4c-3eb92c629313",
        "protocol_port": 80,
        "subnet_id": "013d3059-87a4-45a5-91e9-d721068ae0b2",
        "tenant_id": "1a3e005cf9ce40308c900bcb08e5320c",
        "weight": 5
    }
}
```

2.11.20. Remove member from pool

Method	URI	Description
DELETE	`/v2.0/lbaas/pools/{pool_id}/members/{member_id}`	Removes a member from a pool.

This operation removes the specified member and its associated configuration from the tenant account. Any and all configuration data is immediately purged and is not recoverable.

This operation does not require a request body.

This operation does not return a response body.

A member cannot be deleted if the attached load balancer does not have a `provisioning_status` of `ACTIVE`.

Example: Remove a member from a pool

Normal response codes: 204

Error response codes: Internal-server-error (500), serviceUnavailable (503), unauthorized (401), badRequest (400), conflict (409), overLimit (413)

2.11.20.1. Request

This operation does not accept a request body.

2.11.21. Create health monitor

Method	URI	Description
POST	`/v2.0/lbaas/healthmonitors`	Creates a health monitor.

This operation provisions a new health monitor based on the configuration defined in the request object. After the request is validated and progress has started on the provisioning process, a response object is returned. The object contains a unique identifier.

The caller of this operation must specify these health monitor attributes, at a minimum:

- `tenant_id`: Only required if the caller has an administrative role and wants to create a health monitor for another tenant.
- `type`: The type of health monitor. Must be one of TCP, HTTP, HTTPS
- `delay`: The interval in seconds between health checks.
- `timeout`: The time in seconds that a health check times out.
- `max_retries`: Number of failed health checks before marked as OFFLINE.
- `pool_id`: The pool that this health monitor will monitor.

Some attributes will receive default values if not specified in the request and are only useful when health monitor type of HTTP(S) is specified:

- `http_method`: The default is GET.
- `url_path`: The default is `/`.
- `expected_codes`: The expected http status codes to get from a successful health check. Default is 200.
- `admin_state_up`: The default is true.

If the request cannot be fulfilled due to insufficient or invalid data, an HTTP 400 (Bad Request) error response will be returned with information regarding the nature of the failure in the response body. Failures in the validation process are non-recoverable and require the caller to correct the cause of the failure and POST the request again.

You can configure all documented features of the health monitor at creation time by specifying the additional elements or attributes in the request.

Users with an administrative role can create health monitors on behalf of other tenants by specifying a tenant_id attribute different than their own.

A health monitor cannot be update if the load balancer it is attempting to be attached to is not in an ACTIVE provisioning_status.

Example: Create a health monitor

Normal response codes: 201

Error response codes: unauthorized (401), itemNotFound (404), conflict (409), overLimit (413), Internal-server-error (500), serviceUnavailable (503)

2.11.21.1. Request

Example 2.139. Create health monitor: JSON request

```
{
    "healthmonitor": {
        "admin_state_up": true,
        "delay": "1",
        "expected_codes": "200,201,202",
        "http_method": "GET",
        "max_retries": 5,
        "pool_id": "74aa2010-a59f-4d35-a436-60a6da882819",
        "timeout": 1,
        "type": "HTTP",
        "url_path": "/index.html"
    }
}
```

2.11.21.2. Response

Example 2.140. Create health monitor: JSON response

```
{
    "healthmonitor": {
        "admin_state_up": true,
        "delay": 1,
        "expected_codes": "200,201,202",
        "http_method": "GET",
        "id": "0a9ac99d-0a09-4b18-8499-a0796850279a",
        "max_retries": 5,
        "pools": [
            {
                "id": "74aa2010-a59f-4d35-a436-60a6da882819"
            }
        ],
        "tenant_id": "6f3584d5754048a18e30685362b88411",
        "timeout": 1,
        "type": "HTTP",
        "url_path": "/index.html"
    }
}
```

2.11.22. List health monitors

Method	URI	Description
GET	`/v2.0/lbaas/healthmonitors`	Lists health monitors.

This operation lists all health monitors associated with your tenant account.

This operation does not require a request body.

This operation returns a response body. It returns a (potentially empty) list, each element in the list is a health monitor that can contain the following attributes:

- `id`
- `tenant_id`
- `type`
- `delay`
- `timeout`
- `max_retries`
- `http_method`
- `url_path`
- `expected_codes`
- `admin_state_up`
- `pool_id`
- `pools`

Example: List health monitors

Normal response codes: 200

Error response codes: unauthorized (401), Internal-server-error (500), serviceUnavailable (503)

2.11.22.1. Request

This operation does not accept a request body.

2.11.22.2. Response

Example 2.141. List health monitors: JSON response

