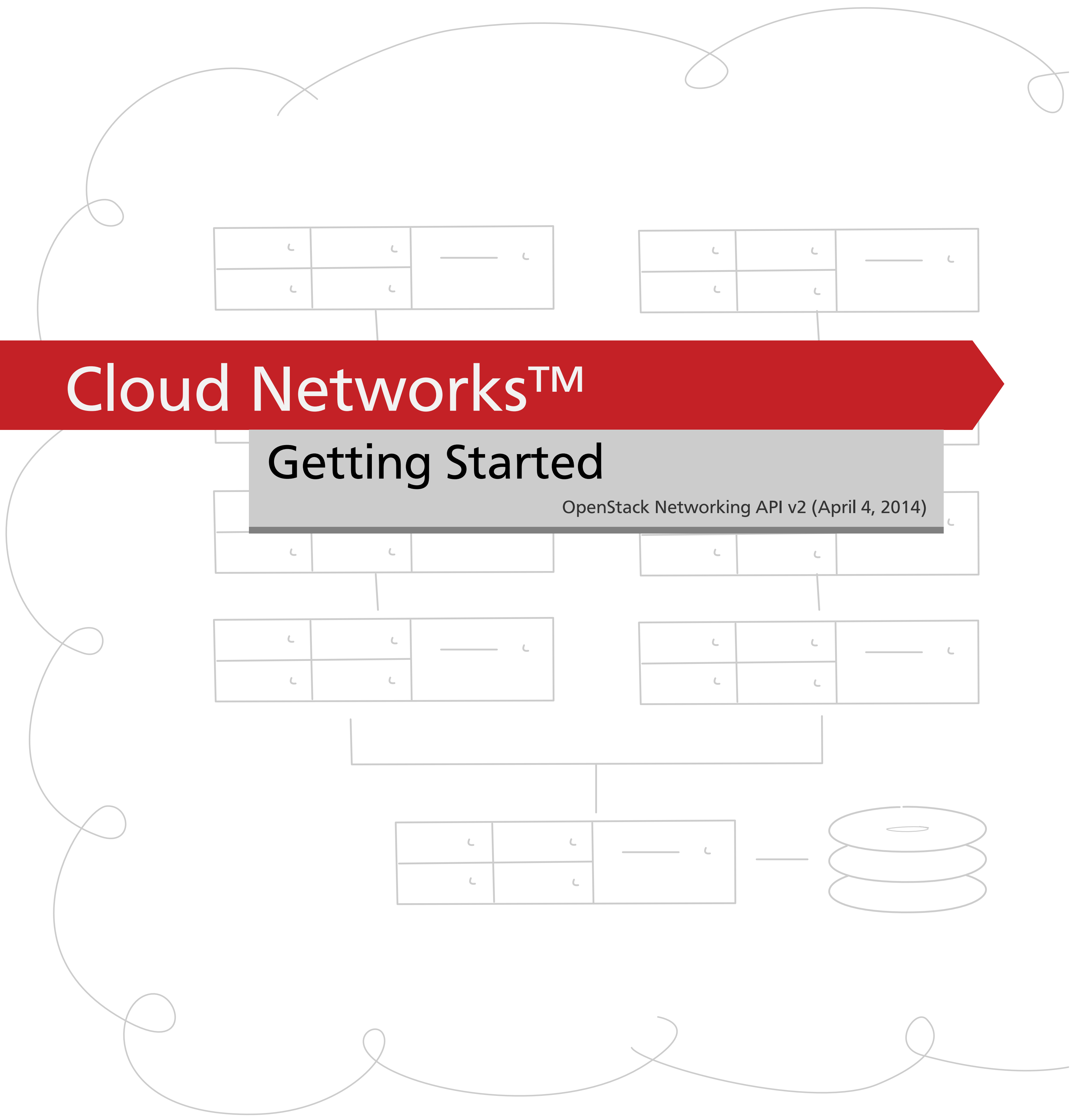


Cloud Networks™

Getting Started

OpenStack Networking API v2 (April 4, 2014)

rackspace®

the open cloud company

docs.rackspace.com/api

Cloud Networks™ Getting Started

OpenStack Networking API v2 (2014-04-04)
©2014 Rackspace, US Inc.

This document is for software developers who develop applications by using Rackspace Cloud Networks™, which is powered by the OpenStack Networking code base. In addition to the core features of the OpenStack Networking Application Programming Interface (API) v2, Rackspace has also deployed certain extensions as permitted by the OpenStack Networking API contract. The document is for informational purposes only and is provided “AS IS.”

Table of Contents

List of Tables

Preface

Rackspace Cloud Networks enables you to create isolated networks and provision *server* instances with Rackspace networks or the isolated networks that you created.

When you create an isolated network, it is associated with your tenant ID.

Cloud Networks was originally released using the Cloud Servers Nova-network (see API Operations - Nova in *Cloud Networks™ Developer Guide*). Now Cloud Networks has been released using Neutron (see API Operations - Neutron in *Cloud Networks™ Developer Guide*). Both sets of API operations work. For information that helps you determine which method is better for you, see “Networking: Nova-network versus Neutron” in *Cloud Networks™ Developer Guide*.

To request access to Cloud Networks now, click "I want Cloud Networks".

To detect if you are enabled to use Cloud Networks, issue a **GET /limits** call or a **nova absolute-limits** call and review the `maxTotalPrivateNetworks` value:

Table 1. maxTotalPrivateNetworks Value

maxTotalPrivateNetworks Value	Description
0	Cloud Networks is disabled.
Greater than 0	Cloud Networks is enabled. The value indicates the number of isolated networks that you are allowed to create.

For more information about querying limits, see Get Limits.

We welcome feedback, comments, and bug reports. Login to the Rackspace customer portal at http://www.rackspace.com/support/.

Intended Audience

This guide assists software developers who want to develop applications by using Cloud Servers and Cloud Networks.

To use this information, you should have access to an active Rackspace Cloud Servers account and access to Cloud Networks. You should also be familiar with the following concepts:

- Cloud Servers service
- *RESTful* web services
- *HTTP*/1.1
- JSON or XML data serialization formats

Pricing and Service Level

Next generation Cloud Networks is part of the Rackspace Cloud and your use through the API is billed according to the pricing schedule for Cloud Servers at http://www.rackspace.com/cloud/servers/pricing/.

The Service Level Agreement (SLA) for Cloud Networks, as part of Cloud Servers, is available at http://www.rackspace.com/cloud/servers/service-levels/.

Document Change History

This version of the guide applies only to v3.0 of the API. The most recent changes are described in the following table:

Revision Date	Summary of Changes
March 14, 2014	Reorganized doc and added neutron api calls
February 17, 2014	Removed nova install from git instructions.
February 13, 2014	Removed supernova install instructions and added supernova link to resources section.
December 4, 2013	Updated the name for Managed Cloud Service Level
November 4, 2013	Published Performance server information, previously limited access.
October 31, 2013	Fixed the link for requesting Cloud Networks access to http://www.iwantcloudnetworks.com/.
September 26, 2013	• updated examples for d2to1 in install and update nova client section. • removed v4-fixed-ip from the section called “Provision a Server with an Isolated Network with the nova Client” [50]
August 28, 2013	Removed references to bandwidth since the RAX extension is no longer implemented. It causes an unsustainable load on Nova. See the section called “Step 4. List Servers and Attached Networks” [55].
June 26, 2013	Added Server Key Pairs nova commands.
June 24, 2013	Added important note that user may choose any endpoint regardless of account origin.
June 20, 2013	Updated link to supernova git repository
June 18, 2013	Removed optional parameters from nova_summary.
June 17, 2013	Removed optional parameters, such as hint, block_device_mapping, availability_zone, and security_groups, from the section called “Step 3. Provision a Server with an Isolated Network” [48].
May 23, 2013	Updated "control panel" to "Control Panel."
May 13, 2013	Added the Cloud Networks virtual interface extension, which enables you to create virtual interfaces for, list virtual interfaces for, and delete virtual interfaces from existing server instances.
March 26, 2013	Updated to add the list servers call, which lists servers and their attached networks.
February 7, 2013	Corrected the JSON request to add missing slashes in the section called “Create an Isolated Network with cURL” [44].
January 8, 2013	Updated the authentication section with additional information about the token environment variable used in the cURL examples.
December 14, 2012	Added a link for the json.tool: json — JSON encoder and decoder.
December 4, 2012	Corrected the NOVA_RACK_AUTH environment variable to NOVA_RAX_AUTH.
November 14, 2012	Updated formatting in PDF.
November 7, 2012	Updated formatting of nova client and cURL examples.
November 1, 2012	Fixed formatting problems in code examples.
October 31, 2012	• Fixed the link for requesting Cloud Networks access to http://www.iwantcloudnetworks.com/.

Revision Date	Summary of Changes
	• Corrected an error in list_networks.
October 30, 2012	• Cloud Networks phased-release launch. • First edition of this guide.

Resources

Next generation Cloud Servers v2	Cloud Networks v2	Identity v2
• *Next Generation Cloud Servers Release Notes* • *Next Generation Cloud Servers Getting Started* • *Next Generation Cloud Servers Developer Guide*	• *Cloud Networks Release Notes* • *Cloud Networks Getting Started* • *Cloud Networks Developer Guide*	• *Cloud Identity Client Developer Guide v2.0*

For additional Cloud Servers and Cloud Networks service resources, see the Rackspace Cloud site, which provides related documents and links to Rackspace support channels including Knowledge Center articles, phone, chat, and tickets.

For product updates and announcements through Twitter, see http://twitter.com/rackspace.

For information about the supernova client, which is an unsupported wrapper for the nova client useful for managing multiple nova environments, see supernova client.

Overview

Cloud Networks enables you to create a virtual Layer 2 network, known as an isolated network, which gives you greater control and security when you deploy web applications.

When you create a next generation Cloud Server, Cloud Networks enables you to attach one or more networks to your server. You can attach an isolated network that you have created or a Rackspace network.

If you install the Cloud Networks virtual interface extension, you can create a virtual interface to a specified Rackspace or isolated network and attach that network to an existing server instance. You can also list virtual interfaces for and delete virtual interfaces from a server instance. For information about the Cloud Networks virtual interface extension, see "Cloud Networks Virtual Interface Extension" in *Cloud Networks™ Developer Guide*.

Cloud Networks enables you to attach one or more of the following networks to your server:

- ***PublicNet***. Provides access to the Internet, to Rackspace services (such as Cloud Monitoring, Managed Cloud Service Level support, RackConnect, and Cloud Backup), and to certain operating system updates.

  When you list networks through Cloud Networks, PublicNet is labeled `public`.

- ***ServiceNet***. Provides access to Rackspace services such as Cloud Files, Cloud Databases, and Cloud Backup, and to certain packages and patches through an internal-only, multi-tenant network connection within each Rackspace data center.

  When you list networks through Cloud Networks, ServiceNet is labeled `private`.

  You can use ServiceNet for communications among web servers, application servers, and database servers without incurring bandwidth charges. However, without an isolated network, you must apply security rules to protect data integrity. When you add or remove a server, you must update the security rules on individual servers to permit or deny connections from newly added servers or removed servers.

- **Isolated**. Enables you to deploy web applications on a virtual Layer 2 network that you create through Cloud Networks. An isolated network keeps your server separate from PublicNet, ServiceNet, or both. When you create a isolated network, it is associated with your tenant ID.

When you provision a new server, the networks that are attached to it depend on which of the following methods you use to provision it:

- **The Cloud Servers API**. You must specify the networks that you want to attach to your server. If you do not specify any networks, ServiceNet and PublicNet are attached by default. However, if you specify an isolated network, you must explicitly specify the

UUIDs for PublicNet and ServiceNet to attach these networks to your server. The UUID for ServiceNet is `11111111-1111-1111-1111-111111111111`, and the UUID for PublicNet is `00000000-0000-0000-0000-000000000000`.

- **The nova boot command**. You must specify the networks that you want to attach to your server. If you do not specify any networks, ServiceNet and PublicNet are attached by default. To attach to isolated networks that you have created, you must explicitly specify them in the command. If you do so, those networks, in addition to PublicNet and ServiceNet, are attached to your server.

- **The Cloud Control Panel**. PublicNet and ServiceNet are automatically attached, but you can disable these networks during the server creation process. You can also attach any isolated networks that you have created.

Note

You can explicitly opt out of attaching to the Rackspace networks, which introduces certain potential complications. For more information, see "Limitations of Detaching from Rackspace Networks" in *Cloud Networks™ Developer Guide*.

You can use Cloud Networks to perform the following tasks:

- List networks to which the specified tenant has access.

- Create isolated networks.

- Show details for isolated networks.

- Delete an isolated network, but only if it is not associated with any server.

  To detach a network from a server, you must use the Cloud Networks virtual interface extension to delete the virtual interface for the network from the server. See "Cloud Networks Virtual Interface Extension" in *Cloud Networks™ Developer Guide*.

- Manage subnets.

- Manage ports.

- Manage Layer-3 routers.

To list the networks that are attached to servers, issue a Cloud Servers List Servers operation. For more information, see List Servers in the *Cloud Servers Developer Guide*.

Networking: Nova-network versus Neutron

Rackspace first introduced networking services by using the Nova-network API based on OpenStack. Now we are providing the Neutron API, also based on OpenStack, which offers more functionality and flexibility than the Nova-network API. Both sets of APIs continue to work well, but the Neutron API will be the base for all the future networking services that Rackspace offers.

The Neutron API provides three primary, top-level resources (networks, ports, and subnets) and includes the ability to perform the following functions:

- Use all existing create, read, update, delete (CRUD) API operations for networks
- Attach and detach networks
- Self-service additional IP addresses
- Pass custom routes to servers
- Set the default gateway on cloud networks
- View quotas related to networks

Note

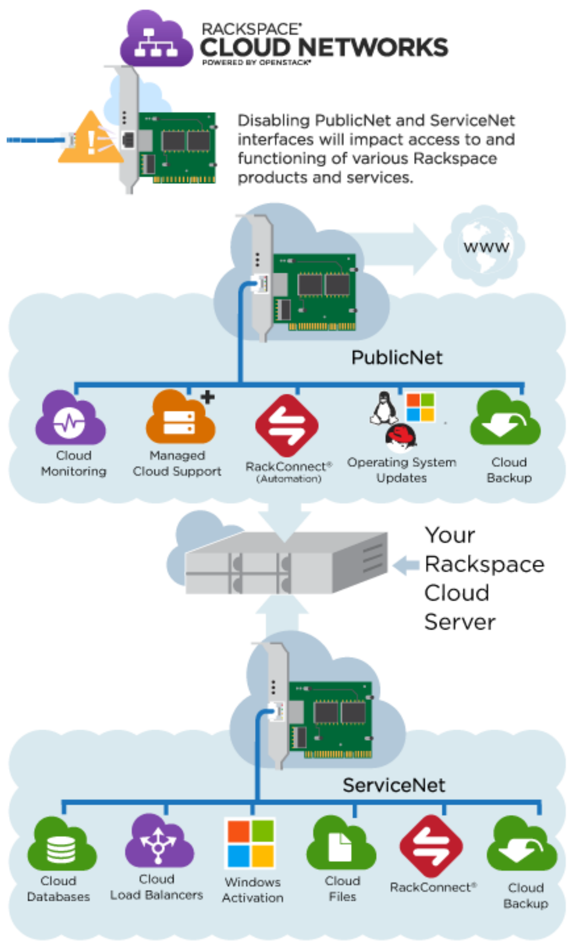
The Neutron API operations return only JSON-formatted responses. There is no option to receive responses in XML.

