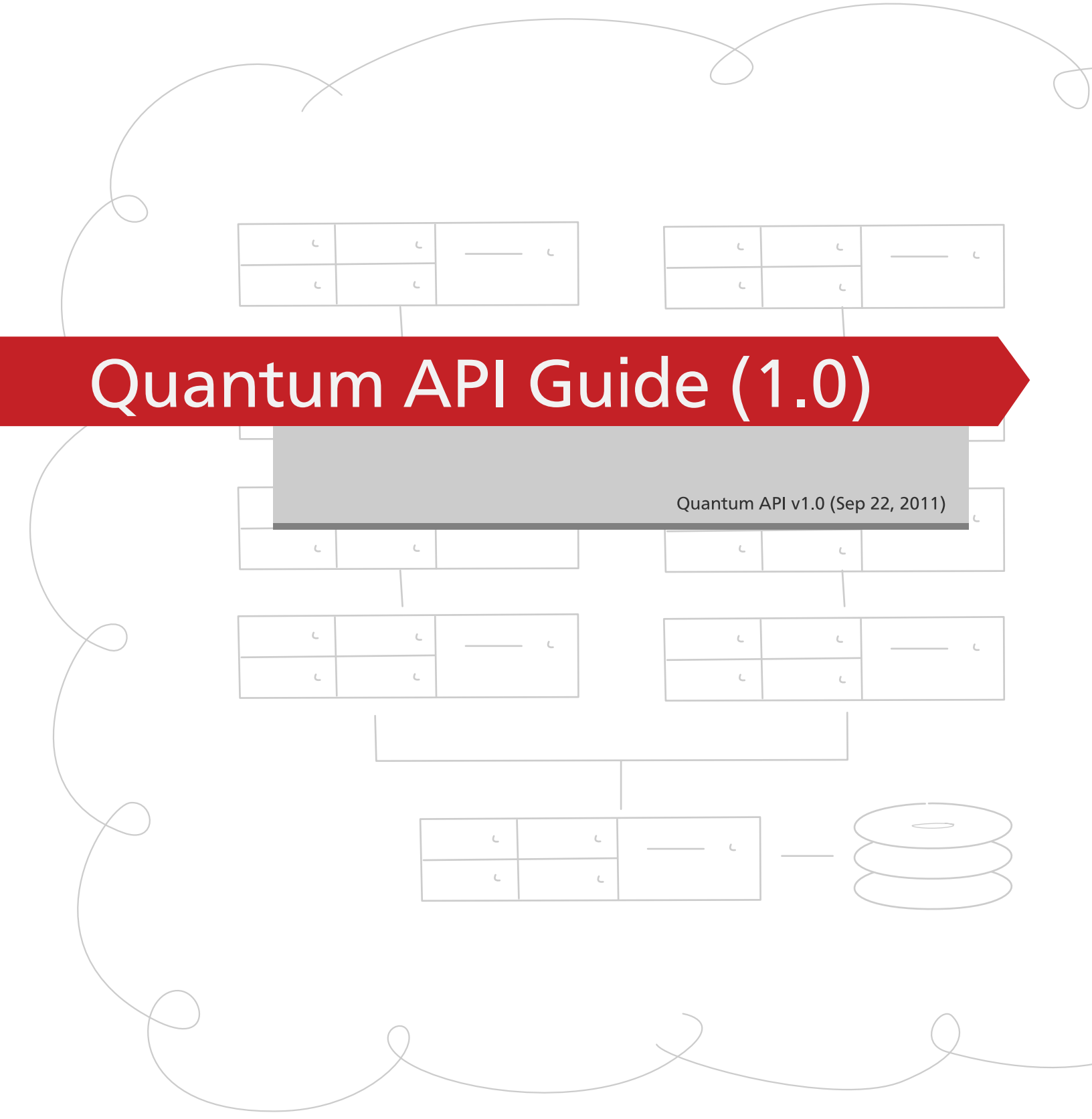

Quantum API Guide (1.0)

Quantum API v1.0 (Sep 22, 2011)

openstack™

docs.openstack.org

Quantum API Guide (1.0)

Quantum API v1.0 (2011-09-22)

This document is intended for software developers interested in developing applications using the OpenStack Quantum Layer-2 Networking Service (API).

Table of Contents

List of Tables

List of Examples

1. Overview

Quantum is a project to provide "network connectivity as a service" between devices managed by the OpenStack compute service. For more information on Quantum and the other network-related projects please check the Quantum home page (http://wiki.openstack.org/Quantum) and the NetStack home page (http://wiki.openstack.org/Network).

We welcome feedback, comments, and bug reports at bugs.launchpad.net/Quantum.

1.1. Intended Audience

This Guide is intended to assist software developers who want to develop applications using the Quantum API. To use the information provided here, you should first have a general understanding of the OpenStack Quantum network service, the OpenStack compute service (Nova), and the integration between the two. The user should also have access to a plugin providing the implementation for the API described in this document. Two plugins are included in the Quantum distribution:

- Openvswitch - Implementing Quantum API with Open vSwitch
- Cisco - Implementing Quantum API for Cisco UCS blades and Nexus switches

You should also be familiar with:

- ReSTful web services
- HTTP/1.1
- JSON and/or XML data serialization formats

1.2. Document Change History

The most recent changes are described in the table below:

Revision Date	Summary of Changes
September 22, 2011	• Initial release.