```
{
    "healthmonitors": [
        {
            "admin_state_up": true,
            "delay": 1,
            "expected_codes": "200,201,202",
            "http_method": "GET",
            "id": "0a9ac99d-0a09-4b18-8499-a0796850279a",
            "max_retries": 5,
            "pools": [
                {
                    "id": "74aa2010-a59f-4d35-a436-60a6da882819"
                }
            ],
            "tenant_id": "6f3584d5754048a18e30685362b88411",
            "timeout": 1,
            "type": "HTTP",
            "url_path": "/index.html"
        }
    ]
}
```

2.11.23. Show health monitor details

Method	URI	Description
GET	`/v2.0/lbaas/healthmonitors/{health_monitor_id}`	Shows details for a specified health monitor.

This operation returns a health monitor object identified by healthmonitor_id. If the user is not an admin, and the health monitor object doesn't belong to her tenant account, she would receive a 403 (Forbidden) error.

This operation does not require a request body.

This operation returns a response body. On success, the returned element is a health monitor that can contain the following attributes:

- `id`
- `tenant_id`
- `type`
- `delay`
- `timeout`
- `max_retries`
- `http_method`
- `url_path`
- `expected_codes`
- `admin_state_up`
- `pool_id`
- `pools`

Example: Show health monitor details

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404), conflict (409), overLimit (413), Internal-server-error (500), serviceUnavailable (503)

2.11.23.1. Request

This operation does not accept a request body.

2.11.23.2. Response

Example 2.142. Show health monitor details: JSON response

```
{
    "healthmonitor": {
        "admin_state_up": true,
        "delay": 1,
        "expected_codes": "200,201,202",
        "http_method": "GET",
        "id": "0a9ac99d-0a09-4b18-8499-a0796850279a",
        "max_retries": 5,
        "pools": [
            {
                "id": "74aa2010-a59f-4d35-a436-60a6da882819"
            }
        ],
        "tenant_id": "6f3584d5754048a18e30685362b88411",
        "timeout": 1,
        "type": "HTTP",
        "url_path": "/index.html"
    }
}
```

2.11.24. Update health monitor

Method	URI	Description
PUT	`/v2.0/lbaas/healthmonitors/{health_monitor_id}`	Updates a specified health monitor.

This operation updates the attributes of the specified health monitor. Upon successful validation of the request, the service returns the 202 (Accepted) response code.

The update operation enables you to change one or more health monitor attributes:

- `delay`
- `timeout`
- `max_retries`
- `http_method`
- `url_path`
- `expected_codes`
- `admin_state_up`

Note: The health monitor's ID, tenant_id, pool_id, and type are immutable attributes and cannot be updated. Supplying an unsupported attribute results in a 422 (Immutable) fault.

Normal response codes: 200

Error response codes: Internal-server-error (500), serviceUnavailable (503), unauthorized (401), badRequest (400), overLimit (413)

2.11.24.1. Request

Example 2.143. Update health monitor: JSON request

```
{
    "healthmonitor": {
        "admin_state_up": false,
        "delay": "2",
        "expected_codes": "200",
        "http_method": "POST",
        "max_retries": 2,
        "timeout": 2,
        "url_path": "/page.html"
    }
}
```

2.11.24.2. Response

Example 2.144. Update health monitor: JSON response