Limitations of Detaching from Rackspace Networks

When you provision a next generation Cloud Server, you can attach to one or more isolated networks and the Rackspace networks. You can also explicitly opt out of attaching the Rackspace networks to it, which introduces the following limitations:

- If you do not attach the PublicNet network to your server, the server cannot access to the Internet and some Rackspace products and services.
- If you do not attach the ServiceNet network to your server, the server cannot access certain Rackspace products and services.

The following graphic depicts the services that are available only when you attach the Rackspace networks to your server:



To opt out of attaching the Rackspace networks to your server, perform the following action depending on how you are provisioning the server:

- **nova boot command**. Specify the optional `--no-public` and `--no-service-net` parameters.
- **Cloud Servers API**. If you do not specify any networks, ServiceNet and PublicNet are attached by default. However, if you specify an isolated network, you must explicitly specify the UUIDs for PublicNet and ServiceNet to attach these networks to your server.

  The UUID for ServiceNet is `11111111-1111-1111-1111-111111111111`, and the UUID for PublicNet is `00000000-0000-0000-0000-000000000000`.

  Omit these UUIDs from the request to detach from these networks.
- **Cloud Control Panel**. Clear either or both of the PublicNet and ServiceNet selections during the server creation process. You are warned that your capabilities might be degraded by this choice.

Learn How to Use the Examples in This Guide

For each exercise in this guide, you choose one of the following methods to make Cloud Networks API calls:

- **The nova client**. We recommend that you use the nova client to run simple commands that make API calls. This open-source Python client runs on Linux or Mac OS X systems and is easy to learn and use. You can specify a debug parameter on any nova command to show the underlying API request for the command. This is a good way to become familiar with API requests.

  See the section called "How the nova Client Works" [8].

- **cURL commands**. If you are familiar with or want to learn cURL commands, choose this method. With cURL, you send HTTP requests with embedded API calls from the command line. The cURL examples in this guide show request and response bodies that are in JSON and XML formats. See the section called "How cURL Commands Work" [9]

If you like, you can use cURL in one example and switch to the nova client in the next example. If you choose to use a cURL command for any example, you must export the environment variables required to run the cURL examples. See the section called "Step 3. (Optional) Export Environment Variables to Run cURL Commands" [18].

Each example, whether nova client or cURL, points you to the next step in the process.

How the nova Client Works

The nova client is an open-source Python client that lets you run commands to make API calls. This client is easy to learn and use. Each nova client command runs cURL commands that embed API requests. You can run the nova client from a desktop machine or remote system, such as a Cloud Server.

You install the nova client on a Linux distribution or on Mac OS X. To install the nova client, see the section called "Step 1. (Optional) Install the nova Client with the Cloud Networks Extension" [14].

To configure the nova client, you update your bash profile to set and export nova environment variables.

Note

You can specify a debug parameter on any nova command to show the underlying API request for the command. This is a good way to become familiar with the API requests.

Warning

If you upgrade the operating system of the desktop or remote machine where you installed nova, you may need to reinstall nova.

How cURL Commands Work

cURL is a command-line tool that you can use to interact with *REST* interfaces. cURL lets you to transmit and receive *HTTP* requests and responses from the command line or a shell script, which enables you to work with the API directly. It is available for Linux distributions, Mac OS X, and Windows. For information about cURL, see http://curl.haxx.se/.

To use XML requests and responses, see the section called “XML Requests and Responses” [12].

To run the cURL request examples shown in this guide, copy each example from the HTML version of this guide directly to the command line or a script.

The following command is an example cURL command that provisions a server with an isolated network:

Example 1. cURL Command Example: JSON Request

```
            $ curl https://dfw.servers.api.rackspacecloud.com/v2/$account/servers \
       -X POST \
       -H "X-Auth-Project-Id: $account" \
       -H "Content-Type: application/json" \
       -H "Accept: application/json" \
       -H "X-Auth-Token: $token" \
       -d '{"server": {"name": "my_server_with_network", "imageRef": "d42f821e-c2d1-4796-9f07-
af5ed7912d0e", "flavorRef": "2", "max_count": 1, "min_count": 1, "networks": [{"uuid":
 "538a112a-34d1-47ff-bf1e-c40639e886e2"}, {"uuid": "00000000-0000-0000-0000-000000000000"}, {"uuid":
 "11111111-1111-1111-1111-111111111111"}]}}' \
      | python -m json.tool
```

Note

The carriage returns in the cURL request examples use a backslash (\) as an escape character. The escape character allows continuation of the command across multiple lines. However, do not include the escape character in the JSON or XML request body within the cURL command.

The cURL examples in this guide use the following command-line options:

Table 2. cURL Command-Line Options

Option	Description
`-d`	Sends the specified data in a **POST** request to the HTTP server. Use this option to send a JSON or XML request body to the server.

`-H`	Specifies an extra HTTP header in the request. You can specify any number of extra headers. Precede each header with the `-H` option.
	
	Common headers in Rackspace API requests are as follows:
	
	• `Content-Type`. Required for operations with a request body.
	
	Specifies the format of the request body. Following is the syntax for the header where *`format`* is either `json` or `xml`.
	
	`Content-Type: application/format`
	
	• `X-Auth-Token`. Required. Specifies the authentication token.
	
	• `X-Auth-Project-Id`. Optional. Specifies the project ID, which can be your account number or another value.
	
	• `Accept`. Optional.
	
	Specifies the format of the response body. Following is the syntax for the header where *`format`* is either `json` or `xml`. The default is `json`.
	
	`Accept: application/format`

`-i`	Includes the HTTP header in the output.
`-s`	Specifies silent or quiet mode, which makes cURL mute. No progress or error messages are shown.
`-T`	Transfers the specified local file to the remote URL.
`-X`	Specifies the request method to use when communicating with the HTTP server. The specified request is used instead of the default method, which is **GET**.

Note

For commands that return a response, you can append the following code to the command to call json.tool to pretty-print output:

```
| python -m json.tool
```

To use json.tool, import the json module. For information about json.tool, see json — JSON encoder and decoder.

If you run a Python version older than 2.6, import the simplejson module and use simplejson.tool. For information about simplejson.tool, see simplejson — JSON encoder and decoder.

If you do not want to pretty-print JSON output, omit this code.

XML Requests and Responses

The following example shows a cURL command that specifies an XML request body and returns an XML response. The command creates a server.

Example 2. cURL Command Example: XML Request

```
$ curl -i https://dfw.servers.api.rackspacecloud.com/v2/$account/servers.xml❶ \
     -X POST \
     -H "X-Auth-Project-Id: $account" \
     -H "Content-Type: application/xml" \  ❷
     -H "Accept: application/xml" \  ❸
     -H "X-Auth-Token: $token" \
     -T server_post_req.xml❹ | ppxml ❺
```

This example includes the following changes to the basic JSON request format in Example 5, "cURL Command Example: JSON Request" in *Cloud Networks™ Developer Guide*:

❶ The endpoint in the cURL command has `.xml` appended to it to return an XML response.

❷ The `Content-Type:` header specifies `application/xml` instead of `application/json`.

❸ The `Accept:` header specifies `application/xml` instead of `application/json`.

❹ The request body, if required, should be specified it in XML format. In this example, the XML body is passed in the `server_post_req.xml` file, as follows:

```
                    <?xml version="1.0" encoding="UTF-8"?>
<server xmlns="http://docs.openstack.org/compute/api/v1.1" imageRef="3afe97b2-26dc-49c5-a2cc-a2fc8d80c001"
 flavorRef="2"
  name="api-test-server-xml2">
  <metadata>
    <meta key="My Server Name">API Test Server XML</meta>
  </metadata>
  <personality>
    <file path="/etc/banner.txt"> ICAgICAgDQoiQSBjbG91ZCBkb2VzIG5vdCBrbm93IHdoeSBp
      dCBtb3ZlcyBpbiBqdXN0IHN1Y2ggYSBkaXJlY3Rpb24gYW5k IGF0IHN1Y2ggYSBzcGVlZC4uLkl0IGZlZWxzIGFuIGltcHVs
      c2lvbi4uLnRoaXMgaXMgdGhlIHBsYWNlIHRvIGdvIG5vdy4g QnV0IHRoZSBza3kga25vd3MgdGhlIHJlYXNvbnMgYW5kIHRo
      ZSBwYXR0ZXJucyBiZWhpbmQgYWxsIGNsb3VkcywgYW5kIHlv dSB3aWxsIGtub3csIHRvbywgd2hlbiB5b3UgbGlmdCB5b3Vy
      c2VsZiBoaWdoIGVub3VnaCB0byBzZWUgYmV5b25kIGhvcml6 b25zLiINCg0KLVJpY2hhcmQgQmFjaA==</file>
  </personality>
  <networks>
    <network uuid="0ef47ac7-6797-4e01-8a47-ed26ec3aaa56"/>
    <network uuid="00000000-0000-0000-0000-000000000000"/>
    <network uuid="11111111-1111-1111-1111-111111111111"/>
  </networks>
  <keypair>
    <key_name>name_of_keypair-96bbe50e-05e1-4d59-9115-4779a3ebcc2e</key_name>
  </keypair>
</server>
```

❺ To pretty-print the XML output, set the `ppxml` alias, as follows:

```
                  $ alias ppxml='python -c "import sys, xml.dom.minidom; print xml.dom.minidom.parseString(sys.stdin.
read()).toprettyxml()"'
```

Then, append the `ppxml` alias to the cURL command.

The cURL command returns information about the new server in XML format, as shown in the following example.

Example 3. cURL Command Example: XML Response

```
            <?xml version='1.0' encoding='UTF-8'?>
<server
    xmlns:OS-DCF="http://docs.openstack.org/compute/ext/disk_config/api/v1.1"
    xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns="http://docs.openstack.org/compute/api/v1.1"
    id="ed5c7754-29b6-45fa-96cb-ab64958c8c0a" adminPass="Dd5pNZtpVVQ3"
    OS-DCF:diskConfig="AUTO">
    <metadata/>
    <atom:link
        href="https://dfw.servers.api.rackspacecloud.com/v2/010101/servers/ed5c7754-29b6-45fa-96cb-
ab64958c8c0a"
        rel="self"/>
    <atom:link
        href="https://dfw.servers.api.rackspacecloud.com/010101/servers/ed5c7754-29b6-45fa-96cb-
ab64958c8c0a"
        rel="bookmark"/>
</server>
```

Set Up the Environment and Authenticate

Steps 1 and 2 walk you through the steps to set up your working environment. You can choose to use the nova client or cURL. Select the appropriate method to prepare for the exercises to create isolated networks and servers. Step 3 walks you through getting authenticated.

Step 1. (Optional) Install the nova Client with the Cloud Networks Extension

To manage your servers from the command line, you can use the open-source nova client application called `python-novaclient`.

To install the nova client, you first install the `rackspace-novaclient` package, which includes the nova client and the Rackspace extensions including the Cloud Networks extension.

If you want to use the Cloud Networks virtual interface extension, you must also install the `os_virtual_interfacesv2_python_novaclient_ext` package.

Note

The python-novaclient is developed and maintained by the OpenStack community. While we do not directly support python-novaclient, you can post in the comments below if you run into any difficulties.

Before you begin, install the following software:

Table 3. Nova Client Prerequisite Software

Prerequisite	Description
Python 2.6 or later	Currently, the nova client does not support Python 3.
setuptools package	Installed by default on Mac OS X. Many Linux distributions provide packages to make **setuptools** easy to install. Search your package manager for **setuptools** to find an installation package. If you cannot find one, download the **setuptools** package directly from http://pypi.python.org/pypi/setuptools.
pip package	To install the nova client on a Mac OS X or Linux system, use **pip** because it is easy and ensures that you get the latest version of the nova client from the Python Package Index. Also, it lets you update the package later on.

Prerequisite	Description

	Install **pip** through the package manager for your system:
	• Mac OS X
	`$ sudo easy_install pip`
	• Ubuntu
	`$ aptitude install python-pip`
	• Debian
	`$ aptitude install python-pip`
	• Fedora
	`$ yum install python-pip`
	• CentOS, or RHEL (from EPEL or another 3rd party repository)
	`$ yum install python-pip`

To install the nova client with Rackspace extensions:

1. **Install or update the nova client package with Rackspace extensions**

   The `rackspace-novaclient` package includes the nova client and Rackspace extensions, including the Cloud Networks extension.

   Run the following command for your operating system to install the `rackspace-novaclient` package:

   For Ubuntu, Debian, or OS X:

   ```
   $ sudo pip install rackspace-novaclient
   ```

   For RHEL, CentOS, or Fedora:

   ```
   $ sudo python-pip install rackspace-novaclient
   ```

   ### Note

   If you previously installed the `rackspace-novaclient` package, run the following command for your operating system to upgrade it:

   For Ubuntu, Debian, or OS X:

   ```
   $ sudo pip install --upgrade rackspace-novaclient
   ```

   For RHEL, CentOS, or Fedora:

```
                        $ sudo python-pip install --upgrade
rackspace-novaclient
```

Important

- If you previously installed the earlier `rax_networks_python_novaclient_ext` package, run the following command for your operating system to remove it:

  For Ubuntu, Debian, or OS X:

  ```
                              $ sudo pip uninstall
  rax_networks_python_novaclient_ext
  ```

  For RHEL, CentOS, or Fedora:

  ```
                              $ sudo python-pip uninstall
  rax_networks_python_novaclient_ext
  ```

- If you have trouble using pip to install the nova client, you can also download a nova client installation package from the python package repository. Or, you can use pip to individually install the `python-novaclient` and `rackspace-auth-openstack` packages.