1.3. Glossary

Term	Description
Entity	A generic term for any piece of hardware or software desiring to connect to the network services provided by Quantum. An entity may make use of Quantum by implementing a VIF.
Layer-2 network	A virtual Ethernet network managed by the Quantum service. For the time being, Quantum will manage only Ethernet networks.
Network	A virtual network providing connectivity between entities, i.e.: collection of virtual ports sharing network connectivity. In the Quantum terminology, a network is always a Layer-2 network.
Plugin	Software component providing the actual implementation for Quantum APIs.
Port	A port on the virtual network switch represented by a Quantum virtual Layer-2 network.
VIF	A Virtual network InterFace plugged into a port of a Quantum network; typically a virtual network interface belonging to a VM
Attachment	Association of an interface identifier to a logical port, which represent 'plugging' an interface into a port.

1.4. Theory of Operation

This section presents the objects and semantics of Quantum's logical model.

Quantum abstracts the physical implementation of the network, allowing plugins to configure and manage physical resources. Quantum is a standalone service, in that it requires no other project within OpenStack to function correctly.

Further Quantum is agnostic to the entities it allows to connect. While we anticipate Nova instances will be a heavy user of Quantum, any entity can make use of any Quantum created network so long as it provides an appropriate interfaces for exposing VIFs to Quantum itself.

Virtual Interfaces(VIF) in the logical model are analogous to physical network interface cards (NICs). VIFs are typically owned a managed by an external service; for instance when Quantum is used for building OpenStack networks, VIFs would be created, owned, and managed in Nova. VIFs are connected to Quantum networks via ports. A port is analogous to a port on a network switch, and it has an administrative state. Quantum API allows for controlling the administrative state of the port, which can be either 'DOWN' or 'ACTIVE'.

When a VIF is attached to a port the Quantum API creates an attachment object, which specifies the fact that a VIF with a given identifier is plugged into the port.

The Quantum plugin is responsible for managing virtual and/or physical network switches to implement the network forwarding connectivity described by the Quantum networks, ports, and attachments.

VIFs attached to ACTIVE ports are required to have access to the L2 broadcast domain defined by the network where they are attached. Each VIF shall be capable of exchanging traffic with all other entities attached through ACTIVE ports.

2. Concepts

To use the Quantum API effectively, developers should understand the concepts introduced in this chapter.

2.1. Network

Each tenant can define one or more networks. A network is a virtual isolated layer-2 broadcast domain reserved to the tenant. A tenant can create several ports for a network, and plug virtual interfaces into these ports.

2.2. Port

A port represents a virtual switch port on a logical network switch where all the interfaces attached to a given network are connected.

A port has an administrative state which is either 'DOWN' or 'ACTIVE'. Ports which are administratively down will not be able to receive/send traffic.

2.3. Attachment

An attachment represents an interface plugged into a logical port. At any time at most one attachment can be plugged into a given port.

An attachment typically identified a virtual network interface. Network interfaces are typically defined in an external services which uses Quantum, for instance the OpenStack Compute service, Nova.

3. General API Information

The OpenStack Quantum API is defined as a ReSTful HTTP service. The API takes advantage of all aspects of the HTTP protocol (methods, URIs, media types, response codes, etc.) and providers are free to use existing features of the protocol such as caching, persistent connections, and content compression among others. For example, providers who employ a caching layer may respond with a 203 when a request is served from the cache instead of a 200. Additionally, providers may offer support for conditional **GET** requests using ETags, or they may send a redirect in response to a **GET** request. Clients should be written to account for these differences.

3.1. Authentication

The current version of the OpenStack Quantum service does not require that each request will include the credentials of the user submitting the request.

However, Quantum deployments can support several authentication schemes (OAuth, Basic Auth, Token). The authentication scheme used is determined by the provider of the Quantum service. Please contact your provider to determine the best way to authenticate against this API.

Ideally, middleware modules for Authentication and/or Authorization should be inserted in the first stages of the Quantum pipeline (available in `etc/quantum.conf`).

Note

Some authentication schemes may require that the API operate using SSL over HTTP (HTTPS).

3.2. URI structure

Each request to the OpenStack Quantum API must refer to a specific version of the API itself, and it must also identify the tenant for which the request is being sent.

This information is specified in the URI. The URI for each request to the OpenStack Quantum API should be prefixed with the API version identifier and the tenant identifier, as follows:

```
/{Quantum-version}/tenants/{tenant-id}/{Quantum-API-entity}
```

As an example, the following URI represents a request for retrieving all the networks configured for the tenant "ABC" using the 1.0 API.

```
/v1.0/ABC/networks
```

3.3. Request/Response Types

The OpenStack Quantum API supports both the JSON and XML data serialization formats. The format for both the request and the response can be specified either using the `Content-Type` header, the `Accept` header or adding an `.xml` or `.json` extension to the request URI.

If conflicting formats are specified in headers and/or format extensions, the latter takes precedence. XML is currently the default format for both requests and responses.