```
{
    "healthmonitor": {
        "admin_state_up": false,
        "delay": 2,
        "expected_codes": "200",
        "http_method": "POST",
        "id": "0a9ac99d-0a09-4b18-8499-a0796850279a",
        "max_retries": 2,
        "pools": [
            {
                "id": "74aa2010-a59f-4d35-a436-60a6da882819"
            }
        ],
        "tenant_id": "6f3584d5754048a18e30685362b88411",
        "timeout": 2,
        "type": "HTTP",
        "url_path": "/page.html"
    }
}
```

2.11.25. Remove health monitor

Method	URI	Description
DELETE	`/v2.0/lbaas/healthmonitors/{health_monitor_id}`	Removes a specified health monitor.

This operation removes the specified health monitor and its associated configuration from the tenant account. Any and all configuration data is immediately purged and is not recoverable.

This operation does not require a request body.

This operation does not return a response body.

You cannot delete a health monitor if the attached load balancer does not have a `provisioning_status` of ACTIVE.

Example: Delete a health monitor

Normal response codes: 204

Error response codes: Internal-server-error (500), serviceUnavailable (503), unauthorized (401), badRequest (400), conflict (409), overLimit (413)

2.11.25.1. Request

This operation does not accept a request body.

2.12. Virtual-Private-Network-as-a-Service (VPNaaS) 2.0 (CURRENT)

The VPNaaS extension enables OpenStack tenants to extend private networks across the public telecommunication infrastructure.

This initial implementation of the VPNaaS extension provides:

- Site-to-site VPN that connects two private networks.
- Multiple VPN connections per tenant.
- IKEv1 policy support with 3des, aes-128, aes-256, or aes-192 encryption.
- IPSec policy support with 3des, aes-128, aes-192, or aes-256 encryption, sha1 authentication, ESP, AH, or AH-ESP transform protocol, and tunnel or transport mode encapsulation.
- Dead Peer Detection (DPD) with hold, clear, restart, disabled, or restart-by-peer actions.

This extension introduces these resources:

- `service`. A parent object that associates VPN with a specific subnet and router.

- `ikepolicy`. The Internet Key Exchange (IKE) policy that identifies the authentication and encryption algorithm to use during phase one and two negotiation of a VPN connection.
- `ipsecpolicy`. The IP security policy that specifies the authentication and encryption algorithm and encapsulation mode to use for the established VPN connection.
- `ipsec-site-connection`. Details for the site-to-site IPsec connection, including the peer CIDRs, MTU, authentication mode, peer address, DPD settings, and status.

Method	URI	Description
GET	`/v2.0/vpn/vpnservices`	Lists VPN services.
POST	`/v2.0/vpn/vpnservices`	Creates a VPN service.
GET	`/v2.0/vpn/vpnservices/{service_id}`	Shows details for a specified VPN service.
PUT	`/v2.0/vpn/vpnservices/{service_id}`	Updates a specified VPN service.
DELETE	`/v2.0/vpn/vpnservices/{service_id}`	Removes a specified VPN service.
GET	`/v2.0/vpn/ikepolicies`	Lists IKE policies.
POST	`/v2.0/vpn/ikepolicies`	Creates an IKE policy.
GET	`/v2.0/vpn/ikepolicies/{ikepolicy_id}`	Shows details for a specified IKE policy.
PUT	`/v2.0/vpn/ikepolicies/{ikepolicy_id}`	Updates policy settings in a specified IKE policy.
DELETE	`/v2.0/vpn/ikepolicies/{ikepolicy_id}`	Removes a specified IKE policy.
GET	`/v2.0/vpn/ipsecpolicies`	Lists IPSec policies.
POST	`/v2.0/vpn/ipsecpolicies`	Creates an IPSec policy.
GET	`/v2.0/vpn/ipsecpolicies/{ipsecpolicy_id}`	Shows details for a specified IPSec policy.
PUT	`/v2.0/vpn/ipsecpolicies/{ipsecpolicy_id}`	Updates policy settings in a specified IPSec policy.
DELETE	`/v2.0/vpn/ipsecpolicies/{ipsecpolicy_id}`	Removes a specified IPSec policy.
GET	`/v2.0/vpn/ipsecsiteconnections`	Lists IPSec connections.
POST	`/v2.0/vpn/ipsecsiteconnections`	Creates an IPSec connection.
GET	`/v2.0/vpn/ipsecsiteconnections/{connection_id}`	Shows details for a specified IPSec connection.
PUT	`/v2.0/vpn/ipsecsiteconnections/{connection_id}`	Updates connection settings for a specified IPSec connection.
DELETE	`/v2.0/vpn/ipsecsiteconnections/{connection_id}`	Removes a specified IPSec connection.

2.12.1. List VPN services

Method	URI	Description
GET	`/v2.0/vpn/vpnservices`	Lists VPN services.

This operation lists all VPN services. The list might be empty.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403)

2.12.1.1. Request

This operation does not accept a request body.

2.12.1.2. Response

Example 2.145. List VPN services: JSON response

```
{
    "vpnservices": [
        {
            "router_id": "ec8619be-0ba8-4955-8835-3b49ddb76f89",
            "status": "PENDING_CREATE",
            "name": "myservice",
            "admin_state_up": true,
            "subnet_id": "f4fb4528-ed93-467c-a57b-11c7ea9f963e",
            "tenant_id": "ccb81365fe36411a9011e90491fe1330",
            "id": "9faaf49f-dd89-4e39-a8c6-101839aa49bc",
            "description": ""
        }
    ]
}
```

2.12.2. Create VPN service

Method	URI	Description
POST	`/v2.0/vpn/vpnservices`	Creates a VPN service.

Creates a VPN service object. The service is associated with a router and a local (private) subnet. After the service is created, it can contain multiple VPN connections.

Example:

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

2.12.2.1. Request

Example 2.146. Create VPN service: JSON request