2. **Set environment variables**

   Edit your bash profile to add and set environment variables that enable the nova client to connect to your Rackspace Cloud account. Use nano or an editor of your choice to edit your `.bash_profile` file:

   ```
   $ nano ~/.bash_profile
   ```

   Depending on your account location, add one of the following sets of lines to your bash profile and save the file:

   Example 4. US Data Center Example

   ```
   export OS_AUTH_URL=https://identity.api.rackspacecloud.com/v2.0/
   export OS_AUTH_SYSTEM=rackspace
   export OS_REGION_NAME=DFW
   export OS_USERNAME=<username>
   export OS_TENANT_NAME=<tenant_id>
   export NOVA_RAX_AUTH=1
   export OS_PASSWORD=<api_key>
   export OS_PROJECT_ID=<tenant_id>
   export OS_NO_CACHE=1
   ```

   Example 5. UK Data Center Example

   ```
   export OS_AUTH_URL=https://lon.identity.api.rackspacecloud.com/v2.0/
   export OS_AUTH_SYSTEM=rackspace_uk
   export OS_REGION_NAME=LON
   export OS_USERNAME=<username>
   ```

```
export OS_TENANT_NAME=<tenant_id>
export NOVA_RAX_AUTH=1
export OS_PASSWORD=<apikey>
export OS_PROJECT_ID=<tenant_id>
export OS_NO_CACHE=1
```

3. **Set permissions on the bash profile**

   Because the bash profile contains a password, set permissions on it so other people cannot read it:

   ```
   $  chmod 600 ~/.bash_profile
   ```

4. **Source the environment variables**

   To source the variables to make them available in your current shell, run the following command:

   ```
   $ source ~/.bash_profile
   ```

5. **Test the nova client**

   To verify that you can talk to the API server, authenticate and list images:

   ```
   $ nova credentials
   $ nova image-list
   ```

   Then, list networks:

   ```
   $ nova network-list
   ```

6. **Get help for nova client commands**

   To get help for nova client commands, run the following command:

   ```
   $ nova help
   ```

   To get help for a specific command, type the command name after the *help* parameter, as follows:

   ```
   $ nova help <command_name>
   ```

   You cannot use every command that is listed. The nova client was written for use with recent development versions of OpenStack, so it includes support for some features that are not available in the Rackspace Cloud.

   Note

   To show and become familiar with the underlying API v2 request for a command, specify the `--debug` parameter as the first parameter on any command. For example:

   ```
   $ nova --debug list
   ```

7. **Troubleshooting**

   The python-novaclient is developed and maintained by the OpenStack community.

- If you cannot run commands successfully, make sure that you entered your user name, password, and account number correctly in the bash profile file.
- If you change any environment variables, either log out and back in or source your bash profile again.
- To override some environment variable settings, you can use the options that are listed at the end of the **nova help** output. For example, you can override the `OS_PASSWORD` setting in the bash profile by specifying a password on a nova command, as follows:

  ```
                      $ nova --password <password> image-list
  ```

  Where *`password`* is your Rackspace password.
- While Rackspace does not directly support python-novaclient, if the above steps don't help, you can post a question in the comments below.

Next step:the section called “Step 2. (Optional) Install the neutron Client with the Cloud Networks Extension” [18]

Step 2. (Optional) Install the neutron Client with the Cloud Networks Extension

Follow the necessary steps to install the neutron client (TBD).

Next step:the section called “Step 3. (Optional) Export Environment Variables to Run cURL Commands” [18]

Step 3. (Optional) Export Environment Variables to Run cURL Commands

To run the cURL command examples in this guide, follow this procedure to export environment variables.

Export your tenant ID and authentication token to environment variables. All cURL examples in this guide use these environment variables.

To export environment variables:

1. Export your tenant ID to the `account` environment variable:

   ```
   $ export account="<tenantId>"
   ```

   Where *`tenantId`* is the value in the `tenantId` field in the authentication response.

2. Export your authentication token to the `token` environment variable:

```
$ export token="<id>"
```

Where *`id`* is the authentication token value in the `id` field in the `token` element in the authentication response.

Next step: the section called “Step 4. Get Authentication Token” [19]

Step 4. Get Authentication Token

To authenticate access to Rackspace Cloud services, you issue an authentication request to the Rackspace Cloud Identity Service, which is an implementation of the OpenStack Keystone Identity Service v2.0.

Important

Multiple Rackspace Cloud Identity Service endpoints exist. You may use any endpoint, regardless of where your account was created.

Use the chosen endpoint, as follows:

National location	Rackspace Cloud Identity Service endpoint
US	https://identity.api.rackspacecloud.com/v2.0
UK	https://lon.identity.api.rackspacecloud.com/v2.0

In response to valid credentials, an authentication request to the Rackspace Cloud Identity Service returns an authentication token and a service catalog that contains a list of all services and endpoints available for this token. Because the authentication token expires after 24 hours, you must generate a token once a day.

For detailed information about the OpenStack Keystone Identity Service v2.0, see *Cloud Identity Client Developer Guide API v2.0*. For information about support for legacy identity endpoints, see Alternate Authentication Endpoints.

Next step: Choose one of the following methods:

- Authenticate with the nova client
- Authenticate with cURL

Authenticate with the nova Client

To authenticate with the nova client:

1. Get an authentication token:

   ```
   $ nova credentials
   ```

   Successful authentication returns user credentials, including ID, name, roles, and the authentication token. The token appears in the `id` field in the `Token` box.

   ```
   +------------------
   +---------------------------------------------------------------------------------------
   +
   | User Credentials | Value
                                    |
   +------------------
   +---------------------------------------------------------------------------------------
   +
   | id               | 170454
                                    |
   | name             | MyRackspaceAcct
                                    |
   | roles            | [{u'description': u'User Admin Role.', u'id': u'3',
    u'name': u'identity:user-admin'}] |
   +------------------
   +---------------------------------------------------------------------------------------
   +
   +---------+----------------------------------------+
   | Token   | Value                                  |
   +---------+----------------------------------------+
   | expires | 2012-07-28T13:58:56.000-05:00          |
   | id      | 1bd336d5-e0c6-49d9-b902-d3dbdc463062   |
   | tenant  | {u'id': u'010101', u'name': u'010101'} |
   +---------+----------------------------------------+
   ```

   After you generate a token, the nova client automatically injects the token into any nova client commands that you issue.

   However, because the token expires after 24 hours, you must generate a new token once a day.

2. Get the service catalog with a list of endpoints:

   ```
   $ nova endpoints
   ```

   For each service, the response includes the public URL, which is the endpoint that you use to access the service, the region, service name, tenant ID, the version ID, and endpoints that you can use to get version information for the API.

   To access the Cloud Networks or next generation Cloud Servers service, use the `publicURL` value for the `cloudServersOpenStack` service.

   The following output shows the information returned for the DFW region for the Cloud Networks or next generation Cloud Servers service:

```
+-----------------------
+------------------------------------------------------+
| cloudServersOpenStack | Value
     |
+-----------------------
+------------------------------------------------------+
| publicURL             | https://dfw.servers.api.rackspacecloud.com/v2/
010101 |
| region                | DFW
     |
| serviceName           | cloudServersOpenStack
     |
| tenantId              | 010101
     |
| versionId             | 2
     |
| versionInfo           | https://dfw.servers.api.rackspacecloud.com/v2
     |
| versionList           | https://dfw.servers.api.rackspacecloud.com/
     |
+-----------------------
+------------------------------------------------------+
```

The `cloudServersOpenStack` service might show multiple endpoints to enable regional choice. Select the appropriate endpoint for the region that you want to interact with by examining the `region` field.

3. Copy the values in the `publicURL` and `tenantId` fields for the `cloudServersOpenStack` service for your region.

Next step: Create Your First Cloud Network [27]

Authenticate with cURL

To authenticate, you issue a **POST /tokens** request to the appropriate Rackspace Cloud Identity Service endpoint.

In the request body, supply one of the following sets of credentials:

- user name and password
- user name and API key

Your username and password are the ones that you use to log into the Rackspace Cloud Control Panel.

To obtain your API key, log into http://mycloud.rackspace.com, click your username, and select **API Keys** to get your key.

To authenticate with cURL:

1. Get an authentication token and a service catalog with a list of endpoints.

   Use one of the following cURL commands to authenticate to the US Identity endpoint:

   #### Example 6. Authenticate with Username and Password: JSON Request

```
$ curl -s https://identity.api.rackspacecloud.com/v2.0/tokens -X 'POST' \
       -d '{"auth":{"passwordCredentials":{"username":"theUserName", "password":"thePassword"}}}' \
       -H "Content-Type: application/json" | python -m json.tool
```

   #### Example 7. Authenticate with Username and API Key: JSON Request

```
$ curl -s https://identity.api.rackspacecloud.com/v2.0/tokens -X 'POST' \
       -d '{"auth":{"RAX-KSKEY:apiKeyCredentials":{"username":"theUserName", "apiKey":"00000000000000000000"}}}' \
       -H "Content-Type: application/json" | python -m json.tool
```

   Note

   For information about the json.tool, see Note [10].

   In response to valid credentials, your request returns an authentication token and a service catalog with the endpoints to request services.

   Do not include explicit API endpoints in your scripts and applications. Instead, find the endpoint for your service and region.

   The following output shows a partial authentication response in JSON format:

   #### Example 8. Authenticate: JSON Response

```
{
    "access": {
        "serviceCatalog": [
            {
                "endpoints": [
                    {
                        "publicURL": "https://ord.loadbalancers.api.rackspacecloud.com/v1.0/010101",
                        "region": "ORD",
                        "tenantId": "010101"
                    },
                    {
                        "publicURL": "https://dfw.loadbalancers.api.rackspacecloud.com/v1.0/010101",
                        "region": "DFW",
                        "tenantId": "010101"
                    }
                ],
                "name": "cloudLoadBalancers",
                "type": "rax:load-balancer"
            },
            {
                "endpoints": [
                    {
                        "publicURL": "https://monitoring.api.rackspacecloud.com/v1.0/010101",
                        "tenantId": "010101"
                    }
                ],
                "name": "cloudMonitoring",
                "type": "rax:monitor"
            },
            {
                "endpoints": [
                    {
                        "publicURL": "https://preprod.dfw.servers.api.rackspacecloud.com/v2/010101",
                        "region": "DFW",
                        "tenantId": "010101"
                    }
                ],           ❶
                "name": "cloudServersPreprod",
                "type": "compute"
            },
            {                               ❷
                "endpoints": [
                    {
                        "internalURL": "https://snet-storage101.dfw1.clouddrive.com/v1/
MossoCloudFS_530f8649-324c-499c-a075-2195854d52a7",
                        "publicURL": "https://storage101.dfw1.clouddrive.com/v1/MossoCloudFS_530f8649-324c-499c-
a075-2195854d52a7",
                        "region": "DFW",
                        "tenantId": "MossoCloudFS_530f8649-324c-499c-a075-2195854d52a7"
                    }
                ],
                "name": "cloudFiles",
                "type": "object-store"
            },
            {
                "endpoints": [
                    {                           ❸
                        "publicURL": "https://servers.api.rackspacecloud.com/v1.0/010101",
                        "tenantId": "010101",
                        "versionId": "1.0",
                        "versionInfo": "https://servers.api.rackspacecloud.com/v1.0",
                        "versionList": "https://servers.api.rackspacecloud.com/"
                    }                                  ❹
                ],                                      ❺
                "name": "cloudServers",
                "type": "compute"
            },
            {
                "endpoints": [
                    {
                        "publicURL": "https://dfw.servers.api.rackspacecloud.com/v2/010101",
                        "region": "DFW",
                        "tenantId": "010101",
                        "versionId": "2",
                        "versionInfo": "https://dfw.servers.api.rackspacecloud.com/v2",
                        "versionList": "https://dfw.servers.api.rackspacecloud.com/"
                    },
                    {
                        "publicURL": "https://ord.servers.api.rackspacecloud.com/v2/010101",
                        "region": "ORD",
                        "tenantId": "010101",
                        "versionId": "2",
                        "versionInfo": "https://ord.servers.api.rackspacecloud.com/v2",
                        "versionList": "https://ord.servers.api.rackspacecloud.com/"
                    }
                ],
                "name": "cloudServersOpenStack",
                "type": "compute"
            },
            {
                "endpoints": [
                    {
                        "publicURL": "https://dns.api.rackspacecloud.com/v1.0/010101",
                        "tenantId": "010101"
                    }
                ],
                "name": "cloudDNS",
                "type": "rax:dns"
            },
            {
                "endpoints": [
                    {
                        "publicURL": "https://ord.databases.api.rackspacecloud.com/v1.0/010101",
                        "region": "ORD",
                        "tenantId": "010101"
                    },
                    {
                        "publicURL": "https://dfw.databases.api.rackspacecloud.com/v1.0/010101",
                        "region": "DFW",
                        "tenantId": "010101"
                    }
                ],
                "name": "cloudDatabases",
                "type": "rax:database"
            },
            {
                "endpoints": [
                    {
                        "publicURL": "https://cdn1.clouddrive.com/v1/MossoCloudFS_530f8649-324c-499c-
a075-2195854d52a7",
                        "region": "DFW",
                        "tenantId": "MossoCloudFS_530f8649-324c-499c-a075-2195854d52a7"
                    }
                ],
                "name": "cloudFilesCDN",
                "type": "rax:object-cdn"
            }
        ],
        "token": {
            "expires": "2012-08-14T12:31:16.000-05:00",
            "id": "459a28e0-777f-416c-8f22-9f6598fabd2f",
            "tenant": {
                "id": "010101",
                "name": "010101"
            }
        },
        "user": {
            "id": "01010156",
            "name": "MyRackspaceAcct",
            "roles": [
                {
                    "description": "User Admin Role.",
                    "id": "3",
                    "name": "identity:user-admin"
                }
            ]
        }
    }
}
```

Successful authentication returns the following information:

❶ **Endpoints to request Rackspace Cloud services**. Appears in the `endpoints` element in the `serviceCatalog` element.

Endpoints information includes the public URL, which is the endpoint that you use to access the service, region, tenant ID, and version information.

To access the Cloud Networks or next generation Cloud Servers service, use the endpoint for the `cloudServersOpenStack` service.

❷ **Tenant ID**. Appears in the `tenantId` field in the `endpoints` element. Also known as the account number.

You include the tenant ID in the endpoint URL when you call a Cloud service.

In the following example, you export the tenant ID, `010101`, to the `account` environment variable and the authentication token to the `token` environment variable. Then, you issue a cURL command, as follows:

```
$ export account="010101"
$ export token="00000000-0000-0000-000000000000"
$ curl -s https://dfw.servers.api.rackspacecloud.com/v2/$account/images/detail \
     -H "X-Auth-Token: $token" | python -m json.tool
```

❸ **The name of the service**. Appears in the `name` field.

Locate the correct service name in the service catalog, as follows:

- **First generation Cloud Servers**. Named `cloudServers` in the catalog.

If you use the authentication token to access this service, you can view and perform first generation Cloud Servers API operations against your first generation Cloud Servers.

- **Cloud Networks or next generation Cloud Servers**. Named `cloudServersOpenStack` in the catalog.

  To access the Cloud Networks or next generation Cloud Servers service, use the `publicURL` value for the `cloudServersOpenStack` service.

  Might show multiple endpoints to enable regional choice. Select the appropriate endpoint for the region that you want to interact with by examining the `region` field.