Table 3.1. JSON and XML Response Formats

Format	Accept Header	Query Extension	Default
JSON	application/json	.json	No
XML	application/xml	.xml	Yes

Example 3.1. Request/Response with Headers: JSON

```
POST   /v1.0/tenants/tenantX/networks HTTP/1.1
Host   127.0.0.1:9696
Accept   text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Content-Type application/json; charset=UTF-8
Content-Length 30
```

```
{
   "network": {
       "name": "test"
    }
}
```

```
HTTP/1.1 200 Accepted
Content-Type application/json
Content-Length 59
```

```
{
```

```
    "network": {
       "id": "611851f2-df8b-4455-b84b-8c73b7ca5dec"
    }
}
```

Notice, in the above example, that both the `Content-Type` and the `Accept` headers are specified. The `Content-Type` header always takes precedence over the `Accept` header. The value of the latter header is therefore ignored in the example above.

Example 3.2. Request/Response with Headers: XML

```
POST    /v1.0/tenants/tenantX/networks HTTP/1.1
Host   127.0.0.1:9696
Accept   text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Content-Type application/xml; charset=UTF-8
Content-Length 22
```

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

```
HTTP/1.1 200 Accepted
Content-Type application/xml
Content-Length 52
```

```
<?xml version="1.0" encoding="UTF-8"?>
<network id="e1150d1c-e953-402d-aa75-e35d803d914b"/>
```

Example 3.3. Request/Response with Extensions: JSON

```
POST    /v1.0/tenants/tenantX/networks.json HTTP/1.1
Host   127.0.0.1:9696
Accept   text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Content-Type application/json; charset=UTF-8
Content-Length 30
```

```
{
   "network": {
       "name": "test"
    }
}
```

```
HTTP/1.1 200 Accepted
Content-Type application/json
Content-Length 59
```

```
{
    "network": {
       "id": "611851f2-df8b-4455-b84b-8c73b7ca5dec"
    }
}
```

3.4. Asynchronous Behavior by Quantum Plugins

The Quantum API presents a logical model of network connectivity consisting of networks, ports, and attachments. It is up to the Quantum plugin to communicate with all managed virtual and/or physical switches to ensure that these devices implement packet forwarding behavior consistent with the logical model.

The plugin's task of mapping from the logical model to the physical world might happen asynchronously. This means that when an API client modifies the logical model using an HTTP POST, PUT, or DELETE, the API call may return prior to the plugin performing any modifications to underlying virtual and/or physical switching devices. The only guarantee an API client has is that all subsequent API calls will properly reflect the changed logical model.

As a concrete example, consider the case where a client uses an HTTP PUT to set the attachment for a port. There is no guarantee that packets sent by the interface named in the attachment will be forwarded immediately once the HTTP call returns. However, there is a guarantee that a subsequent HTTP GET to view the attachment on that port would return the new attachment value.

Note

Future versions of the API may expose a notion of an "operational status" on a logical entity like a network or port.

This would indicate whether the Quantum plugin had successfully configured virtual and/or physical switches in order to implement the network connectivity described by that element of the logical model.

For example, a port might have an operational status of "DOWN" because the VM interface specified as an attachment was not currently running on any physical server.

3.5. Versions

The Quantum API uses a URI based versioning scheme. The first element of the URI path contains the target version identifier.

Example 3.4. Request with URI versioning

```
GET    /v1.0/tenants/tenantX/networks HTTP/1.1
Host   127.0.0.1:9696
Accept   text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Content-Type application/xml; charset=UTF-8
Content-Length 22
```

Available API versions can be retrieved by performing a **GET** on the root URL (i.e. with the version and everything to the right of it truncated) of the Quantum Service.

Example 3.5. Versions List Request/Response (XML)

```
GET    / HTTP/1.1
Host   127.0.0.1:9696
Content-Type application/xml
```

```
<versions>
 <version id="v1.0" status="CURRENT">
  <links>
   <link href="http://127.0.0.1:9696/v1.0" rel="self"/>
  </links>
 </version>
 <version id="v1.1" status="FUTURE">
  <links>
   <link href="http://127.0.0.1:9696/v1.1" rel="self"/>
  </links>
 </version>
</versions>
```

Example 3.6. Version List Request/Response: JSON

```
GET    / HTTP/1.1
Host   127.0.0.1:9696
Content-Type application/json
```

```
{
   "versions": [
       {
           "status": "CURRENT",
           "id": "v1.0",
           "links": [
               {
                   "href": "http://127.0.0.1:9696/v1.0",
                   "rel": "self"
               }
           ]
       },
       {
           "status": "FUTURE",
           "id": "v1.1",
           "links": [
               {
                   "href": "http://127.0.0.1:9696/v1.1",
                   "rel": "self"
               }
           ]
       }
   ]
}
```

3.6. Extensions

The Quantum API is extensible. Extensions serve several purposes:

- They allow the introduction of new features in the API without requiring a version change;
- They allow the introduction of vendor specific niche functionality
- They act as a proving ground for experimental functionalities which might be included in a future version of the API.

Applications can programmatically determine what extensions are available by performing a **GET** on the `/v1.0/extensions` URI. Note that this is a versioned request — that is, an extension available in one API version may not be available in another.