```
{
    "vpnservice": {
        "subnet_id": "f4fb4528-ed93-467c-a57b-11c7ea9f963e",
        "router_id": "ec8619be-0ba8-4955-8835-3b49ddb76f89",
        "name": "myservice",
        "admin_state_up": true
    }
}
```

2.12.2.2. Response

Example 2.147. Create VPN service: JSON response

```
{
    "vpnservice": {
        "router_id": "ec8619be-0ba8-4955-8835-3b49ddb76f89",
        "status": "PENDING_CREATE",
        "name": "myservice",
        "admin_state_up": true,
        "subnet_id": "f4fb4528-ed93-467c-a57b-11c7ea9f963e",
        "tenant_id": "ccb81365fe36411a9011e90491fe1330",
        "id": "9faaf49f-dd89-4e39-a8c6-101839aa49bc",
        "description": ""
    }
}
```

2.12.3. Show VPN service details

Method	URI	Description
GET	`/v2.0/vpn/vpnservices/{service_id}`	Shows details for a specified VPN service.

Shows the details for a specified VPN service. If the user is not an administrative user and the VPN service object does not belong to the user's tenant account, a 403 (Forbidden) error is returned.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404)

2.12.3.1. Request

This table shows the URI parameters for the show vpn service details request:

Name	Type	Description
`{service_id}`	UUID	The UUID for the VPN service.

This operation does not accept a request body.

2.12.3.2. Response

Example 2.148. Show VPN service details: JSON response

```
{
    "vpnservice": {
        "router_id": "ec8619be-0ba8-4955-8835-3b49ddb76f89",
        "status": "PENDING_CREATE",
        "name": "myservice",
        "admin_state_up": true,
        "subnet_id": "f4fb4528-ed93-467c-a57b-11c7ea9f963e",
        "tenant_id": "ccb81365fe36411a9011e90491fe1330",
        "id": "9faaf49f-dd89-4e39-a8c6-101839aa49bc",
        "description": ""
    }
}
```

2.12.4. Update VPN service

Method	URI	Description
PUT	/v2.0/vpn/vpnservices/{service_id}	Updates a specified VPN service.

This operation updates the attributes of a specified VPN service. To update a service, the service status cannot be a PENDING_* status.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), itemNotFound (404)

2.12.4.1. Request

This table shows the URI parameters for the update vpn service request:

Name	Type	Description
{service_id}	UUID	The UUID for the VPN service.

Example 2.149. Update VPN service: JSON request

```
{
    "vpnservice": {
        "description": "Updated description"
    }
}
```

2.12.4.2. Response

Example 2.150. Update VPN service: JSON response

```
{
    "vpnservice": {
        "router_id": "881b7b30-4efb-407e-a162-5630a7af3595",
        "status": "ACTIVE",
        "name": "myvpn",
        "admin_state_up": true,
        "subnet_id": "25f8a35c-82d5-4f55-a45b-6965936b33f6",
        "tenant_id": "26de9cd6cae94c8cb9f79d660d628e1f",
        "id": "41bfef97-af4e-4f6b-a5d3-4678859d2485",
        "description": "Updated description"
    }
}
```

2.12.5. Remove VPN service

Method	URI	Description
DELETE	`/v2.0/vpn/vpnservices/{service_id}`	Removes a specified VPN service.

This operation removes a specified VPN service. If the service has connections, the request is rejected.

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404), conflict (409)

2.12.5.1. Request

This table shows the URI parameters for the remove vpn service request:

Name	Type	Description
`{service_id}`	UUID	The UUID for the VPN service.

This operation does not accept a request body.

2.12.6. List IKE policies

Method	URI	Description
GET	`/v2.0/vpn/ikepolicies`	Lists IKE policies.

This operation lists all IKE policies.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403)

2.12.6.1. Request

This operation does not accept a request body.

2.12.6.2. Response

Example 2.151. List IKE policies: JSON response

```
{
    "ikepolicies": [
        {
            "name": "ikepolicy1",
            "tenant_id": "ccb81365fe36411a9011e90491fe1330",
            "auth_algorithm": "sha1",
            "encryption_algorithm": "aes-256",
            "pfs": "group5",
            "phase1_negotiation_mode": "main",
            "lifetime": {
                "units": "seconds",
                "value": 3600
            },
            "ike_version": "v1",
            "id": "5522aff7-1b3c-48dd-9c3c-b50f016b73db",
            "description": ""
        }
    ]
}
```

2.12.7. Create IKE policy

Method	URI	Description
POST	`/v2.0/vpn/ikepolicies`	Creates an IKE policy.

The IKE policy is used for phases one and two negotiation of the VPN connection. You can specify both the authentication and encryption algorithms for connections.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

2.12.7.1. Request

Example 2.152. Create IKE policy: JSON request