  If you use the authentication token to access this service, you can view and perform Cloud Networks or next generation Cloud Servers API operations against your next generation Cloud Servers. To complete Cloud Networks API operations, you must also get access to this service. To request access, click here.

❹ **Expiration date and time for authentication token**. Appears in the `expires` field in the `token` element.

After this date and time, the token is no longer valid.

This field predicts the maximum lifespan for a token, but does not guarantee that the token reaches that lifespan.

Clients are encouraged to cache a token until it expires.

Because the authentication token expires after 24 hours, you must generate a token once a day.

❺ **Authentication token**. Appears in the `id` field in the `token` element.

You pass the authentication token in the `X-Auth-Token` header each time that you send a request to a service.

In the following example, you export the tenant ID, `010101`, to the `account` environment variable. You also export the authentication token, `00000000-0000-0000-000000000000`, to the `token` environment variable. Then, you issue a cURL command, as follows:

```
$ export account="010101"
$ export token="00000000-0000-0000-000000000000"
$ curl -s https://dfw.servers.api.rackspacecloud.com/v2/$account/images/detail \
     -H "X-Auth-Token: $token" | python -m json.tool
```

2. Copy the values in the `publicURL` and `tenantId` fields for the `cloudServersOpenStack` service for your region.

   Copy the authentication token from the `id` field in the `token` element.

   In the next step, you set environment variables to these values.

Next step: Create Your First Cloud Network [27]

Create Your First Cloud Network

Before creating your first network, choose whether to use the Neutron API or the nova-network API. For help deciding, see the section called "Networking: Nova-network versus Neutron" [5]. The sections in this chapter provide examples of creating a network using each API.

Create Your First Cloud Network Using Neutron

These examples, using the Neutron API, walk you through the steps to create an isolated network and create a server that is attached your isolated network, showing you how to access the Cloud Networks API through neutron client commands or cURL commands so that you can quickly create isolated networks, subnets, and servers.

The exercises also help you learn how cURL commands and the Cloud Networks neutron and nova-network APIs work.

High-Level Task Flow

The high-level task flow for Cloud Networks involves creating a network, associating a subnet with that network, and booting a server that is attached to the network. Network cleanup includes deleting the server, deleting any ports associated with the network, and deleting the networks. Cloud Networks automatically deletes any subnets associated with the deleted network.

To Use Cloud Networks: A High Level Flow

1. **Create a network.**

   The tenant (user) creates a network.

   For example, the tenant creates the `net1` network. Its ID is `net1_id`.

2. **Associate a subnet with the network.**

   The tenant associates a subnet with that network.

   For example, the tenant associates the 10.0.0.0/24 subnet with the `net1` network.

3. **Boot (or create) a server and attach it to the network.**

   The tenant boots a server and specifies a single NIC that connects to the network.

   The following examples use the nova client to boot a server.

   In the following example, nova contacts Cloud Networks to create the NIC and attach it to the `net1` network, with the ID `net1_id`:

   ```
   $ nova boot serverName --image image --flavor flavor --nic net-id=net1_id
   ```

In the following example, nova first creates the `port1` port, and then boots the server with a specified port. Cloud Networks creates a NIC and attaches it to the port `port1`, with the ID `port1_id`:

```
$ nova boot serverName --image image --flavor flavor --nic port-
id=port1_id
```

Cloud Networks chooses and assigns an IP address to the `port1` port.

4. **Delete the server.**

   The tenant deletes the server.

   The Cloud Servers service contacts the Cloud Networks service and deletes the `port1` port.

   Upon deletion, the allocated IP address is returned to the pool of available IP addresses.

5. **Delete any ports.**

   If the tenant created any ports and associated them with the network, the tenant deletes the ports.

6. **Delete the network.**

   The tenant deletes the network. This operation deletes a Cloud Networks network and its associated subnets, provided that no port is currently configured on the network.

Step 1. Create a Network and a Sub-Network

Before you create a server, you create a network, and an optional subnet, that you can attach to your new server.

To create a network, you specify a name and an admin_state for your network. To create a subnet, you specify a network, an IP address, and a name for your subnet.

After you create a network, copy its network ID. You use this ID to create a subnet or attach the network to a server. After you create a subnet, copy its subnet ID for future reference.

Next step: Choose one of the following methods:

- the section called “Create a Network and Subnet with the neutron Client” [28]
- the section called “Create a Network and Subnet with cURL” [29]

Create a Network and Subnet with the neutron Client

To create a network with the neutron client

1. Issue the following neutron command:

```
$ neutron net-create <amp>
```

Positional arguments:

- The network name. In this example, the name is `amp`.

Output:

The command returns the response, as shown in the following output:

```
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| admin_state_up | True                                 |
| id             | 29f52c7e-6efd-4335-a14a-db77d32a2555 |
| name           | amp                                  |
| shared         | False                                |
| status         | ACTIVE                               |
| subnets        |                                      |
| tenant_id      | 53501b3c25d34f8ea293c03298caed60     |
+----------------+--------------------------------------+
```

2. Copy the `id` value from the output. You will use this value when you create a subnet, provision your server, or perform other related activities.

To create a subnet and request a JSON response

1. Issue the following cURL command:

```
                            $ curl -s https://dfw.servers.api.rackspacecloud.com/v2.0/
$account/subnets \
       -X POST \
       -H "Content-Type: application/json" \
       -H "Accept: application/json" \
       -H "X-Auth-Token: $token" \
       -d '{"subnets":[{"cidr":"120.1.2.0/24","ip_version":4,
"network_id":"29f52c7e-6efd-4335-a14a-db77d32a2555"}]}' \
       | python -m json.tool
```

In this cURL command, you specify several options. For information about options, see Table 2, "cURL Command-Line Options" [10].

Output:

The operation returns the response body, as shown in the following JSON response:

2. Copy the `id` value from the output for future reference.

Next step:the section called "Step 2. List Networks and Subnets" [32]

Create a Network and Subnet with cURL

You can request a JSON or XML response when you issue a request in a cURL command.

JSON Request and Response

To create a network and request a JSON response

1. Issue the following cURL command:

```
                            $ curl -s https://dfw.servers.api.rackspacecloud.com/v2.0/
networks \
       -X POST \
       -H "Content-Type: application/json" \
       -H "Accept: application/json" \
       -H "X-Auth-Token: $token" \
       -d '{"network": {"name": "sample_network", "admin_state_up": "false"}}' | python -m
 json.tool
```

In this cURL command, you specify several options. For information about options, see Table 2, "cURL Command-Line Options" [10].

Output:

The operation returns the response body, as shown in the following JSON response:

```
                          {
   "network":{
      "status":"ACTIVE",
      "subnets":[

      ],
      "name":"sample_network",
      "provider:physical_network":null,
      "admin_state_up":true,
      "tenant_id":"4fd44f30292945e481c7b8a0c8908869",
      "provider:network_type":"local",
      "shared":false,
      "id":"baed79dd-9136-4260-b9a9-d9dfa2bf6547",
      "provider:segmentation_id":null
   }
}
```

2. Copy the `id` value from the output. You will use this value when you create a subnet, provision your server, or perform other related activities.

To create a subnet and request a JSON response

1. Issue the following cURL command:

```
                            $ curl -s https://dfw.servers.api.rackspacecloud.com/v2.0/
$account/subnets \
       -X POST \
       -H "Content-Type: application/json" \
       -H "Accept: application/json" \
       -H "X-Auth-Token: $token" \
       -d '{"subnets":[{"cidr":"120.1.2.0/24","ip_version":4,
"network_id":"29f52c7e-6efd-4335-a14a-db77d32a2555"}]}' \
       | python -m json.tool
```

In this cURL command, you specify several options. For information about options, see Table 2, "cURL Command-Line Options" [10].

Output:

The operation returns the response body, as shown in the following JSON response:

2. Copy the `id` value from the output for future reference.

> **Next step:** the section called “Step 2. List Networks and Subnets” [32]

XML Request and Response

To create a network and request a XML response

1. Issue the following cURL command:

```
                            $ curl -i https://dfw.servers.api.rackspacecloud.com/v2.0/
networks.xml \
          -X POST -H "X-Auth-Project-Id: $account" -H "Content-Type: application/xml" \
          -H "Accept: application/xml" \
          -H "X-Auth-Token: $token" \
          -d 'network name="sample_network" admin_state_up="false"'
```

Output:

The operation returns the response body, as shown in the following 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>
```

2. Copy the `id` value from the output. You will use this value when you create a subnet, provision your server, or perform other related activities.

To create a subnet and request a XML response

1. Issue the following cURL command:

```
                            $ curl -s https://dfw.servers.api.rackspacecloud.com/v2.0/
$account/subnets \
          -X POST -H "X-Auth-Project-Id: $account" -H "Content-Type: application/xml" \
          -H "Accept: application/xml" \
          -H "X-Auth-Token: $token" \
          -d '{"subnets":[{"cidr":"120.1.2.0/24","ip_version":4,
"network_id":"29f52c7e-6efd-4335-a14a-db77d32a2555"}]}' \
       | python -m json.tool
```

Output:

The operation returns the response body, as shown in the following XML response:

2. Copy the `id` value from the output for future reference.

Next step:the section called “Step 2. List Networks and Subnets” [32]

Step 2. List Networks and Subnets

After you create networks or subnets, you may want to view your networks, subnets, and the Rackspace networks.

For information about the Rackspace networks, see the section called “Limitations of Detaching from Rackspace Networks” [6].

Next step: Choose one of the following methods:

- the section called “List Networks and Subnets with the neutron Client” [32]
- the section called “List Networks and Subnets with cURL” [34]

List Networks and Subnets with the neutron Client

To list networks with the neutron client

- Issue the following neutron client command:

```
$ neutron net-list
```

Output:

For each network, the command returns the result as shown in the following output

```
+--------------------------------------+---------
+-----------------------------------------------------+
| id                                   | name     | subnets
                              |
+--------------------------------------+---------
+-----------------------------------------------------+
| 29f52c7e-6efd-4335-a14a-db77d32a2555 | amp      |
 23e3059e-4f39-4f7f-8cf2-c326e5de6c37 120.1.2.0/24    |
+--------------------------------------+---------
+-----------------------------------------------------+
```

To list subnets with the neutron client

- Issue the following neutron client command:

```
$ neutron subnet-list
```

Output:

For each nsubnet, the command returns the result as shown in the following output

```
+--------------------------------------+----------------+----------------
+--------------------------------------------------+
| id                                   | name           | cidr           |
 allocation_pools                                  |
+--------------------------------------+----------------+----------------
+--------------------------------------------------+
| 23e3059e-4f39-4f7f-8cf2-c326e5de6c37 | Subamp         | 120.1.2.0/24   |
 {"start": "120.1.2.2", "end": "120.1.2.254"}      |
+--------------------------------------+----------------+----------------
+--------------------------------------------------+
```

Next step: the section called “Step 3. Show Network and Subnet Information” [35]

List Networks and Subnets with cURL

You can request a JSON or XML response when you issue a request in a cURL command.

JSON Request and Response

To list networks and request a JSON response

- Issue the following cURL command:

```
                        $ curl https://dfw.servers.api.rackspacecloud.com/v2.0/
$account/subnets \
       -X GET -H "X-Auth-Token: $token" | python -m json.tool
```

Output:

The API operation returns a response body as shown in the following output:

To list subnets and request a JSON response

- Issue the following cURL command:

```
                        $ curl https://dfw.servers.api.rackspacecloud.com/v2.0/
$account/subnets \
       -X GET -H "X-Auth-Token: $token" | python -m json.tool
```

Output:

The API operation returns a response body as shown in the following output:

Next step: the section called “Step 3. Show Network and Subnet Information” [35]

XML Request and Response

To list networks and request an XML response

- Issue the following cURL command:

```
                        $curl -i https://dfw.servers.api.rackspacecloud.com/v2.0/
$account/networks.xml \
        -H "Content-Type: application/xml" \
        -H "Accept: application/xml" \
        -X GET -H "X-Auth-Token: $token"
```

Output:

The API operation returns a response body as shown in the following output:

```
                            <?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:router="http://docs.openstack.org/ext/neutron/router/api/v1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <network>
        <status>ACTIVE</status>
        <subnets>
            <subnet>54d6f61d-db07-451c-9ab3-b9609b6b6f0b</subnet>
        </subnets>
        <name>private-network</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">True</router:external>
        <shared quantum:type="bool">True</shared>
        <id>d32019d3-bc6e-4319-9c1d-6722fc136a22</id>
        <provider:segmentation_id xsi:nil="true"/>
    </network>
    <network>
        <status>ACTIVE</status>
        <subnets>
            <subnet>08eae331-0402-425a-923c-34f7cfe39c1b</subnet>
        </subnets>
        <name>private</name>
        <provider:physical_network xsi:nil="true"/>
        <admin_state_up quantum:type="bool">True</admin_state_up>
        <tenant_id>26a7980765d0414dbc1fc1f88cdb7e6e</tenant_id>
        <provider:network_type>local</provider:network_type>
        <router:external quantum:type="bool">True</router:external>
        <shared quantum:type="bool">True</shared>
        <id>db193ab3-96e3-4cb3-8fc5-05f4296d0324</id>
        <provider:segmentation_id xsi:nil="true"/>
    </network>
</networks>
```

ServiceNet is labeled as `private` and PublicNet is labeled as `public` in the network list.

Next step: the section called “Step 3. Show Network and Subnet Information” [35]

Step 3. Show Network and Subnet Information

You can show details for a specified network, or subnet, using the ID.

The response body returns the following fields:

Table 4. Show Network Response Fields

Name	Description
status	The status for an isolated private network.
subnets	An array of subnets for an isolated private network.
name	The name of an isolated private network.
admin_state_up	The status for an isolated private network.
tenant_id	The tenant_id of the network owner.
id	The network ID.
shared	Indicates whether this network is shared across all tenants.

Next step: Choose one of the following methods:

- the section called “Show Network and Subnet Information with the neutron Client” [37]
- the section called “Show Network and Subnet Information with cURL” [38]

Show Network and Subnet Information with the neutron Client

To show information for a specified network ID

- Issue the following neutron client command:

```
$ neutron net-details <id>
```

Positional argument:

- The ID of the network for which you want to show information. To determine the network ID, issue a **neutron** `net-list` command.

Output:

The command returns the result as shown in the following output:

To show information for a specified subnet ID

- Issue the following neutron client command:

```
$ neutron subnet-details <id>
```

Positional argument:

- The ID of the subnet for which you want to show information. To determine the subnet ID, issue a **neutron** `subnet-list` command.

Output:

The command returns the result as shown in the following output:

Next step:the section called “Step 6. Delete Your Network” [62]

Show Network and Subnet Information with cURL

You can request a JSON or XML response when you issue a request in a cURL command.

JSON Request and Response

To show information in a JSON response for a specified network ID

- Issue the following cURL command:

```
                            $ curl https://dfw.servers.api.rackspacecloud.com/v2.0/$account/
networks/{id} \
        -X GET \
        -H "X-Auth-Project-Id: $account" \
        -H "Accept: application/json" \
        -H "Content-Type: application/json"\
        -H "X-Auth-Token: $token" | python -m json.tool
```

Positional argument:

- *id*. The ID of the network for which you want to show information. To determine the network ID, issue a **neutron** `network-list` command.

Output:

The API operation returns a response body as shown in the following output:

```
                        {
   "network":{
      "status":"ACTIVE",
      "subnets":[
         "54d6f61d-db07-451c-9ab3-b9609b6b6f0b"
      ],
      "name":"private-network",
      "provider:physical_network":null,
      "admin_state_up":true,
      "tenant_id":"4fd44f30292945e481c7b8a0c8908869",
      "provider:network_type":"local",
      "router:external":true,
      "shared":true,
      "id":"d32019d3-bc6e-4319-9c1d-6722fc136a22",
      "provider:segmentation_id":null
   }
}
```

To show information for a specified subnet ID in a JSON response

- Issue the following cURL command:

```
                            $ curl https://dfw.servers.api.rackspacecloud.com/v2.0/$account/
subnets/{id} \
        -X GET \
        -H "X-Auth-Project-Id: $account" \
        -H "Accept: application/json" \
        -H "Content-Type: application/json"\
        -H "X-Auth-Token: $token" | python -m json.tool
```

Positional argument:

- The ID of the subnet for which you want to show information. To determine the subnet ID, issue a **neutron** `subnet-list` command.

Output:

The API operation returns a response body as shown in the following output:

Next step:the section called “Step 6. Delete Your Network” [62]

XML Request and Response

To show information for a specified subnet ID in an XML response

- Issue the following cURL command:

```
                            $ curl https://dfw.servers.api.rackspacecloud.com/v2.0/$account/
subnets/{id}.xml \
        -X GET \
        -H "X-Auth-Project-Id: $account" \
        -H "Accept: application/xml" \
        -H "Content-Type: application/xml"\
        -H "X-Auth-Token: $token"
```

Positional argument:

- The ID of the network for which you want to show information. To determine the network ID, issue a **neutron** `subnet-list` command.

Output:

The API operation returns the response body as shown in the following output:

Next step:the section called “Step 6. Delete Your Network” [62]

Step 4. Delete Your Network or Subnet

If you no longer need your network, or subnet, you can delete it.

To delete a network, specify the network ID. To delete a subnet, specify the subnet ID.

Next step: Choose one of the following methods:

- the section called “Delete Network or Subnet with the neutron Client” [39]
- the section called “Delete Network or Subnet with cURL” [41]

Delete Network or Subnet with the neutron Client

To delete a specified network

- **Delete the network**

Issue the following neutron client command:

```
$ neutron net-delete <id>
```

Positional argument:

- The network ID of the network that you want to delete. To determine the network ID, issue a **neutron** `net-list` command.

Output: None.

To delete a specified subnet

- **Delete the subnet**

  Issue the following neutron client command:

```
$ neutron subnet-delete <id>
```

Positional argument:

- The subnet ID of the subnet that you want to delete. To determine the subnet ID, issue a **neutron** `subnet-list` command.

Output: None.

Delete Network or Subnet with cURL

To delete a specified network

- **Delete the network**

  Issue the following cURL command:

```
                         $ curl -i https://dfw.servers.api.rackspacecloud.com/v2.0/$account/
networks/{id} \
       -X 'DELETE' \
       -H "X-Auth-Token: $token" \
       -H "X-Auth-Project-Id: $account" \
       -H "Accept: application/json"
```

Note

Include the `-i` option in the cURL command to show the response header. Omit the `| python -m json.tool` clause from the command because the API operation does not return a JSON response.

Argument:

- *`id`*. The network ID of the network that you want to delete. To determine the network ID, issue a **nova** `network-list` command.

Output:

Successful deletion returns an Accepted (202) response code:

```
HTTP/1.1 202 Accepted
Date: Thu, 04 Oct 2012 16:23:30 GMT
Content-Length: 58
Content-Type: text/html;charset=UTF-8
Server: Jetty(8.0.y.z-SNAPSHOT)

202 Accepted

The request is accepted for processing.
```

To delete a specified subnet

- **Delete the subnet**

  Issue the following cURL command:

```
                         $ curl -i https://dfw.servers.api.rackspacecloud.com/v2.0/$account/
subnets/{id} \
       -X 'DELETE' \
       -H "X-Auth-Token: $token" \
       -H "X-Auth-Project-Id: $account" \
       -H "Accept: application/json"
```

Note

Include the `-i` option in the cURL command to show the response header. Omit the `| python -m json.tool` clause from the command because the API operation does not return a JSON response.

Argument:

- *`id`*. The subnet ID of the subnet that you want to delete. To determine the network ID, issue a **nova** `network-list` command.

Output:

Successful deletion returns an Accepted (202) response code:

```
HTTP/1.1 202 Accepted
Date: Thu, 04 Oct 2012 16:23:30 GMT
Content-Length: 58
Content-Type: text/html;charset=UTF-8
Server: Jetty(8.0.y.z-SNAPSHOT)

202 Accepted

The request is accepted for processing.
```

Create Your First Cloud Network Using Nova-network

These examples, using the Nova-network API, walk you through the steps to create an isolated network and create a server that is attached your isolated network, showing you how to access the Cloud Networks API through nova client commands or cURL commands so that you can quickly create isolated networks and servers.

The exercises also help you learn how cURL commands and the Cloud Networks API work.

Step 1. Create an Isolated Network

Before you create a server, you create an isolated network that you can attach to your new server.

To create an isolated network, you specify a name for your network and a Classless Inter-Domain Routing (*CIDR*). A CIDR is a range of IP addresses used by a network. A CIDR address looks like a normal IP address except that it ends with a slash followed by a number. The number after the slash represents the number of addresses in the range. For more information, see *CIDR Notation*.

Limitations

- The isolated network must exist in the same region as the Cloud Server.
- You can create up to three isolated networks with up to 64 servers attached to each one.
- After you create an isolated network, you cannot rename it.

For a full list of limitations, see the *Cloud Networks Release Notes*.

After you create an isolated network, copy its network ID. You use this ID to attach the network to your server when you create your server.

Next step: Choose one of the following methods:

- the section called "Create an Isolated Network with the nova Client " [43]
- the section called "Create an Isolated Network with cURL" [44]

Create an Isolated Network with the nova Client

To create an isolated network with the nova client

1. Issue the following command:

   ```
   $ nova network-create <network_label> <cidr>
   ```

   Positional arguments:

   - *network_label*. The network label, such as `my_new_network`,
   - *cidr*. The IP block from which to allocate, such as `172.16.0.0/24` or `2001:DB8::/64`.

   Output:

   The command returns the CIDR, ID, and label for the new network, as shown in the following output:

   ```
   +----------+--------------------------------------+
   | Property | Value                                |
   +----------+--------------------------------------+
   | cidr     | 192.168.0.0/24                       |
   | id       | 1f84c238-b05a-4374-a0cb-aa6140032cd1 |
   | label    | new_network                          |
   +----------+--------------------------------------+
   ```

2. Copy the `id` value from the output. In this example, the `id` value is `1f84c238-b05a-4374-a0cb-aa6140032cd1`, but use the value returned for your network.

   You will use this value when you provision your server in the next step.

Next step:the section called "Step 2. List Networks" [45]

Create an Isolated Network with cURL

You can request a JSON or XML response when you issue a request in a cURL command.

JSON Request and Response

To create an isolated network and request a JSON response

1. Issue the following cURL command:

```
                            $ curl -s https://dfw.servers.api.rackspacecloud.com/v2/
$account/os-networksv2 \
       -X POST \
       -H "X-Auth-Project-Id: $account" \
       -H "Content-Type: application/json" \
       -H "Accept: application/json" \
       -H "X-Auth-Token: $token" \
       -d '{"network": {"cidr": "192.168.0.0/24", "label": "superprivate"}}' | python -m
 json.tool
```

   In this cURL command, you specify several options. For information about options, see Table 2, "cURL Command-Line Options" [10].

   Output:

   The response returns the *CIDR*, ID, and label for the isolated network, as shown in the following JSON response:

```
                          {
    "network": {
        "cidr": "192.168.0.0/24",
        "id": "1ff4489e-db0e-45a6-8c9f-4616c6ef5db1",
        "label": "superprivate"
    }
}
```

2. Copy the `id` value from the output. In this example, the `id` value is `1f84c238-b05a-4374-a0cb-aa6140032cd1`, but use the value returned for your network.

   You will use this value when you provision your server in the next step.

Next step:the section called "Step 2. List Networks" [45]

XML Request and Response

To create an isolated network and request a XML response

1. Issue the following cURL command:

```
                            $ curl -i https://dfw.servers.api.rackspacecloud.com/v2/
$account/os-networksv2.xml \
          -X POST -H "X-Auth-Project-Id: $account" -H "Content-Type: application/xml" \
          -H "Accept: application/xml" \
          -H "X-Auth-Token: $token" \
          -d '<network cidr="192.168.0.0/24" label="superprivate_xml" />'
```

   Output:

The response returns the *CIDR*, ID, and label for the isolated network, as shown in the following XML response:

```
                        <network>
    <cidr>192.168.0.0/24</cidr>
    <id>f212726e-6321-4210-9bae-a13f5a33f83f</id>
    <label>superprivate_xml</label>
</network>
```

2. Copy the `id` value from the output. In this example, the `id` value is `1f84c238-b05a-4374-a0cb-aa6140032cd1`, but use the value returned for your network.

   You will use this value when you provision your server in the next step.

Next step:the section called “Step 2. List Networks” [45]

Step 2. List Networks

After you create an isolated network, list networks to view your network and the Rackspace networks.

For information about the Rackspace networks, see the section called “Limitations of Detaching from Rackspace Networks” [6].

Next step: Choose one of the following methods:

- the section called “List Networks with the nova Client” [45]
- the section called “List Networks with cURL” [47]

List Networks with the nova Client

To list networks with the nova client

- Issue the following nova client command:

```
                        $ nova network-list
```

Output:

For each network, the command returns the network ID, label, and *CIDR*, as shown in the following output:

```
+--------------------------------------+-------------+----------------+
| ID                                   | Label       | CIDR           |
+--------------------------------------+-------------+----------------+
| 00000000-0000-0000-0000-000000000000 | public      |                |
| 11111111-1111-1111-1111-111111111111 | private     |                |
| 1f84c238-b05a-4374-a0cb-aa6140032cd1 | new_network | 192.168.0.0/24 |
+--------------------------------------+-------------+----------------+
```

In the network list, ServiceNet is labeled as `private`, and PublicNet is labeled as `public`.

Next step:the section called “Step 3. Provision a Server with an Isolated Network” [48]

List Networks with cURL

You can request a JSON or XML response when you issue a request in a cURL command.

JSON Response

To list networks and request a JSON response

- Issue the following cURL command:

```
                        $ curl https://dfw.servers.api.rackspacecloud.com/v2/
$account/os-networksv2 \
       -X GET -H "X-Auth-Token: $token" | python -m json.tool
```

Output:

For each isolated network, the response returns the *CIDR*. Additionally, for isolated networks and Rackspace networks, the response returns the network ID and label.

```
                            {
   "networks":[
      {
         "cidr":"192.168.0.0/24",
         "id":"1f84c238-b05a-4374-a0cb-aa6140032cd1",
         "label":"new_network"
      },
      {
         "id":"00000000-0000-0000-0000-000000000000",
         "label":"public"
      },
      {
         "id":"11111111-1111-1111-1111-111111111111",
         "label":"private"
      }
   ]
}
```

Next step:the section called “Step 3. Provision a Server with an Isolated Network” [48]

XML Response

To list networks and request an XML response

- Issue the following cURL command:

```
                        $curl -i https://dfw.servers.api.rackspacecloud.com/v2/
$account/os-networksv2.xml \
        -H "Content-Type: application/xml" \
        -H "Accept: application/xml" \
        -X GET -H "X-Auth-Token: $token"
```

Output:

For each isolated network, the response returns the *CIDR*. Additionally, for isolated networks and Rackspace networks, the response returns the network ID and label.

```
                            <?xml version='1.0' encoding='UTF-8'?>
<networks>
    <network>
        <cidr>172.16.0.0/24</cidr>
        <id>e65accc0-1d98-45eb-af76-ab3d31edc7d2</id>
        <label>new_network</label>
    </network>
    <network>
        <id>00000000-0000-0000-0000-000000000000</id>
        <label>public</label>
    </network>
    <network>
        <id>11111111-1111-1111-1111-111111111111</id>
        <label>private</label>
    </network>
</networks>
```

ServiceNet is labeled as `private` and PublicNet is labeled as `public` in the network list.

Next step:the section called "Step 3. Provision a Server with an Isolated Network" [48]

Step 3. Provision a Server with an Isolated Network

To create your new server and attach an isolated network, you need the following information:

- The name of the new server. Use a name of your choice.
- The image ID. For how to list images, see List Images.
- The flavor ID. For how to list flavors, see List Flavors.
- The network ID of your isolated network. This is the ID you copied in the section called "Step 1. Create an Isolated Network" [42]

You specify these values as parameters in a nova command or as attributes in the request body in a cURL command.

Note

You cannot attach an isolated network to an existing server. However, to work around this limitation, you can create an image of the server and build a new server based on that image. When you build the new server, you can attach the isolated network. For information, see *Attach a Cloud Network to an Existing Cloud Server*. To use the create image API, see Create Image.

Note

You can use the Cloud Networks virtual interface extension to attach an isolated network to an existing server. For information, see Attach Your Network to an Existing Server [65].

Next step: Choose one of the following methods:

- the section called “Provision a Server with an Isolated Network with the nova Client” [50]Provision a server with an isolated network with the nova client
- the section called “Provision a Server with an Isolated Network with cURL” [53]Provision a server with an isolated network with cURL

Provision a Server with an Isolated Network with the nova Client

To provision a server and attach an isolated network

- Issue the following nova client command:

```
                        $ nova boot <name> --flavor <flavor> --image <image> --nic [net-id=<private-net-uuid> [--no-
public] [--no-service-net]
```

Positional arguments:

- *`name`*. The name of the server.

Optional arguments:

- `--no-service-net`. Optional. Opts out of attaching ServiceNet to your server.

  Note

  RackConnect and Managed Cloud Service Level customers will receive an error if they opt out of attaching ServiceNet to their server.

  For example, to opt out of attaching ServiceNet to your server, issue the following command:

```
                                      $ nova boot <name> --flavor <flavor> --image <image> --nic net-id=
<private-net-uuid> --no-service-net
```

- `--no-public`. Optional. Opts out of attaching PublicNet to your server.

  Note

  RackConnect and Managed Cloud Service Level customers will receive an error if they opt out of attaching PublicNet to their server.

  For example, to opt out of attaching PublicNet to your server, issue the following command:

```
                                      $ nova boot <name> --flavor <flavor> --image <image> --nic net-id=
<private-net-uuid> --no-public
```

- `--disk-config {auto|manual}`. Specifies whether to expand the primary partition to fill the disk. This value overrides the value inherited from the image.
- `--flavor` *`flavor`*. Required. The flavor ID for the new server. To get a list of flavors, issue **nova flavor-list**.
- `--image` *`image`*. Required. The image ID for the new server. To get a list of images, issue **nova image-list**.
- `--meta` *`key=value`*. Arbitrary key and value metadata that is written to `/meta.js` on the new server. Can be specified multiple times.
- `--file` *`dst-path=src-path`*. Stores arbitrary files from *`src-path`* locally to *`dst- path`* on the new server. You can store up to five files.

- `--key_name` *`key_name`*. Key name of key pair. First, create the key pair with the **keypair-add** command.
- `--user_data` *`user-data`*. User data file, which is exposed by the metadata server.
- `--nic net-id=`*`private-net-id`*>. Creates a NIC on the server. Specify this option multiple times to create multiple NICs. Optionally, to attach a NIC with a specified UUID to a network, specify the `net-id=`*`private-net-id`* parameter.

  If you do not specify any networks on the `--nic` parameter, PublicNet and ServiceNet are attached to your server.

  Note

  ServiceNet is labeled as `private` and PublicNet is labeled as `public` in the network list.

  If you specify additional networks through the —`nic` parameter, these networks, in addition to PublicNet and ServiceNet, are attached to your server.

  You can specify the optional `--no-public` and `--no-service-net` parameters to opt out of attaching PublicNet and ServiceNet to your server.

  Specify the `--no-public` parameter to opt out of attaching PublicNet to your server.

  Specify the `--no-service-net` parameter to opt out of attaching ServiceNet to your server.
- `--config-drive` *`value`*. Enables a configuration drive.
- `--poll`. Blocks while the instance builds so progress can be reported.

For example, you might issue the following command to attach a NIC to your server. Additionally, PublicNet and ServiceNet are attached to your server.