Example 3.7. Extensions Response: XML

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

<extensions>

    <extension
        name="Cisco Port Profile"
        namespace="http://docs.ciscocloud.com/api/ext/portprofile/v1.0"
        alias="Cisco Port Profile"
        updated="2011-07-23T13:25:27-06:00">
        <description>Portprofile include QoS information</description>
    </extension>

    <extension
        name="Cisco qos"
        namespace="http://docs.ciscocloud.com/api/ext/qos/v1.0"
        alias="Cisco qos"
        updated="2011-07-25T13:25:27-06:00">
        <description>qos include qos_name and qos_desc</description>
    </extension>

</extensions>
```

Example 3.8. Extensions Response: JSON

```
{
    "extensions": [
        {
            "name": "Cisco Port Profile",
            "namespace": "http://docs.ciscocloud.com/api/ext/portprofile/
v1.0",
            "alias": "Cisco Port Profile",
            "updated": "2011-07-23T13:25:27-06:00",
            "description": "Portprofile include QoS information",
        },
        {
            "name": "Cisco qos",
            "namespace": "http://docs.ciscocloud.com/api/ext/qos/v1.0",
            "alias": "Cisco qos",
            "updated": "2011-07-25T13:25:27-06:00",
            "description": "qos include qos_name and qos_desc",
        },

    ]
}
```

Extensions may also be queried individually by their unique alias by performing a **GET** on the `/v1.0/extensions/alias_name`. This provides the simplest method of checking if an extension is available as an unavailable extension will issue an itemNotFound (404) response.

Note

Existing core API resources can be extended with new actions or extra data in request/response of existing actions. Further new resources can also be added as extensions. Extensions usually have vendor specific tags that prevent clash with other extensions. Availability of an extension will vary with deployments and the specific plugin in use.

Important

Applications should be prepared to ignore response data that contains extension elements. Applications should also verify that an extension is available before submitting an extended request.

3.7. Faults

When an error occurs at request time, the system will return an HTTP error response code denoting the type of error. The system will also return additional information about the fault in the body of the response.

Example 3.9. "Network not found" fault Response (XML)

```
<networkNotFound code="420" xmlns="http://netstack.org/quantum/api/v1.0">
   <message>
       Unable to find a network with the specified identifier.
   </message>
   <detail>
       Network 8de6af7c-ef95-4ac1-9d37-172f8df33a1f could not be found
   </detail>
</networkNotFound
```

Example 3.10. "Network not found" fault Response (XML)

```
{
   "networkNotFound": {
       "message": "Unable to find a network with the specified identifier.",
        "code": 420,
        "detail": "Network 8de6af7c-ef95-4ac1-9d37-172f8df33a1f could not be
 found"
   }
}
```

The error code is returned in the body of the response for convenience. The message section returns a human-readable message that is appropriate for display to the end user. The detail section is optional and may contain information—for example, a stack trace—to assist in tracking down an error.

The root element of the fault (e.g. networkNotFound) may change depending on the type of error. The following is a list of possible elements along with their associated error codes.

Table 3.2. Fault Elements and Error Codes

Fault Element	Error Code	Description
BadRequest	400	Malformed request body. The Quantum service is unable to parse the contents of the request body.
Unauthorized	401	User has not provided authentication credentials. If authentication is provided by the Keystone identity service, this might mean that either no authentication token has been supplied in the request, or that the token itself is either invalid or expired.
Forbidden	403	The user does not have the necessary rights to execute the requested operation
ItemNotFound	404	The requested resource does not exist on the Quantum API server
NetworkNotFound	420	The specified network has not been created or has been removed.
NetworkInUse	421	The specified network has attachments plugged into one or more of its ports.
PortNotFound	430	The specified port has not been created or has been removed.
RequestedStateInvalid	431	Indicates a request to change port to an administrative state not currently supported.
PortInUse	432	The specified port cannot be removed as there is an attachment plugged in it.
AlreadyAttached	440	Attachment is already plugged into another port.

Note

The error codes 401 and 403 will be returned only if some Authentication/ Authorization system has been enabled in the Quantum pipeline

4. API Operations

4.1. Networks

This section describes the operations exposed by Quantum API for manipulating network resources.

4.1.1. List Networks

Verb	URI	Description
GET	/tenants/*{tenant-id}*/networks	List summary of networks configured in Quantum for a given tenant, identified by tenant-id

Normal Response Code(s): 200

Error Response Code(s): Unauthorized (401), Forbidden (403)

This operation returns the list of all networks currently defined in Quantum for the tenant specified in the request URI. The returned list will provide the unique identifier of each network configured for the tenant specified in the resource URI.

Note

TenantId is a unique tenant identifier. The Quantum service does not directly manages tenants. Tenant management should be performed by the identity service

This operation does not require a request body.

Example 4.1. Networks List Request/Response (XML)

Request:

```
GET /tenants/XYZ/networks.xml
```

Response:

```
<networks>
    <network id="8bec1293-16bd-4568-ba75-1f58bec0b4c3"/>
    <network id="2a39409c-7146-4501-8429-3579e03e9b56"/>
</networks>
```

Example 4.2. Networks List Request/Response (JSON)

Request:

```
GET /tenants/XYZ/networks.json
```

Response:

```
{
    "networks":
        [
            {
                "id": "8bec1293-16bd-4568-ba75-1f58bec0b4c3"
            },
            {
                "id": "2a39409c-7146-4501-8429-3579e03e9b56"
            }
        ]
}
```

4.1.2. List Networks Details

Verb	URI	Description
GET	/tenants/*{tenant-id}*/networks/detail	List more detailed information about networks configured in Quantum for a given tenant, identified by tenant-id

Normal Response Code(s): 200

Error Response Code(s): Unauthorized (401), Forbidden (403)

This operation returns the list of all networks currently defined in Quantum; for each network, its identifier and name are returned.

This operation does not require a request body.

Example 4.3. Networks List Request/Response (XML)

Request:

```
GET /tenants/XYZ/networks/detail.xml
```

Response:

```
<networks>
    <network
     id="8bec1293-16bd-4568-ba75-1f58bec0b4c3"
        name="network_1"/>
    <network
     id="2a39409c-7146-4501-8429-3579e03e9b56"
        name="network_2"/>
</networks>
```

Example 4.4. Networks List Request/Response (JSON)

Request:

```
GET /tenants/XYZ/networks/detail.json
```

Response:

```
{
    "networks":
         [
          {
           "id": "8bec1293-16bd-4568-ba75-1f58bec0b4c3",
           "name": "network_1"
          },
          {
           "id": "2a39409c-7146-4501-8429-3579e03e9b56",
           "name": "network_2"
          }
         ]
}
```

4.1.3. Show Network

Verb	URI	Description
GET	/tenants/*{tenant-id}*/ networks/*{network-id}*	List information for a specific network, identified by *network-id*, for a given tenant, identified by *tenant-id*.

Normal Response Code(s): 200

Error Response Code(s): Unauthorized (401), Forbidden (403), NetworkNotFound (420),

This operation returns the identifier and the name for a specific network configured in Quantum.

This operation does not require a request body.

Example 4.5. Show Network Request/Response (XML)

Request:

```
GET /tenants/XYZ/networks/8bec1293-16bd-4568-ba75-1f58bec0b4c3.xml
```

Response:

```
    <network
           id="8bec1293-16bd-4568-ba75-1f58bec0b4c3"
           name="test_network"/>
```

Example 4.6. Show Network Request/Response (JSON)

Request:

```
GET /tenants/XYZ/networks/8bec1293-16bd-4568-ba75-1f58bec0b4c3.json
```

Response:

```
{
   "network":
       {
           "id": "8bec1293-16bd-4568-ba75-1f58bec0b4c3",
           "name": "test_network"
       }
}
```

4.1.4. Show Network Details

Verb	URI	Description
GET	/tenants/*{tenant-id}*/ networks/*{network-id}*/detail	List detailed information for a specific network, identified by *network-id*, for a given tenant, identified by *tenant-id*.

Normal Response Code(s): 200

Error Response Code(s): Unauthorized (401), Forbidden (403), NetworkNotFound (420),

This operation returns detailed information concerning the network specified in the request URI. Returned data include the full list of ports configured for the network and attachments plugged into such ports.

If no attachment is plugged into a port, the response will not include an `attachment` child element for that port.

This operation does not require a request body.

Example 4.7. Show Network Detail Request/Response (XML)

Request:

```
GET /tenants/XYZ/networks/8bec1293-16bd-4568-ba75-1f58bec0b4c3/detail.xml
```

Response:

```
   <network
           id="8bec1293-16bd-4568-ba75-1f58bec0b4c3"
           name="test_network">
        <ports>
            <port
    id="98017ddc-efc8-4c25-a915-774b2a633855"
    status="DOWN"/>
            <port
```

```
    id="b832be00-6553-4f69-af33-acd554e36d08"
    status="ACTIVE">
                <attachment id="test_interface_identifier"/>
            </port>
        </ports>
    </network>
```

Example 4.8. Show Network Detail Request/Response (JSON)

Request:

```
GET /tenants/XYZ/networks/8bec1293-16bd-4568-ba75-1f58bec0b4c3/detail.json
```

Response:

```
{
   "network":
       {
           "id": "8bec1293-16bd-4568-ba75-1f58bec0b4c3",
           "name": "test_network"
           "ports":
               [
                {
                    "id": "98017ddc-efc8-4c25-a915-774b2a633855",
                    "state": "DOWN"
                },
                {
                    "id": b832be00-6553-4f69-af33-acd554e36d08",
                    "state": "ACTIVE",
                    "attachment":
                        {
                            "id": "test_interface_identifier"
                        }
                }
               ]
       }
}
```

4.1.5. Create Network

Verb	URI	Description
POST	/tenants/*{tenant-id}*/networks	Creates a new logical layer-2 network for the tenant identified by *tenant-id*

Normal Response Code(s): 200

Error Response Code(s): BadRequest (400) Unauthorized (401), Forbidden (403),

This operation creates a Layer-2 network in Quantum based on the information provided in the request body.

Quantum validates the request, and dispatches it to the plugin, and then returns the unique identifier of the network to the caller. Although the network API entity can

be immediately used for other operations, this does not guarantee that the network will be available when the API call returns, as this depends on the particular plugin implementation.

If the validation phase fails, the network object is not created at all, and a 400 error is returned to the caller.

Note

The Quantum API v1.0 does not provide an interface for checking the progress of asynchronous operations performed by plugins.

This will be addressed in future releases of the Quantum API.

The body for this request must contain a Network object specifying a symbolic name for the network.

Example 4.9. Create Network Request/Response (XML)

Request:

```
POST /tenants/XYZ/networks.xml
```

```
<network
    name="test_create_network"/>
```

Response:

```
<network
    id="158233b0-ca9a-40b4-8614-54a4a99d47d1"/>
```

Example 4.10. Create Network Request/Response (JSON)

Request:

```
POST /tenants/XYZ/networks.json
```

```
{
   "network":
       {
            "name": "test_create_network"
       }
}
```

Response:

```
{
   "network":
       {
           "id": "158233b0-ca9a-40b4-8614-54a4a99d47d1",
```

```
        }
}
```

4.1.6. Update Network

Verb	URI	Description
PUT	/tenants/*{tenant-id}*/ networks/*{network-id}*	Renames the network identified by *network-id* for the tenant identified by *tenant-id*

Normal Response Code(s): 204

Error Response Code(s): BadRequest (400) Unauthorized (401), Forbidden (403), NetworkNotFound (420)

This operation renames a Quantum network using the data provided in the request body.

The body for this request must contain a Network object specifying a symbolic name for the network. The network entity specified in the request body can contain the network's identifier as well, even if it is not required, as the identifier must be expressed on the URI; in this case the identifier in the request body will be ignored.

Example 4.11. Update Network Request/Response (XML)

Request:

```
PUT /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1.xml
```

```
<network
    name="test_create_network"/>
```

Response:

No data returned in response body

Example 4.12. Update Network Request/Response (JSON)

Request:

```
PUT /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1.json
```

```
{
   "network":
       {
            "name": "test_create_network"
       }
}
```

Response:

No data returned in response body

4.1.7. Delete Network

Verb	URI	Description
PUT	/tenants/*{tenant-id}*/ networks/*{network-id}*	Destroys the network identified by *network-id* for the tenant identified by *tenant-id*

Normal Response Code(s): 204

Error Response Code(s): BadRequest (400) Unauthorized (401), Forbidden (403), NetworkNotFound (420), NetworkInUse (421)

This operation removes the network specified in the URI. This request will fail as long as there is at least one port on the network with attachments plugged in it. If all ports on the networks are unattached, they will be destroyed together with the network itself.

As for the create operation there is no guarantee that the plugin will have completely removed the network when the call returns. Quantum forwards the request to the plugin, which will then destroy the network.

Please note that this operation cannot be undone.

This operation does not require a request body.

Example 4.13. Delete Network Request/Response (XML)

Request:

```
DELETE /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1.xml
```

Response:

No data returned in response body

Example 4.14. Update Network Request/Response (JSON)

Request:

```
DELETE /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1.json
```

Response:

No data returned in response body

4.2. Ports

This section describes the operations exposed by Quantum API for manipulating port resources.

4.2.1. List Ports

Verb	URI	Description
GET	/tenants/*{tenant-id}*/networks/*{network-id}*/*ports*	Lists all the ports currently defined for a Quantum network, identified by *network-id*

Normal Response Code(s): 200

Error Response Code(s): Unauthorized (401), Forbidden (403), NetworkNotFound (420)

This operation lists all the ports currently configured for a network. For each port the response reports its unique identifier. If no ports have been created on the network an empty list will be returned.

This operation does not require a request body.

Example 4.15. Port List Request/Response (XML)

Request:

```
GET /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports.xml
```

Response:

```
<ports>
    <port
     id="98017ddc-efc8-4c25-a915-774b2a633855"/>
    <port
     id="b832be00-6553-4f69-af33-acd554e36d08"/>
</ports>
```

Example 4.16. Port List Request/Response (JSON)

Request:

```
GET /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports.json
```

Response:

```
{
    "ports":
        [
           {
               "id": "98017ddc-efc8-4c25-a915-774b2a633855"
           },
           {
               "id": "b832be00-6553-4f69-af33-acd554e36d08"
           }
        ]
}
```

4.2.2. List Ports Details

Verb	URI	Description
GET	/tenants/*{tenant-id}*/networks/*{network-id}*/ports/detail	Lists detailed information for all the ports currently defined for the network identified by *network-id*

Normal Response Code(s): 200

Error Response Code(s): Unauthorized (401), Forbidden (403), NetworkNotFound (420)

This operation lists detailed information for all the ports currently configured for a network. Response for each port includes its identifier and the current administrative state. If no ports have been created on the network an empty list will be returned.

This operation does not require a request body.

Example 4.17. Port List Details Request/Response (XML)

Request:

```
GET /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/
detail.xml
```

Response:

```
<ports>
    <port
     id="98017ddc-efc8-4c25-a915-774b2a633855"
        state="ACTIVE"/>
    <port
     id="b832be00-6553-4f69-af33-acd554e36d08"
        state="ACTIVE"/>
```

```
</ports>
```

Example 4.18. Port List Details Request/Response (JSON)

Request:

```
GET /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/
detail.json
```

Response:

```
{
   "ports": [
       {
           "id": "98017ddc-efc8-4c25-a915-774b2a633855",
           "state": "ACTIVE",
       },
       {
           "id": "b832be00-6553-4f69-af33-acd554e36d08",
           "state": "ACTIVE",
       }
   ]
}
```

4.2.3. Show Port

Verb	URI	Description
GET	/tenants/*{tenant-id}*/networks/*{network-id}*/ports/ *{port-id}*	Retrieves the port *port-id* configured for the network *network-id* belonging to the tenant *tenant-id*.

Normal Response Code(s): 200

Error Response Code(s): Unauthorized (401), Forbidden (403), NetworkNotFound (420), PortNotFound (430)

This operation returns the unique identifier and the current administrative state for a specific port configured for the network specified in the request URI.

This operation does not require a request body.

Example 4.19. Show Port Request/Response (XML)

Request:

```
GET /tenants/33/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/98017ddc-
efc8-4c25-a915-774b2a633855.xml
```

Response:

```
<port
    id="98017ddc-efc8-4c25-a915-774b2a633855"
    state="DOWN"/>
```

Example 4.20. Show Port Request/Response (JSON)

Request:

```
GET /tenants/33/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/98017ddc-
efc8-4c25-a915-774b2a633855.json
```

Response:

```
{
    "port":
        {
            "state": "DOWN",
            "id": "98017ddc-efc8-4c25-a915-774b2a633855"
        }
}
```

4.2.4. Show Port Details

Verb	URI	Description
GET	/tenants/*{tenant-id}*/networks/*{network-id}*/ports/ *{port-id}*/detail	Retrieves detailed information for the port *port-id* configured for the network *network-id* belonging to the tenant *tenant-id*.

Normal Response Code(s): 200

Error Response Code(s): Unauthorized (401), Forbidden (403), NetworkNotFound (420), PortNotFound (430)

This operation provides at least the identifier and the current administrative state for specific port configured for a given network.

If an attachment is plugged into the port, this operation will return the identifier of the attachment as well.

This operation does not require a request body.

Example 4.21. Show Port Detail Request/Response (XML)

Request:

```
GET /tenants/33/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/98017ddc-
efc8-4c25-a915-774b2a633855/detail.xml
```

Response:

```
<port
    id="98017ddc-efc8-4c25-a915-774b2a633855"
    state="DOWN">
    <attachment
     id="test_interface_identifier"/>
```

```
</port>
```

Example 4.22. Show Port Detail Request/Response (JSON)

Request:

```
GET /tenants/33/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/98017ddc-
efc8-4c25-a915-774b2a633855/detail.json
```

Response:

```
{
    "port":
        {
            "state": "DOWN",
            "id": "98017ddc-efc8-4c25-a915-774b2a633855"
            "attachment":
                {
                    "id": "test_interface_identifier"
                }
        }
}
```

4.2.5. Create Port

Verb	URI	Description
POST	/tenants/*{tenant-id}*/networks/*{network-id}*/ports	Creates a port on the network specified in the request URI, identified by *network-id*

Normal Response Code(s): 200

Error Response Code(s): BadRequest (400), Unauthorized (401), Forbidden (403), NetworkNotFound (420), RequestedStateInvalid (431)

This operation creates a port on a Quantum network based on the information provided in the request body. Quantum validates the request, and dispatches the request to the plugin, which creates the port and attaches it to the appropriate network.

This operation could not be implemented for some plugins as the number of ports available might be fixed when the network is created.

If the validation phase fails, the port object is not created at all, and a `BadRequest` error is returned to the caller.

The operation returns a port with an identifier, and set its administrative state set to `DOWN`, unless a state has been explicitly specified in the request body.

Please note that this operation does not guarantee that the port has been actually created when the calls returns, as the plugin might still be performing the necessary operations on the network infrastructure. However, the port entity can be immediately used for API operations.

The request body is not mandatory for this operation, but it can optionally contain the administrative state for the port being created, which can be either `DOWN` or `ACTIVE`. The administrative state should be encapsulated into a Port object within the request body, as shown in the example below.

Example 4.23. Create Port Request/Response (XML)

Request:

```
POST /tenants/33/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports.xml
```

```
<port
 state="ACTIVE"/>
```

Response:

```
<port
 id="98017ddc-efc8-4c25-a915-774b2a633855"/>
```

Example 4.24. Create Port Request/Response (JSON)

Request:

```
POST /tenants/33/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports.json
```

```
{
    "port":
     {
         "state": "ACTIVE"
     }
}
```

Response:

```
{
    "port":
     {
      "id": "98017ddc-efc8-4c25-a915-774b2a633855"
     }
}
```

4.2.6. Update Port

Verb	URI	Description
PUT	/tenants/*{tenant-id}*/networks/*{network-id}*/ports/ *{port-id}*	Sets the administrative state for the port identified by *port-id* on the network identified by *network-id*

Normal Response Code(s): 204

Error Response Code(s): BadRequest (400), Unauthorized (401), Forbidden (403), NetworkNotFound (420), PortNotFound (430), RequestedStateInvalid (431)

This operation sets the administrative state for a port. Currently Quantum recognizes two port states: `DOWN` and `ACTIVE`. In the `DOWN` state a port will not provide connectivity to the network.

This feature allows the tenant the ability to take entities offline without affecting the logical topology.

The operation will return the `RequestedStateInvalid` error code if the specified administrative state is not either `DOWN` or `ACTIVE`.

The operation's request body must contain a Port object with the new administrative state for the port.

Example 4.25. Set Port State Request/Response (XML)

Request:

```
PUT tenants/33/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/98017ddc-
efc8-4c25-a915-774b2a633855.xml
```

```
<port
 state="ACTIVE"/>
```

Response:

No data returned in response body.

Example 4.26. Create Port Request/Response (JSON)

Request:

```
PUT tenants/33/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/98017ddc-
efc8-4c25-a915-774b2a633855.json
```