```
{
    "ikepolicy": {
        "phase1_negotiation_mode": "main",
        "auth_algorithm": "sha1",
        "encryption_algorithm": "aes-128",
        "pfs": "group5",
        "lifetime": {
            "units": "seconds",
            "value": 7200
        },
        "ike_version": "v1",
        "name": "ikepolicy1"
    }
}
```

2.12.7.2. Response

Example 2.153. Create IKE policy: JSON response

```
{
    "ikepolicy": {
        "name": "ikepolicy1",
        "tenant_id": "ccb81365fe36411a9011e90491fe1330",
        "auth_algorithm": "sha1",
        "encryption_algorithm": "aes-128",
        "pfs": "group5",
        "phase1_negotiation_mode": "main",
        "lifetime": {
            "units": "seconds",
            "value": 7200
        },
        "ike_version": "v1",
        "id": "5522aff7-1b3c-48dd-9c3c-b50f016b73db",
        "description": ""
    }
}
```

2.12.8. Show IKE policies

Method	URI	Description
GET	`/v2.0/vpn/ikepolicies/{ikepolicy_id}`	Shows details for a specified IKE policy.

Shows the details for a specified IKE policy.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404)

2.12.8.1. Request

This table shows the URI parameters for the show ike policies request:

Name	Type	Description
`{ikepolicy_id}`	UUID	The UUID for the IKE policy.

This operation does not accept a request body.

2.12.8.2. Response

Example 2.154. Show IKE policies: JSON response

```
{
    "ikepolicy": {
        "name": "ikepolicy1",
        "tenant_id": "ccb81365fe36411a9011e90491fe1330",
        "auth_algorithm": "sha1",
        "encryption_algorithm": "aes-256",
        "pfs": "group5",
        "phase1_negotiation_mode": "main",
        "lifetime": {
            "units": "seconds",
            "value": 3600
        },
        "ike_version": "v1",
        "id": "5522aff7-1b3c-48dd-9c3c-b50f016b73db",
        "description": ""
    }
}
```

2.12.9. Update IKE policy

Method	URI	Description
PUT	`/v2.0/vpn/ikepolicies/{ikepolicy_id}`	Updates policy settings in a specified IKE policy.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), itemNotFound (404)

2.12.9.1. Request

This table shows the URI parameters for the update ike policy request:

Name	Type	Description
`{ikepolicy_id}`	UUID	The UUID for the IKE policy.

Example 2.155. Update IKE policy: JSON request

```
{
    "ikepolicy": {
        "encryption_algorithm": "aes-256"
    }
}
```

2.12.9.2. Response

Example 2.156. Update IKE policy: JSON response

```
{
    "ikepolicy": {
        "name": "ikepolicy1",
        "tenant_id": "ccb81365fe36411a9011e90491fe1330",
        "auth_algorithm": "sha1",
        "encryption_algorithm": "aes-256",
        "pfs": "group5",
        "phase1_negotiation_mode": "main",
        "lifetime": {
            "units": "seconds",
            "value": 3600
        },
        "ike_version": "v1",
        "id": "5522aff7-1b3c-48dd-9c3c-b50f016b73db",
        "description": ""
    }
}
```

2.12.10. Remove IKE policy

Method	URI	Description
DELETE	`/v2.0/vpn/ikepolicies/{ikepolicy_id}`	Removes a specified IKE policy.

Removes the IKE policy specified in the request.

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404), conflict (409)

2.12.10.1. Request

This table shows the URI parameters for the remove ike policy request:

Name	Type	Description
`{ikepolicy_id}`	UUID	The UUID for the IKE policy.

This operation does not accept a request body.

2.12.11. List IPSec policies

Method	URI	Description
GET	/v2.0/vpn/ipsecpolicies	Lists IPSec policies.

This operation lists all IPSec policies.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403)

2.12.11.1. Request

This operation does not accept a request body.

2.12.11.2. Response

Example 2.157. List IPSec policies: JSON response