```
                            $ nova boot my_server_with_network --
flavor 2 --image d42f821e-c2d1-4796-9f07-af5ed7912d0e --nic net-id=
e65accc0-1d98-45eb-af76-ab3d31edc7d2
```

Output: The command returns information about the new server, as shown in the following output:

```
+-------------------------+---------------------------------------+
| Property                | Value                                 |
+-------------------------+---------------------------------------+
| OS-DCF:diskConfig       | AUTO                                  |
| OS-EXT-STS:power_state  | 0                                     |
| OS-EXT-STS:task_state   | scheduling                            |
| OS-EXT-STS:vm_state     | building                              |
| accessIPv4              |                                       |
| accessIPv6              |                                       |
| adminPass                | abababababab                         |
| created                  | 2012-10-04T18:07:56Z                 |
| flavor                   | 512MB Standard Instance              |
| hostId                   |                                      |
| id                       | 72859427-1e74-4a4f-b6b5-f547c1c3d452 |
| image                    | Fedora 17 (Beefy Miracle)            |
| metadata                 | {}                                   |
| name                     | my_server_with_network               |
| progress                 | 0                                    |
| status                   | BUILD                                |
| tenant_id                | 010101                               |
| updated                  | 2012-10-04T18:07:56Z                 |
| user_id                  | 170454                               |
+--------------------------+--------------------------------------+
```

Next step: the section called “Step 4. List Servers and Attached Networks” [55]

Provision a Server with an Isolated Network with cURL

JSON Request and Response

To provision a server with an isolated network to the server

- Issue the following cURL command:

```
                            $ curl https://dfw.servers.api.rackspacecloud.com/v2/$account/servers \
        -X POST \
        -H "X-Auth-Project-Id: $account" \
        -H "Content-Type: application/json" \
        -H "Accept: application/json" \
        -H "X-Auth-Token: $token" \
        -d '{"server": {"name": "my_server_with_network", "imageRef": "d42f821e-c2d1-4796-9f07-af5ed7912d0e",
 "flavorRef": "2", "max_count": 1, "min_count": 1, "networks": [{"uuid": "538a112a-34d1-47ff-bf1e-c40639e886e2"},
 {"uuid": "00000000-0000-0000-0000-000000000000"}, {"uuid": "11111111-1111-1111-1111-111111111111"}]}}' \
        | python -m json.tool
```

In addition to the isolated network, PublicNet and ServiceNet are attached to the server. You must explicitly specify these networks to attach them to your server.

Output: The API operation returns information about the new server, as shown in the following output:

```
                        {
    "server": {
        "OS-DCF:diskConfig": "AUTO",
        "adminPass": "LMoheHauXt8w",
        "id": "ef08aa7a-b5e4-4bb8-86df-5ac56230f841",
        "links": [
            {
                "href": "https://dfw.servers.api.rackspacecloud.com/v2/010101/servers/ef08aa7a-
b5e4-4bb8-86df-5ac56230f841",
                "rel": "self"
            },
            {
                "href": "https://dfw.servers.api.rackspacecloud.com/010101/servers/ef08aa7a-
b5e4-4bb8-86df-5ac56230f841",
                "rel": "bookmark"
            }
        ]
    }
}
```

Next step: the section called “Step 4. List Servers and Attached Networks” [55]

XML Request and Response

To provision a server and attach an isolated network to the server

- Issue the following cURL command:

```
                            $ curl -i https://dfw.servers.api.rackspacecloud.com/v2/$account/servers.xml \
       -X POST \
       -H "X-Auth-Project-Id: $account" \
       -H "Content-Type: application/xml" \
       -H "Accept: application/xml" \
       -H "X-Auth-Token: $token" \
       -d '<?xml version="1.0" encoding="UTF-8"?><server xmlns="http://docs.openstack.org/compute/api/v1.1"
 imageRef="3afe97b2-26dc-49c5-a2cc-a2fc8d80c001" flavorRef="2" name="api-test-server-xml2"><metadata><meta key=
"My Server Name">API Test Server XML</meta></metadata><networks><uuid>0ef47ac7-6797-4e01-8a47-ed26ec3aaa56</
uuid><uuid>00000000-0000-0000-0000-000000000000</uuid><uuid>11111111-1111-1111-1111-111111111111</uuid></
networks></server>'
```

In addition to the isolated network, PublicNet and ServiceNet are attached to the server. You must explicitly specify these networks to attach them to your server.

Output: The API operation returns information about the new server, as shown in the following output:

```
<?xml version='1.0' encoding='UTF-8'?>
<server
    xmlns:OS-DCF="http://docs.openstack.org/compute/ext/disk_config/api/v1.1"
    xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns="http://docs.openstack.org/compute/api/v1.1"
    id="ed5c7754-29b6-45fa-96cb-ab64958c8c0a" adminPass="Dd5pNZtpVVQ3"
    OS-DCF:diskConfig="AUTO">
    <metadata/>
    <atom:link
        href="https://dfw.servers.api.rackspacecloud.com/v2/010101/servers/
ed5c7754-29b6-45fa-96cb-ab64958c8c0a"
        rel="self"/>
    <atom:link
        href="https://dfw.servers.api.rackspacecloud.com/010101/servers/ed5c7754-29b6-45fa-96cb-
ab64958c8c0a"
        rel="bookmark"/>
</server>
```

Next step: the section called “Step 4. List Servers and Attached Networks” [55]

Step 4. List Servers and Attached Networks

To show all the servers in your Cloud account, you can list servers. When you list servers, the addresses for any attached networks are displayed. The networks include any isolated networks that you have created and Rackspace public and private networks.

Next step: Choose one of the following methods:

- the section called "List Servers with the nova Client" [56]
- the section called "List Servers with cURL" [57]

List Servers with the nova Client

To list servers with the nova client:

1. Issue the following command:

   ```
   $ nova list
   ```

   For each server, the command returns the server ID, name, status, and addresses for any attached networks, as shown in the following output:

   ```
   +--------------------------------------+----------------+--------+-----------------------------------------------------------------------------------------------+
   | ID                                   | Name           | Status | Networks                                                                                      |
   +--------------------------------------+----------------+--------+-----------------------------------------------------------------------------------------------+
   | 9da98125-0de8-4b84-880c-b42977c32773 | myUbuntuServer | ACTIVE | public=2001:4800:780d:0509:d87b:9cbc:ff04:488b, 198.101.231.59; private=10.179.224.74 |
   | a09e7493-7429-41e1-8d3f-384d7ece09c0 | UbuntuDevStack | ACTIVE | public=2001:4800:780e:0510:d87b:9cbc:ff04:3e81, 50.56.186.185; private=10.180.13.75   |
   +--------------------------------------+----------------+--------+-----------------------------------------------------------------------------------------------+
   ```

   The networks include any isolated networks that you have created and Rackspace public and private networks.

2. Look for your server in the list of servers. Servers are listed by server ID.

   Copy the server ID for your server. To update or delete your server, you need this ID.

   You use the public IP address when you log into your server.

Next step:the section called “Step 5. Show Network Information” [59]

List Servers with cURL

To list servers with cURL:

1. Issue the following command:

```
                $ curl -s https://dfw.servers.api.rackspacecloud.com/v2/$account/servers/detail \
       -H "X-Auth-Token: $token" | python -m json.tool
```

For each server, the command returns the disk configuration, the addresses of any attached networks, flavor and image information, the server ID, and the server status. The networks include any isolated networks that you have created and Rackspace public and private networks.

The following output shows the list servers information, including information for the Ubuntu server that you just created:

```
{
    "servers": [
        {
            "OS-DCF:diskConfig": "AUTO",
            "OS-EXT-STS:power_state": 0,
            "OS-EXT-STS:task_state": "spawning",
            "OS-EXT-STS:vm_state": "building",
            "accessIPv4": "",
            "accessIPv6": "",
            "addresses": {
                "private": [
                    {
                        "addr": "10.180.16.21",
                        "version": 4
                    }
                ],
                "public": [
                    {
                        "addr": "2001:4800:780e:0510:d87b:9cbc:ff04:4bac",
                        "version": 6
                    },
                    {
                        "addr": "198.101.242.47",
                        "version": 4
                    }
                ]
            },
            "created": "2012-08-16T17:11:43Z",
            "flavor": {
                "id": "6",
                "links": [
                    {
                        "href": "https://dfw.servers.api.rackspacecloud.com/010101/flavors/6",
                        "rel": "bookmark"
                    }
                ]
            },
            "hostId": "692f611d2d84e5368b5995d7733ca5f175dc670eda91eded50a47eab",
            "id": "1a861bf7-2a5e-40a4-acb3-1fb058cf2a74",
            "image": {
                "id": "3afe97b2-26dc-49c5-a2cc-a2fc8d80c001",
                "links": [
                    {
                        "href": "https://dfw.servers.api.rackspacecloud.com/010101/images/
3afe97b2-26dc-49c5-a2cc-a2fc8d80c001",
                        "rel": "bookmark"
                    }
                ]
            },
            "links": [
                {
                    "href": "https://dfw.servers.api.rackspacecloud.com/v2/010101/servers/
1a861bf7-2a5e-40a4-acb3-1fb058cf2a74",
                    "rel": "self"
                },
                {
                    "href": "https://dfw.servers.api.rackspacecloud.com/010101/servers/
1a861bf7-2a5e-40a4-acb3-1fb058cf2a74",
                    "rel": "bookmark"
                }
            ],
            "metadata": {
                "My Server Name": "Ubuntu 11.10 server"
            },
            "name": "myUbuntuServer",
            "progress": 75,
            "status": "BUILD",
            "tenant_id": "010101",
            "updated": "2012-08-16T17:14:50Z",
            "user_id": "170454"
        },
        {
            "OS-DCF:diskConfig": "AUTO",
            "OS-EXT-STS:power_state": 1,
            "OS-EXT-STS:task_state": null,
            "OS-EXT-STS:vm_state": "active",
            "accessIPv4": "",
            "accessIPv6": "",
            "addresses": {
                "private": [
                    {
                        "addr": "10.180.13.75",
                        "version": 4
                    }
                ],
                "public": [
                    {
                        "addr": "2001:4800:780e:0510:d87b:9cbc:ff04:3e81",
                        "version": 6
                    },
                    {
                        "addr": "50.56.186.185",
                        "version": 4
                    }
                ]
            },
            "created": "2012-05-15T15:47:37Z",
            "flavor": {
                "id": "6",
                "links": [
                    {
                        "href": "https://dfw.servers.api.rackspacecloud.com/010101/flavors/6",
                        "rel": "bookmark"
                    }
                ]
            },
            "hostId": "1d65b563fc573c2eb544319e0af598f2b2c5f84f75de252db3757cd3",
            "id": "a09e7493-7429-41e1-8d3f-384d7ece09c0",
            "image": {
                "id": "3afe97b2-26dc-49c5-a2cc-a2fc8d80c001",
                "links": [
                    {
                        "href": "https://dfw.servers.api.rackspacecloud.com/010101/images/
3afe97b2-26dc-49c5-a2cc-a2fc8d80c001",
                        "rel": "bookmark"
                    }
                ]
            },
            "links": [
                {
                    "href": "https://dfw.servers.api.rackspacecloud.com/v2/010101/servers/
a09e7493-7429-41e1-8d3f-384d7ece09c0",
                    "rel": "self"
                },
                {
                    "href": "https://dfw.servers.api.rackspacecloud.com/010101/servers/
a09e7493-7429-41e1-8d3f-384d7ece09c0",
                    "rel": "bookmark"
                }
            ],
            "metadata": {},
            "name": "UbuntuDevStack",
            "progress": 100,
            "status": "ACTIVE",
            "tenant_id": "010101",
            "updated": "2012-05-15T15:55:00Z",
            "user_id": "170454"
        }
    ]
}
```

2. Look for your server in the list of servers. Servers are listed by server ID.

Copy the server ID for your server. To update or delete your server, you need this ID.

You use the public IP address when you log into your server.

Next step:the section called “Step 5. Show Network Information” [59]

Step 5. Show Network Information

You can show details for a specified network ID.

The response body returns the following fields:

Table 5. Show Network Response Fields

Name	Description
cidr	The *CIDR* for an isolated private network.
id	The network ID.
label	The name of the network.

Next step: Choose one of the following methods:

- the section called “Show Network Information with the nova Client” [60]
- the section called “Show Network with cURL” [61]

Show Network Information with the nova Client

To show information for a specified network ID

- Issue the following nova client command:

```
$ nova network <id>
```

Positional argument:

- *`id`*. The ID of the network for which you want to show information. To determine the network ID, issue a **nova** `network-list` command.

Output:

The command returns the *CIDR*, network ID, and label, as shown in the following output:

```
+----------+--------------------------------------+
| Property | Value                                |
+----------+--------------------------------------+
| cidr     | 172.16.0.0/24                        |
| id       | e65accc0-1d98-45eb-af76-ab3d31edc7d2 |
| label    | new_network                          |
+----------+--------------------------------------+
```

Next step:the section called “Step 6. Delete Your Network” [62]

Show Network with cURL

You can request a JSON or XML response when you issue a request in a cURL command.

JSON Response

To show information in a JSON response for a specified network ID

- Issue the following cURL command:

```
                            $ curl https://dfw.servers.api.rackspacecloud.com/v2/$account/os-
networksv2/<id> \
        -X GET \
        -H "X-Auth-Project-Id: $account" \
        -H "Accept: application/json" \
        -H "Content-Type: application/json"\
        -H "X-Auth-Token: $token" | python -m json.tool
```

Positional argument:

- *id*. The ID of the network for which you want to show information. To determine the network ID, issue a **nova** `network-list` command.

Output:

The API operation returns the *CIDR*, network ID, and label, as shown in the following output:

```
                        {
    "network": {
        "cidr": "192.168.0.0/24",
        "id": "f212726e-6321-4210-9bae-a13f5a33f83f",
        "label": "superprivate_xml"
    }
}
```

Next step:the section called “Step 6. Delete Your Network” [62]

XML Response

To show information in an XML response for a specified network ID

- Issue the following cURL command:

```
                            $ curl https://dfw.servers.api.rackspacecloud.com/v2/$account/os-
networksv2/<id>.xml \
        -X GET \
        -H "X-Auth-Project-Id: $account" \
        -H "Accept: application/xml" \
        -H "Content-Type: application/xml"\
        -H "X-Auth-Token: $token"
```

Positional argument:

- *id*. The ID of the network for which you want to show information. To determine the network ID, issue a **nova** `network-list` command.

Output:

The API operation returns the *CIDR*, network ID, and label, as shown in the following output:

```
                        <network>
    <cidr>192.168.0.0/24</cidr>
    <id>f212726e-6321-4210-9bae-a13f5a33f83f</id>
    <label>superprivate_xml</label>
</network>
```

Next step:the section called “Step 6. Delete Your Network” [62]

Step 6. Delete Your Network

If you no longer need your network, you can delete it.

You cannot delete an isolated network unless the network is not attached to any server.

To detach a network from a server, you must delete the virtual interface for the network from the server. Then, you can delete the network.

After the network is deleted, it no longer appears in the list of networks.

Next step: Choose one of the following methods:

- the section called “Delete Network with the nova Client” [62]
- the section called “Delete Network with cURL” [64]

Delete Network with the nova Client

To delete a specified network

1. **Delete the virtual interface for the network from the server**

   To detach a network from a server, first delete the virtual interface for the network from the server.

2. **Delete the network**

   Issue the following nova client command:

```
                        $ nova network-delete <id>
```

Positional argument:

- *`id`*. The network ID of the network that you want to delete. To determine the network ID, issue a **nova** `network-list` command.

Output: None.

Delete Network with cURL

To delete a specified network

1. **Delete the virtual interface for the network from the server**

   To detach a network from a server, first delete the virtual interface for the network from the server.

2. **Delete the network**

   Issue the following cURL command:

```
                        $ curl -i https://dfw.servers.api.rackspacecloud.com/v2/$account/os-
networksv2/<id> \
       -X 'DELETE' \
       -H "X-Auth-Token: $token" \
       -H "X-Auth-Project-Id: $account" \
       -H "Accept: application/json"
```

Note

Include the `-i` option in the cURL command to show the response header. Omit the `| python -m json.tool` clause from the command because the API operation does not return a JSON response.

Argument:

- *`id`*. The network ID of the network that you want to delete. To determine the network ID, issue a **nova** `network-list` command.

Output:

Successful deletion returns an Accepted (202) response code:

```
HTTP/1.1 202 Accepted
Date: Thu, 04 Oct 2012 16:23:30 GMT
Content-Length: 58
Content-Type: text/html;charset=UTF-8
Server: Jetty(8.0.y.z-SNAPSHOT)

202 Accepted

The request is accepted for processing.
```

Attach Your Network to an Existing Server

Before attaching your first network to a server, choose whether to use the Neutron API or the Nova-network API. For help deciding, see the section called “Networking: Nova-network versus Neutron” [5]. The sections in this chapter provide examples of attaching a network to an existing server using each API.

Attach Your Network to an Existing Server Using Neutron

How do we do this with Neutron?

Attach Your Network to an Existing Server Using Nova-network

Use the Cloud Networks virtual interface extension to create a virtual interface to a specified network and attach the network to an existing server instance with the Nova-network API.

You can create a maximum of one virtual interface per instance per network.

You can also use the Cloud Networks virtual interface extension to:

- List the virtual interfaces for a server instance.
- Delete a virtual interface and detach it from a server instance.

These examples walk you through the steps to create a virtual interface to a specified network and attach the network to an existing server instance. The simple exercises show you how to access the Cloud Networks virtual interface extension through nova client commands or cURL commands.

The exercises also help you learn how cURL commands and the Cloud Networks API work.

Step 1. Install the Cloud Networks Virtual Interface Extension

To install the Cloud Networks virtual interface extension

1. **Install the Cloud Networks virtual interface extension**

   The `os_virtual_interfacesv2_python_novaclient_ext` package contains the Cloud Networks virtual interface extension.

Run the following command to install the package:

```
                     $ sudo pip install
os_virtual_interfacesv2_python_novaclient_ext
```

Note

If you previously installed this package, run the following command to upgrade it:

```
                          $ sudo pip install
os_virtual_interfacesv2_python_novaclient_ext --upgrade
```

2. **Test the Cloud Networks virtual interface extension**

   To verify that you can talk to the API server, run the following commands.

   First, authenticate:

```
                         $ nova credentials
```

   Then, list networks:

```
                         $ nova network-list
```

   Then, list virtual interfaces:

```
                         $ nova virtual-interface-list instance_id
```

   Where *instance_id* is the ID of your server instance.

Step 2. List Servers

To get the ID of the server instance to which you want to attach a virtual interface, list the servers in your Cloud account.

Next step: Choose one of the following methods:

- the section called "List Servers with the nova Client" [67]
- the section called "List Servers with cURL" [68]

List Servers with the nova Client

To list servers with the nova client:

1. Issue the following command:

```
                        $ nova list
```

   For each server, the command returns the server ID, name, status, and public and private IP addresses, as shown in the following output:

```
+--------------------------------------+----------------+--------
+---------------------------------------------------------------------------------------
+
| ID                                   | Name           | Status |
 Networks
             |
+--------------------------------------+----------------+--------
+---------------------------------------------------------------------------------------
+
| 9da98125-0de8-4b84-880c-b42977c32773 | myUbuntuServer | ACTIVE | public=
2001:4800:780d:0509:d87b:9cbc:ff04:488b, 198.101.231.59; private=10.179.
224.74 |
| a09e7493-7429-41e1-8d3f-384d7ece09c0 | UbuntuDevStack | ACTIVE | public=
2001:4800:780e:0510:d87b:9cbc:ff04:3e81, 50.56.186.185; private=10.180.13.
75    |
+--------------------------------------+----------------+--------
+---------------------------------------------------------------------------------------
+
```

2. Copy the ID for the server to which you want to connect a virtual interface.

Next step: the section called “Step 3. List Networks” [70]

List Servers with cURL

To list servers with cURL:

1. Issue the following command:

```
                        $ curl -s https://dfw.servers.api.rackspacecloud.com/v2/$account/servers/detail \
     -H "X-Auth-Token: $token" | python -m json.tool
```

For each server, the command returns the disk configuration, private and public IP addresses, flavor and image information, the server ID, and the server status.

The following output shows the list servers information:

```
{
    "servers": [
        {
            "OS-DCF:diskConfig": "AUTO",
            "OS-EXT-STS:power_state": 0,
            "OS-EXT-STS:task_state": "spawning",
            "OS-EXT-STS:vm_state": "building",
            "accessIPv4": "",
            "accessIPv6": "",
            "addresses": {
                "private": [
                    {
                        "addr": "10.180.16.21",
                        "version": 4
                    }
                ],
                "public": [
                    {
                        "addr": "2001:4800:780e:0510:d87b:9cbc:ff04:4bac",
                        "version": 6
                    },
                    {
                        "addr": "198.101.242.47",
                        "version": 4
                    }
                ]
            },
            "created": "2012-08-16T17:11:43Z",
            "flavor": {
                "id": "6",
                "links": [
                    {
                        "href": "https://dfw.servers.api.rackspacecloud.com/010101/flavors/6",
                        "rel": "bookmark"
                    }
                ]
            },
            "hostId": "692f611d2d84e5368b5995d7733ca5f175dc670eda91eded50a47eab",
            "id": "1a861bf7-2a5e-40a4-acb3-1fb058cf2a74",
            "image": {
                "id": "3afe97b2-26dc-49c5-a2cc-a2fc8d80c001",
                "links": [
                    {
                        "href": "https://dfw.servers.api.rackspacecloud.com/010101/images/
3afe97b2-26dc-49c5-a2cc-a2fc8d80c001",
                        "rel": "bookmark"
                    }
                ]
            },
            "links": [
                {
                    "href": "https://dfw.servers.api.rackspacecloud.com/v2/010101/servers/
1a861bf7-2a5e-40a4-acb3-1fb058cf2a74",
                    "rel": "self"
                },
                {
                    "href": "https://dfw.servers.api.rackspacecloud.com/010101/servers/
1a861bf7-2a5e-40a4-acb3-1fb058cf2a74",
                    "rel": "bookmark"
                }
            ],
            "metadata": {
                "My Server Name": "Ubuntu 11.10 server"
            },
            "name": "myUbuntuServer",
            "progress": 75,
            "status": "BUILD",
            "tenant_id": "010101",
            "updated": "2012-08-16T17:14:50Z",
            "user_id": "170454"
        },
        {
            "OS-DCF:diskConfig": "AUTO",
            "OS-EXT-STS:power_state": 1,
            "OS-EXT-STS:task_state": null,
            "OS-EXT-STS:vm_state": "active",
            "accessIPv4": "",
            "accessIPv6": "",
            "addresses": {
                "private": [
                    {
                        "addr": "10.180.13.75",
                        "version": 4
                    }
                ],
                "public": [
                    {
                        "addr": "2001:4800:780e:0510:d87b:9cbc:ff04:3e81",
                        "version": 6
                    },
                    {
                        "addr": "50.56.186.185",
                        "version": 4
                    }
                ]
            },
            "created": "2012-05-15T15:47:37Z",
            "flavor": {
                "id": "6",
                "links": [
                    {
                        "href": "https://dfw.servers.api.rackspacecloud.com/010101/flavors/6",
                        "rel": "bookmark"
                    }
                ]
            },
            "hostId": "1d65b563fc573c2eb544319e0af598f2b2c5f84f75de252db3757cd3",
            "id": "a09e7493-7429-41e1-8d3f-384d7ece09c0",
            "image": {
                "id": "3afe97b2-26dc-49c5-a2cc-a2fc8d80c001",
                "links": [
                    {
                        "href": "https://dfw.servers.api.rackspacecloud.com/010101/images/
3afe97b2-26dc-49c5-a2cc-a2fc8d80c001",
                        "rel": "bookmark"
                    }
                ]
            },
            "links": [
                {
                    "href": "https://dfw.servers.api.rackspacecloud.com/v2/010101/servers/
a09e7493-7429-41e1-8d3f-384d7ece09c0",
                    "rel": "self"
                },
                {
                    "href": "https://dfw.servers.api.rackspacecloud.com/010101/servers/
a09e7493-7429-41e1-8d3f-384d7ece09c0",
                    "rel": "bookmark"
                }
            ],
            "metadata": {},
            "name": "UbuntuDevStack",
            "progress": 100,
            "status": "ACTIVE",
            "tenant_id": "010101",
            "updated": "2012-05-15T15:55:00Z",
            "user_id": "170454"
        }
    ]
}
```

2. Copy the ID for the server to which you want to connect a virtual interface.

Next step: the section called “Step 3. List Networks” [70]

Step 3. List Networks

To get the ID of the network for which you want to create virtual interface, list the networks in your Cloud account.

Next step: Choose one of the following methods:

- the section called "List Networks with the nova Client" [70]
- the section called "List Networks with cURL" [71]

List Networks with the nova Client

To list networks with the nova client

1. Issue the following nova client command:

   ```
   $ nova network-list
   ```

   Output:

   For each network, the command returns the network ID, label, and *CIDR*, as shown in the following output:

   ```
   +--------------------------------------+-------------+----------------+
   | ID                                   | Label       | CIDR           |
   +--------------------------------------+-------------+----------------+
   | 00000000-0000-0000-0000-000000000000 | public      |                |
   | 11111111-1111-1111-1111-111111111111 | private     |                |
   | 1f84c238-b05a-4374-a0cb-aa6140032cd1 | new_network | 192.168.0.0/24 |
   +--------------------------------------+-------------+----------------+
   ```

   In the network list, ServiceNet is labeled as `private`, and PublicNet is labeled as `public`.

2. Copy the ID for the network for which you want to create a virtual interface.

Next step: the section called "Step 4. Create a Virtual Interface" [72]

List Networks with cURL

You can request a JSON or XML response when you issue a request in a cURL command.

JSON Request and Response

To list networks and request a JSON response

1. Issue the following cURL command:

```
                            $ curl https://dfw.servers.api.rackspacecloud.com/v2/
$account/os-networksv2 \
       -X GET \
       -H "X-Auth-Token: $token" | python -m json.tool
```

Output:

For each isolated network, the response returns the *CIDR*. Additionally, for isolated networks and Rackspace networks, the response returns the network ID and label.

```
                                {
   "networks":[
      {
         "cidr":"192.168.0.0/24",
         "id":"1f84c238-b05a-4374-a0cb-aa6140032cd1",
         "label":"new_network"
      },
      {
         "id":"00000000-0000-0000-0000-000000000000",
         "label":"public"
      },
      {
         "id":"11111111-1111-1111-1111-111111111111",
         "label":"private"
      }
   ]
}
```

2. Copy the ID for the network for which you want to create a virtual interface.

Next step: the section called “Step 4. Create a Virtual Interface” [72]

XML Request and Response

To list networks and request an XML response

1. Issue the following cURL command:

```
curl -i https://dfw.servers.api.rackspacecloud.com/v2/$account/os-networksv2.xml \
        -H "Content-Type: application/xml" \
        -H "Accept: application/xml" \
        -X GET -H "X-Auth-Token: $token"
```

Output:

For each isolated network, the response returns the *CIDR*. Additionally, for isolated networks and Rackspace networks, the response returns the network ID and label.