```
{
    "port":
     {
         "state": "ACTIVE"
     }
}
```

Response:

No data returned in response body.

4.2.7. Delete Port

Verb	URI	Description
DELETE	/tenants/*{tenant-id}*/networks/*{network-id}*/ports/ *{port-id}*	Destroys the port identified by *port-id* on the network identified by *network-id*

Normal Response Code(s): 204

Error Response Code(s): BadRequest (400), Unauthorized (401), Forbidden (403), NetworkNotFound (420), PortNotFound (430), PortInUse (432)

This operation removes a port from a Quantum network. This operation might not be available for plugins in which the number of ports is fixed at network creation; in this case ports should not be deleted, just as they cannot be created.

The operation is not recoverable and will fail if an attachment is plugged into the port. #

This operation does not require a request body.

Example 4.27. Delete Port State Request/Response (XML)

Request:

```
DELETE tenants/33/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/
ports/98017ddc-efc8-4c25-a915-774b2a633855.xml
```

Response:

No data returned in response body.

Example 4.28. Create Port Request/Response (JSON)

Request:

```
DELETE tenants/33/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/
ports/98017ddc-efc8-4c25-a915-774b2a633855.json
```

Response:

No data returned in response body.

4.3. Attachments

This section describes the operations exposed by Quantum API for manipulating port attachments.

An attachment is typically a virtual network interface belonging to a VM instance. Different kinds of resources can be defined in the future.

4.3.1. Show Attachment for Port

Verb	URI	Description
GET	/tenants/*{tenant-id}*/networks/*{network-id}*/ports/ *{port-id}*/attachment	Returns the identifier of the attachment plugged into the specified port, identified by *port-id*.

Normal Response Code(s): 200

Error Response Code(s): Unauthorized (401), Forbidden (403), NetworkNotFound (420), PortNotFound (430)

This operation returns configuration details for the attachment plugged into the port specified in the request URI. This information is a reference to a virtual interface identifier.

If no attachment is currently plugged into the port, the operation does not return any attachment identifier in the response. The response will contain an empty `attachment` element with no `id` attribute set.

This operation does not require a request body.

Example 4.29. Show Attachment (XML)

Request:

```
GET /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/
b832be00-6553-4f69-af33-acd554e36d08/attachment.xml
```

Response (attachment set):

```
<attachment id="test_interface_identifier"/>
```

Response (attachment not set):

```
<attachment />
```

Example 4.30. Show Attachment (JSON)

Request:

```
GET /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/
b832be00-6553-4f69-af33-acd554e36d08/attachment.json
```

Response (attachment set):

```
{
 "attachment":
     {
         "id": "test_interface_identifier"
     }
}
```

Response (attachment not set):

```
{
 "attachment": {}
}
```

4.3.2. Plug Attachment into Port

Verb	URI	Description
PUT	/tenants/*{tenant-id}*/networks/*{network-id}*/ports/ *{port-id}*/attachment	Plugs a resource, i.e. a virtual network interface into the port identified by *port-id*.

Normal Response Code(s): 204

Error Response Code(s): Unauthorized (401), Forbidden (403), NetworkNotFound (420), PortNotFound (430), PortInUse (432), AlreadyAttached (440)

This operation plugs an attachment into the port specified in the request URL.

The request will be first validated by Quantum and then dispatched to the plugin. It is not guaranteed that the attached resource will be available as soon as the operation returns.

The validation can fail if:

- An attachment with the same identifier is already plugged somewhere else;
- There is already another attachment plugged into the specified port.

If the validation phase fails, the attachment is not created at all, and the appropriate error code is returned to the caller.

If no attachment is currently plugged into the port, the operation does not return any attachment identifier in the response. The response will contain an empty `attachment` element.

The request body for this network should contain a reference to the attachment to plug into the port.

Example 4.31. Plug Attachment (XML)

Request:

```
PUT /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/
b832be00-6553-4f69-af33-acd554e36d08/attachment.xml
```

```
<attachment
 id="test_interface_identifier"/>
```

Response:

No data returned in response body

Example 4.32. Plug Attachment (JSON)

Request:

```
PUT /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/
b832be00-6553-4f69-af33-acd554e36d08/attachment.json
```

```
{
 "attachment":
     {
         "id": "test_interface_identifier"
     }
}
```

Response:

No data returned in response body

4.3.3. Unplug Attachment from Port

Verb	URI	Description
DELETE	/tenants/*{tenant-id}*/networks/*{network-id}*/ports/ *{port-id}*/attachment	Removes the attachment currently plugged into the port identified by *port-id*.

Normal Response Code(s): 204

Error Response Code(s): Unauthorized (401), Forbidden (403), NetworkNotFound (420), PortNotFound (430)

This operation removes the attachment from the port specified in the request URI.

If no attachment is currently plugged into the port, this operation has no effect.

This operation does not require a request body

Example 4.33. Remove Attachment (XML)

Request:

```
DELETE /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/
b832be00-6553-4f69-af33-acd554e36d08/attachment.xml
```

Response:

No data returned in response body

Example 4.34. Remove Attachment (JSON)

Request:

```
DELETE /tenants/XYZ/networks/158233b0-ca9a-40b4-8614-54a4a99d47d1/ports/
b832be00-6553-4f69-af33-acd554e36d08/attachment.json
```

Response:

No data returned in response body