```
{
    "ipsecpolicies": [
        {
            "name": "ipsecpolicy1",
            "transform_protocol": "esp",
            "auth_algorithm": "sha1",
            "encapsulation_mode": "tunnel",
            "encryption_algorithm": "aes-128",
            "pfs": "group14",
            "tenant_id": "ccb81365fe36411a9011e90491fe1330",
            "lifetime": {
                "units": "seconds",
                "value": 3600
            },
            "id": "5291b189-fd84-46e5-84bd-78f40c05d69c",
            "description": ""
        }
    ]
}
```

2.12.12. Create IPSec policy

Method	URI	Description
POST	`/v2.0/vpn/ipsecpolicies`	Creates an IPSec policy.

The IP security policy specifies the authentication and encryption algorithms and encapsulation mode to use for the established VPN connection.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

2.12.12.1. Request

Example 2.158. Create IPSec policy: JSON request

```
{
    "ipsecpolicy": {
        "name": "ipsecpolicy1",
        "transform_protocol": "esp",
        "auth_algorithm": "sha1",
        "encapsulation_mode": "tunnel",
        "encryption_algorithm": "aes-128",
        "pfs": "group5",
        "lifetime": {
            "units": "seconds",
            "value": 7200
        }
    }
}
```

2.12.12.2. Response

Example 2.159. Create IPSec policy: JSON response

```
{
    "ipsecpolicy": {
        "name": "ipsecpolicy1",
        "transform_protocol": "esp",
        "auth_algorithm": "sha1",
        "encapsulation_mode": "tunnel",
        "encryption_algorithm": "aes-128",
        "pfs": "group5",
        "tenant_id": "ccb81365fe36411a9011e90491fe1330",
        "lifetime": {
            "units": "seconds",
            "value": 7200
        },
        "id": "5291b189-fd84-46e5-84bd-78f40c05d69c",
        "description": ""
    }
}
```

2.12.13. Show IPSec Policies

Method	URI	Description
GET	`/v2.0/vpn/ipsecpolicies/{ipsecpolicy_id}`	Shows details for a specified IPSec policy.

Shows the details for a specified IPSec policy.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404)

2.12.13.1. Request

This table shows the URI parameters for the show ipsec policies request:

Name	Type	Description
`{ipsecpolicy_id}`	UUID	The UUID for the IPSec policy.

This operation does not accept a request body.

2.12.13.2. Response

Example 2.160. Show IPSec Policies: JSON response

```
{
    "ipsecpolicy": {
        "name": "ipsecpolicy1",
        "transform_protocol": "esp",
        "auth_algorithm": "sha1",
        "encapsulation_mode": "tunnel",
        "encryption_algorithm": "aes-128",
        "pfs": "group14",
        "tenant_id": "ccb81365fe36411a9011e90491fe1330",
        "lifetime": {
            "units": "seconds",
            "value": 3600
        },
        "id": "5291b189-fd84-46e5-84bd-78f40c05d69c",
        "description": ""
    }
}
```

2.12.14. Update IPSec policy

Method	URI	Description
PUT	`/v2.0/vpn/ipsecpolicies/{ipsecpolicy_id}`	Updates policy settings in a specified IPSec policy.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), itemNotFound (404)

2.12.14.1. Request

This table shows the URI parameters for the update ipsec policy request:

Name	Type	Description
`{ipsecpolicy_id}`	UUID	The UUID for the IPSec policy.

Example 2.161. Update IPSec policy: JSON request

```
{
    "ipsecpolicy": {
        "pfs": "group14"
    }
}
```

2.12.14.2. Response

Example 2.162. Update IPSec policy: JSON response

```
{
    "ipsecpolicy": {
        "name": "ipsecpolicy1",
        "transform_protocol": "esp",
        "auth_algorithm": "sha1",
        "encapsulation_mode": "tunnel",
        "encryption_algorithm": "aes-128",
        "pfs": "group14",
        "tenant_id": "ccb81365fe36411a9011e90491fe1330",
        "lifetime": {
            "units": "seconds",
            "value": 3600
        },
        "id": "5291b189-fd84-46e5-84bd-78f40c05d69c",
        "description": ""
    }
}
```

2.12.15. Remove IPSec policy

Method	URI	Description
DELETE	`/v2.0/vpn/ipsecpolicies/{ipsecpolicy_id}`	Removes a specified IPSec policy.

Removes the IPSec policy specified in the request.

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404), conflict (409)

2.12.15.1. Request

This table shows the URI parameters for the remove ipsec policy request:

Name	Type	Description
`{ipsecpolicy_id}`	UUID	The UUID for the IPSec policy.

This operation does not accept a request body.

2.12.16. List IPSec connections

Method	URI	Description
GET	/v2.0/vpn/ipsecsiteconnections	Lists IPSec connections.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403)

2.12.16.1. Request

This operation does not accept a request body.

2.12.16.2. Response

Example 2.163. List IPSec connections: JSON response