```
                                <?xml version='1.0' encoding='UTF-8'?>
<networks>
    <network>
        <cidr>172.16.0.0/24</cidr>
        <id>e65accc0-1d98-45eb-af76-ab3d31edc7d2</id>
        <label>new_network</label>
    </network>
    <network>
        <id>00000000-0000-0000-0000-000000000000</id>
        <label>public</label>
    </network>
    <network>
        <id>11111111-1111-1111-1111-111111111111</id>
        <label>private</label>
    </network>
</networks>
```

ServiceNet is labeled as `private` and PublicNet is labeled as `public` in the network list.

2. Copy the ID for the network for which you want to create a virtual interface.

Next step: the section called "Step 4. Create a Virtual Interface" [72]

Step 4. Create a Virtual Interface

When you create a virtual interface, you specify the following information:

- **Network ID**. The ID of the network for which you want to create a virtual interface.
- **Server ID**. The ID of the server instance to which you want to connect the virtual interface.

Next step: Choose one of the following methods:

- the section called "Create a Virtual Interface with the nova Client" [72]Create a virtual interface with the nova client
- the section called "Create Virtual Interface with cURL" [74]Create a virtual interface with cURL

Create a Virtual Interface with the nova Client

To create a virtual interface with the nova client

- Issue the following nova client command:

```
                         $ nova virtual-interface-create network_id instance_id
```

Positional arguments:

- *`network_id`*. The ID of the network for which you want to create a virtual interface. You copied this ID in the section called “Step 3. List Networks” [70].
- *`instance_id`*. The ID of the server instance to which you want to connect the virtual interface. You copied this ID in the section called “Step 2. List Servers” [66].

Next step:the section called “Step 5. List Virtual Interfaces for a Server” [75]

Create Virtual Interface with cURL

You can issue a JSON or XML request in a cURL command.

JSON Request and Response

To create a virtual interface and request a JSON response

- To create a virtual interface, you must specify the server ID in the request URI, and include a request body that specifies the network ID for which you want to create a virtual interface.

  For example, issue the following cURL command:

```
                            $ curl https://dfw.servers.api.rackspacecloud.com/v2/
$account/servers/$instance_id/os-virtual-interfacesv2 \
     -X POST \
     -H "Content-Type: application/json" \
     -H "Accept: application/json" \
     -H "X-Auth-Token: $token" \
     -d '{"virtual_interface": {"network_id": "network_id"}}'| python -m json.tool
```

  Arguments:

  - *instance_id*. The ID of the server instance to which you want to connect the virtual interface. You copied this ID in the section called "Step 3. List Networks" [70].
  - *network_id*. The ID of the network for which you want to create a virtual interface. You copied this ID in the section called "Step 2. List Servers" [66].

  Output:

```
                         {
   "virtual_interfaces":[
      {
         "mac_address":"FE:ED:FA:00:08:93",
         "id":"045f195f-3347-487b-8e80-8ee3390eda56",
         "ip_addresses":[
            {
               "address":"192.168.0.1",
               "network_id":"196a0246-86cc-46fa-9ecf-850f67c2cb7c",
               "network_label":"added_network"
            }
         ]
      }
   ]
}
```

Next step:the section called "Step 5. List Virtual Interfaces for a Server" [75]

XML Request and Response

To create a virtual interface and request an XML response

- To create a virtual interface, you must specify the server ID in the request URI, and include a request body that specifies the network ID for which you want to create a virtual interface.

For example, issue the following cURL command:

```
                            $ curl -i https://dfw.servers.api.rackspacecloud.com/v2/
$account/servers/$instance_id/os-virtual-interfacesv2.xml \
      -X POST \
      -H "Content-Type: application/xml" \
      -H "Accept: application/xml" \
      -H "X-Auth-Token: $token" \
      -d '<?xml version="1.0" encoding="UTF-8"?><virtual_interface network_id=
"network_id"/>'
```

Arguments:

- *instance_id*. The ID of the server instance to which you want to connect the virtual interface. You copied this ID in the section called “Step 2. List Servers” [66].
- *network_id*. The ID of the network for which you want to create a virtual interface. You copied this ID in the section called “Step 3. List Networks” [70].

Output:

```
                            <?xml version='1.0' encoding='UTF-8'?>
<virtual_interfaces xmlns="http://docs.openstack.org/compute/api/v1.1">
    <virtual_interface id="24293921-e8fe-4f93-ac52-b8cc08435d00"
        mac_address="FE:ED:FA:00:0D:13">
        <ip_address address="1.1.1.129"
            network_id="6d17d84a-9513-4c4c-bf5a-c2d5c0794292"
            network_label="added_network"/>
    </virtual_interface>
</virtual_interfaces>
```

Next step:the section called “Step 5. List Virtual Interfaces for a Server” [75]

Step 5. List Virtual Interfaces for a Server

To validate that you created and connected the virtual interface to your server, you can list the virtual interfaces for your server.

Next step: Choose one of the following methods:

- the section called “List Virtual Interfaces for a Server with the nova Client” [75]
- the section called “List Virtual Interfaces for a Server with cURL” [77]

List Virtual Interfaces for a Server with the nova Client

To list virtual interfaces for a server with the nova client

1. Issue the following nova client command:

```
                            $ nova virtual-interface-list instance_id
```

Positional argument:

- *instance_id*. The ID of the server instance for which you want to list virtual interfaces. You copied this ID in the section called “Step 2. List Servers” [66].

Output:

For any virtual interfaces that are connected to the specified server, the command returns the network ID, MAC address, and IP addresses, as shown in the following output:

```
+--------------------------------------+-------------------
+-----------------------------------------------------------------------------------
+
| id                                   | mac_address       | ip_addresses
                                                                              |
+--------------------------------------+-------------------
+-----------------------------------------------------------------------------------
+
| 398f4189-5a60-4a5e-8a68-71e4fa014313 | 00:00:00:00:00:01 | label=
private, network_id=39a43ded-7a9b-4a50-8633-e70d48363305, ip_address=172.
16.0.2 |
| d5e3c9b6-bd5d-46c9-ba7b-114df3f37fb3 | 00:00:00:00:00:00 | label=mypriv,
 network_id=1f7920d3-0e63-4fec-a1cb-f7916671e8eb, ip_address=10.1.0.3
 |
| d8a1baa8-3b51-4a74-9e33-f885e438a468 | 00:00:00:00:00:02 | label=public,
 network_id=69ebc6a6-27fc-4f47-aeca-de7c3b4685e7, ip_address=10.0.0.3    |
+--------------------------------------+-------------------
+-----------------------------------------------------------------------------------
+
```

2. If you want to delete this interface later on, copy the ID for the virtual interface.

Next step:the section called “Step 6. Delete a Virtual Interface from a Server” [78]

List Virtual Interfaces for a Server with cURL

You can request a JSON or XML response when you issue a request in a cURL command.

JSON Request and Response

To list virtual interfaces for a server and request a JSON response

1. Issue the following cURL command:

```
                            $ curl https://dfw.servers.api.rackspacecloud.com/v2/
$account/servers/$instance_id/os-virtual-interfacesv2  \
       -X GET \
       -H "X-Auth-Token: $token" | python -m json.tool
```

Argument:

- *instance_id*. The ID of the server instance for which you want to list virtual interfaces. You copied this ID in the section called "Step 2. List Servers" [66].

Output:

```
                            {
    "virtual_interfaces": [
        {
            "id": "a589b11b-cd51-4274-8ec0-832ce799d156",
            "ip_addresses": [
                {
                    "address": "2001:4800:7810:0512:d87b:9cbc:ff04:850c",
                    "network_id": "ba122b32-dbcc-4c21-836e-b701996baeb3",
                    "network_label": "public"
                },
                {
                    "address": "64.49.226.149",
                    "network_id": "ba122b32-dbcc-4c21-836e-b701996baeb3",
                    "network_label": "public"
                }
            ],
            "mac_address": "BC:76:4E:04:85:0C"
        },
        {
            "id": "de7c6d53-b895-4b4a-963c-517ccb0f0775",
            "ip_addresses": [
                {
                    "address": "192.168.0.2",
                    "network_id": "f212726e-6321-4210-9bae-a13f5a33f83f",
                    "network_label": "superprivate_xml"
                }
            ],
            "mac_address": "BC:76:4E:04:85:20"
        },
        {
            "id": "e14e789d-3b98-44a6-9c2d-c23eb1d1465c",
            "ip_addresses": [
                {
                    "address": "10.181.1.30",
                    "network_id": "3b324a1b-31b8-4db5-9fe5-4a2067f60297",
                    "network_label": "private"
                }
            ],
            "mac_address": "BC:76:4E:04:81:55"
        }
    ]
}
```

2. If you want to delete this interface later on, copy the ID for the virtual interface.

Next step:the section called “Step 6. Delete a Virtual Interface from a Server” [78]

XML Request and Response

To list virtual interfaces for a server and request a XML response

1. Issue the following cURL command:

```
                             $ curl -i https://dfw.servers.api.rackspacecloud.com/v2/
$account/servers/$instance_id/os-virtual-interfacesv2.xml  \
       -H "Content-Type: application/xml" \
       -H "Accept: application/xml" \
       -X GET \
       -H "X-Auth-Token: $token"
```

Argument:

- *instance_id*. The ID of the server instance for which you want to list virtual interfaces. You copied this ID in the section called “Step 2. List Servers” [66].

Output:

```
                            <?xml version='1.0' encoding='UTF-8'?>
<virtual_interfaces xmlns="http://docs.openstack.org/compute/api/v1.1">
    <virtual_interface id="a589b11b-cd51-4274-8ec0-832ce799d156" mac_address="BC:76:4E:04:85:0C">
        <ip_address network_id="ba122b32-dbcc-4c21-836e-b701996baeb3" network_label="public"
 address="2001:4800:7810:0512:d87b:9cbc:ff04:850c" />
        <ip_address network_id="ba122b32-dbcc-4c21-836e-b701996baeb3" network_label="public"
 address="64.49.226.149" />
    </virtual_interface>
    <virtual_interface id="de7c6d53-b895-4b4a-963c-517ccb0f0775" mac_address="BC:76:4E:04:85:20">
        <ip_address network_id="f212726e-6321-4210-9bae-a13f5a33f83f" network_label=
"superprivate_xml" address="192.168.0.2" />
    </virtual_interface>
    <virtual_interface id="e14e789d-3b98-44a6-9c2d-c23eb1d1465c" mac_address="BC:76:4E:04:81:55">
        <ip_address network_id="3b324a1b-31b8-4db5-9fe5-4a2067f60297" network_label="public"
 address="10.181.1.30" />
    </virtual_interface>
</virtual_interfaces>
```

2. If you want to delete this interface later on, copy the ID for the virtual interface.

Next step:the section called “Step 6. Delete a Virtual Interface from a Server” [78]

Step 6. Delete a Virtual Interface from a Server

If your server no longer needs the virtual interface, you can delete and detach it from the server.

To delete and detach your virtual interface from a server, you need the ID of your server instance and the ID for the virtual interface.

After you delete the virtual interface, it no longer appears in the list of virtual interfaces for your server.

Next step: Choose one of the following methods:

- the section called “Delete a Virtual Interface with the nova Client” [79]
- the section called “Delete a Virtual Interface with cURL” [80]

Delete a Virtual Interface with the nova Client

To delete a virtual interface with the nova client:

1. Issue the following command:

```
                             $ nova virtual-interface-
delete instance_id interface_id
```

   Positional arguments:

   - *instance_id*. The ID of the server instance from which you want to detach the virtual interface. You copied this ID in the section called “Step 2. List Servers” [66].
   - *interface_id*. The ID of the virtual interface that you want to delete. You copied this ID in the section called “Step 5. List Virtual Interfaces for a Server” [75].

2. Issue the following nova client command to determine if your virtual interface was deleted:

```
                             $ nova virtual-interface-list instance_id
```

   Positional argument:

   - *instance_id*. The ID of the server instance for which you want to list virtual interfaces.

   After the virtual interface is deleted, it no longer appears in the list of virtual interfaces. See the section called “Step 5. List Virtual Interfaces for a Server” [75].

Delete a Virtual Interface with cURL

To delete a virtual interface with cURL:

1. Issue the following command. In the command, specify the ID for the server that you want to delete.

   ```
                            $ curl -i https://dfw.servers.api.rackspacecloud.com/v2/$account/servers/
   $instance_id/os-virtual-interfacesv2/$interface_id \
        -X DELETE \
        -H "X-Auth-Token: $token"
   ```

   Arguments:

   - *`instance_id`*. The ID of the server instance from which you want to detach the virtual interface. You copied this ID in the section called "Step 2. List Servers" [66].
   - *`interface_id`*. The ID of the virtual interface that you want to delete. You copied this ID in the section called "Step 5. List Virtual Interfaces for a Server" [75].

   Note

   Because the delete server method does not return a JSON object, you omit the following code from the command:

   ```
   | python -m json.tool
   ```

   If you include it, the command runs successfully, but you get the following message:

   ```
   No JSON object could be decoded.
   ```

   To see the status code that is returned from the request, you use the `-i` option on the cURL command. This option shows the HTTP header in the output.

   If the delete operation was successful, the HTTP header shows the 204 status code, as shown in the following output:

   ```
   HTTP/1.1 204 No Content
   Date: Thu, 16 Aug 2012 17:19:53 GMT
   Content-Length: 0
   Content-Type: application/json
   X-Compute-Request-Id: req-3bdafeb2-d4b1-41c3-ab19-d310f3f270d3
   Server: Jetty(8.0.y.z-SNAPSHOT)
   ```

2. Issue the following cURL command to determine if your virtual interface was deleted:

   ```
                          $ curl https://dfw.servers.api.rackspacecloud.com/v2/
   $account/servers/$instance_id/os-virtual-interfacesv2  \
          -X GET \
          -H "X-Auth-Token: $token" | python -m json.tool
   ```

   Argument:

   - *`instance_id`*. The ID of the server instance for which you want to list virtual interfaces. You copied this ID in the section called "Step 2. List Servers" [66].

After the virtual interface is deleted, it no longer appears in the list of virtual interfaces.

Glossary

CIDR
: Classless Inter-Domain Routing (CIDR). A method for allocating IP addresses and routing Internet Protocol packets. Used with Cloud Networks.

HTTP
: HyperText Transfer Protocol. The protocol that tells browsers where to find information.

PublicNet
: Provides access to the Internet, Rackspace services such as Cloud Monitoring, Managed Cloud Service Level, RackConnect, Cloud Backup, and certain operating system updates. When you list networks through Cloud Networks, PublicNet is labeled `public`.

REST
: REpresentational State Transfer. A style of architecture for hypermedia systems that is used for the World Wide web.

RESTful
: A kind of web service API that uses REST.

server
: A computer that provides explicit services to the client software running on that system. A server is a virtual machine (VM) instance in the Cloud Servers environment. To create a server, you must specify a name, flavor reference, and image reference.

ServiceNet
: Provides access to Rackspace services such as Cloud Files, Cloud Databases, and Cloud Backup, and to certain packages and patches through an internal only, multi-tenant network connection within each Rackspace data center. When you list networks through Cloud Networks, ServiceNet is labeled as `private`.