```
{
    "ipsec_site_connections": [
        {
            "status": "PENDING_CREATE",
            "psk": "secret",
            "initiator": "bi-directional",
            "name": "vpnconnection1",
            "admin_state_up": true,
            "tenant_id": "ccb81365fe36411a9011e90491fe1330",
            "description": "",
            "auth_mode": "psk",
            "peer_cidrs": [
                "10.1.0.0/24"
            ],
            "mtu": 1500,
            "ikepolicy_id": "bf5612ac-15fb-460c-9b3d-6453da2fafa2",
            "dpd": {
                "action": "hold",
                "interval": 30,
                "timeout": 120
            },
            "route_mode": "static",
            "vpnservice_id": "c2f3178d-5530-4c4a-89fc-050ecd552636",
            "peer_address": "172.24.4.226",
            "peer_id": "172.24.4.226",
            "id": "cbc152a0-7e93-4f98-9f04-b085a4bf2511",
            "ipsecpolicy_id": "8ba867b2-67eb-4835-bb61-c226804a1584"
        }
    ]
}
```

2.12.17. Create IPSec connection

Method	URI	Description
POST	`/v2.0/vpn/ipsecsiteconnections`	Creates an IPSec connection.

Creates a site-to-site IPSec connection for a service.

Normal response codes: 201

Error response codes: badRequest (400), unauthorized (401)

2.12.17.1. Request

Example 2.164. Create IPSec connection: JSON request

```
{
    "ipsec_site_connection": {
        "psk": "secret",
        "initiator": "bi-directional",
        "ipsecpolicy_id": "22b8abdc-e822-45b3-90dd-f2c8512acfa5",
        "admin_state_up": true,
        "peer_cidrs": [
            "10.2.0.0/24"
        ],
        "mtu": "1500",
        "ikepolicy_id": "d3f373dc-0708-4224-b6f8-676adf27dab8",
        "dpd": {
            "action": "disabled",
            "interval": 60,
            "timeout": 240
        },
        "vpnservice_id": "7b347d20-6fa3-4e22-b744-c49ee235ae4f",
        "peer_address": "172.24.4.233",
        "peer_id": "172.24.4.233",
        "name": "vpnconnection1"
    }
}
```

2.12.17.2. Response

Example 2.165. Create IPSec connection: JSON response

```
{
    "ipsec_site_connection": {
        "status": "PENDING_CREATE",
        "psk": "secret",
        "initiator": "bi-directional",
        "name": "vpnconnection1",
        "admin_state_up": true,
        "tenant_id": "b6887d0b45b54a249b2ce3dee01caa47",
        "description": "",
        "auth_mode": "psk",
        "peer_cidrs": [
            "10.2.0.0/24"
        ],
        "mtu": 1500,
        "ikepolicy_id": "d3f373dc-0708-4224-b6f8-676adf27dab8",
        "dpd": {
            "action": "disabled",
            "interval": 60,
            "timeout": 240
        },
        "route_mode": "static",
        "vpnservice_id": "7b347d20-6fa3-4e22-b744-c49ee235ae4f",
        "peer_address": "172.24.4.233",
        "peer_id": "172.24.4.233",
        "id": "af44dfd7-cf91-4451-be57-cd4fdd96b5dc",
        "ipsecpolicy_id": "22b8abdc-e822-45b3-90dd-f2c8512acfa5"
    }
}
```

2.12.18. Show IPSec connection

Method	URI	Description
GET	`/v2.0/vpn/ipsecsiteconnections/{connection_id}`	Shows details for a specified IPSec connection.

Normal response codes: 200

Error response codes: unauthorized (401), forbidden (403), itemNotFound (404)

2.12.18.1. Request

This table shows the URI parameters for the show ipsec connection request:

Name	Type	Description
`{connection_id}`	UUID	The UUID for the IPSec site-to-site connection.

This operation does not accept a request body.

2.12.18.2. Response

Example 2.166. Show IPSec connection: JSON response

```
{
    "ipsec_site_connection": {
        "status": "PENDING_CREATE",
        "psk": "secret",
        "initiator": "bi-directional",
        "name": "vpnconnection1",
        "admin_state_up": true,
        "tenant_id": "ccb81365fe36411a9011e90491fe1330",
        "description": "",
        "auth_mode": "psk",
        "peer_cidrs": [
            "10.1.0.0/24"
        ],
        "mtu": 1500,
        "ikepolicy_id": "bf5612ac-15fb-460c-9b3d-6453da2fafa2",
        "dpd": {
            "action": "hold",
            "interval": 30,
            "timeout": 120
        },
        "route_mode": "static",
        "vpnservice_id": "c2f3178d-5530-4c4a-89fc-050ecd552636",
        "peer_address": "172.24.4.226",
        "peer_id": "172.24.4.226",
        "id": "cbc152a0-7e93-4f98-9f04-b085a4bf2511",
        "ipsecpolicy_id": "8ba867b2-67eb-4835-bb61-c226804a1584"
    }
}
```

2.12.19. Update IPSec connection

Method	URI	Description
PUT	/v2.0/vpn/ipsecsiteconnections/{connection_id}	Updates connection settings for a specified IPSec connection.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), itemNotFound (404)

2.12.19.1. Request

This table shows the URI parameters for the update ipsec connection request:

Name	Type	Description
{connection_id}	UUID	The UUID for the IPSec site-to-site connection.

Example 2.167. Update IPSec connection: JSON request

```
{
    "ipsec_site_connection": {
        "mtu": "2000"
    }
}
```

2.12.19.2. Response

Example 2.168. Update IPSec connection: JSON response

```
{
    "ipsec_site_connection": {
        "status": "DOWN",
        "psk": "secret",
        "initiator": "bi-directional",
        "name": "vpnconnection1",
        "admin_state_up": true,
        "tenant_id": "26de9cd6cae94c8cb9f79d660d628e1f",
        "description": "",
        "auth_mode": "psk",
        "peer_cidrs": [
            "10.2.0.0/24"
        ],
        "mtu": 2000,
        "ikepolicy_id": "771f081c-5ec8-4f9a-b041-015dfb7fbbe2",
        "dpd": {
            "action": "hold",
            "interval": 30,
            "timeout": 120
        },
        "route_mode": "static",
        "vpnservice_id": "41bfef97-af4e-4f6b-a5d3-4678859d2485",
        "peer_address": "172.24.4.233",
        "peer_id": "172.24.4.233",
        "id": "f7cf7305-f491-45f4-ad9c-8e7240fe3d72",
        "ipsecpolicy_id": "9958d4fe-3719-4e8c-84e7-9893895b76b4"
    }
}
```

2.12.20. Remove IPSec connection

Method	URI	Description
DELETE	`/v2.0/vpn/ipsecsiteconnections/{connection_id}`	Removes a specified IPSec connection.

Removes the IPSec connection specified in the request.

Normal response codes: 204

Error response codes: unauthorized (401), itemNotFound (404), conflict (409)

2.12.20.1. Request

This table shows the URI parameters for the remove ipsec connection request:

Name	Type	Description
`{connection_id}`	UUID	The UUID for the IPSec site-to-site connection.

This operation does not accept a request body.

2.13. Extra routes

Adds extra routes to the `router` resource.

You can update a router to add a set of nexthop IPs and destination CIDRs.

Note

The nexthop IP must be a part of a subnet to which the router interfaces are connected. You can configure the `routes` attribute on only update operations.

Method	URI	Description
PUT	`/v2.0/routers/{router_id}`	Configures extra routes on a specified router.

2.13.1. Update router

Method	URI	Description
PUT	`/v2.0/routers/{router_id}`	Configures extra routes on a specified router.

The next hop IP address must be a part of one of the subnets to which the router interfaces are connected. Otherwise, the server responds with the `400 Bad Request` error code.

When a validation error is detected, such as a format error of IP address or CIDR, the server responds with the `400 Bad Request` error code.

When Networking receives a request to delete the router interface for subnets that are used by one or more routes, it responds with a `409 Conflict` error code.

Normal response codes: 200

Error response codes: badRequest (400), unauthorized (401), itemNotFound (404), conflict (409)

2.13.1.1. Request

This table shows the URI parameters for the update router request:

Name	Type	Description
`{router_id}`	UUID	The UUID for the router of interest to you.

Example 2.169. Update router: JSON request

```
{
    "router": {
        "routes": [
            {
                "nexthop": "10.1.0.10",
                "destination": "40.0.1.0/24"
            }
        ]
    }
}
```

2.13.1.2. Response

Example 2.170. Update router: JSON response

```
{
    "router": {
        "status": "ACTIVE",
        "external_gateway_info": {
            "network_id": "5c26e0bb-a9a9-429c-9703-5c417a221096"
        },
        "name": "router1",
        "admin_state_up": true,
        "tenant_id": "936fa220b2c24a87af51026439af7a3e",
        "routes": [
            {
                "nexthop": "10.1.0.10",
                "destination": "40.0.1.0/24"
            }
        ],
        "id": "babc8173-46f6-4b6f-8b95-38c1683a4e22"
    }
}
```