# OpenStack

## End User Guide

current (January 11, 2015)

openstack™

# OpenStack End User Guide

OpenStack is an open-source cloud computing platform for public and private clouds. A series of interrelated projects deliver a cloud infrastructure solution. This guide shows OpenStack end users how to create and manage resources in an OpenStack cloud with the OpenStack dashboard and OpenStack client commands.

This guide documents OpenStack Juno, OpenStack Icehouse and OpenStack Havana releases.

# Table of Contents

# List of Figures

# List of Tables

# List of Examples

# How can I use an OpenStack cloud?

As an OpenStack cloud end user, you can provision your own resources within the limits set by administrators.

The examples in this guide show you how to perform tasks by using the following methods:

- OpenStack dashboard. Use this web-based graphical interface, code named horizon, to view, create, and manage resources.

- OpenStack command-line clients. Each core OpenStack project has a command-line client that you can use to run simple commands to view, create, and manage resources in a cloud and automate tasks by using scripts.

You can modify these examples for your specific use cases.

In addition to these ways of interacting with a cloud, you can access the OpenStack APIs directly or indirectly through cURL commands or open SDKs. You can automate access or build tools to manage resources and services by using the native OpenStack APIs or the EC2 compatibility API.

To use the OpenStack APIs, it helps to be familiar with HTTP/1.1, RESTful web services, the OpenStack services, and JSON or XML data serialization formats.

# Who should read this book?

This book is written for anyone who uses virtual machines and cloud resources to develop software or perform research. You should have years of experience with Linux-based tool sets and be comfortable using both GUI and CLI based tools. While this book includes some information about using Python to create and manage cloud resources, Python knowledge is not a pre-requisite for reading this book.

# Conventions

The OpenStack documentation uses several typesetting conventions.

# Notices

Notices take these forms:

### Note

A handy tip or reminder.

### Important

Something you must be aware of before proceeding.

### Warning

Critical information about the risk of data loss or security issues.

# Command prompts

**$ prompt**      Any user, including the `root` user, can run commands that are prefixed with
                  the $ prompt.

**# prompt**      The `root` user must run commands that are prefixed with the # prompt. You
                  can also prefix these commands with the **sudo** command, if available, to run
                  them.

# Document change history

This version of the guide replaces and obsoletes all earlier versions.

The following table describes the most recent changes:

| Revision Date | Summary of Changes |
|---|---|
| October 15, 2014 | • For the Juno release, this guide has been updated with information about the Database Service for OpenStack (trove). |
| May 9, 2014 | • Add a command line cheat sheet. |
| April 17, 2014 | • For the Icehouse release, this guide has been updated with changes to the dashboard plus the moving of the command reference appendix as indicated below. |
| January 31, 2014 | • Removed the command reference appendix. This information is now in the *OpenStack Command-Line Interface Reference*. |
| December 30, 2013 | • Added the OpenStack Python SDK chapter. |
| October 17, 2013 | • Havana release. |
| August 19, 2013 | • Editorial changes. |
| July 29, 2013 | • First edition of this document. |

# 1. OpenStack dashboard

## Table of Contents

As a cloud end user, you can use the OpenStack dashboard to provision your own resources within the limits set by administrators. You can modify the examples provided in this section to create other types and sizes of server instances.

# Log in to the dashboard

The dashboard is available on the node with the `nova-dashboard` server role.

1. Ask the cloud operator for the host name or public IP address from which you can access the dashboard, and for your user name and password.

2. Open a web browser that has JavaScript and cookies enabled.

   **Note**

   To use the Virtual Network Computing (VNC) client for the dashboard, your browser must support HTML5 Canvas and HTML5 WebSockets. The VNC client is based on noVNC. For details, see noVNC: HTML5 VNC Client. For a list of supported browsers, see Browser support.

3. In the address bar, enter the host name or IP address for the dashboard.

   ```
   https://ipAddressOrHostName/
   ```

   **Note**

   If a certificate warning appears when you try to access the URL for the first time, a self-signed certificate is in use, which is not considered trustworthy by default. Verify the certificate or add an exception in the browser to bypass the warning.

4. On the **Log In** page, enter your user name and password, and click **Sign In**.

   The top of the window displays your user name. You can also access **Settings** or sign out of the dashboard.

   The visible tabs and functions in the dashboard depend on the access permissions, or *roles*, of the user you are logged in as.

   • If you are logged in as an end user, the Project tab is displayed.

   • If you are logged in as an administrator, the Project tab and Admin tab are displayed.

# OpenStack dashboard—Project tab

Projects are organizational units in the cloud, and are also known as tenants or accounts. Each user is a member of one or more projects. Within a project, a user creates and manages instances.

From the **Project** tab, you can view and manage the resources in a selected project, including instances and images. You select the project from the **CURRENT PROJECT** list at the top of the tab.

**Figure 1.1. Project tab**
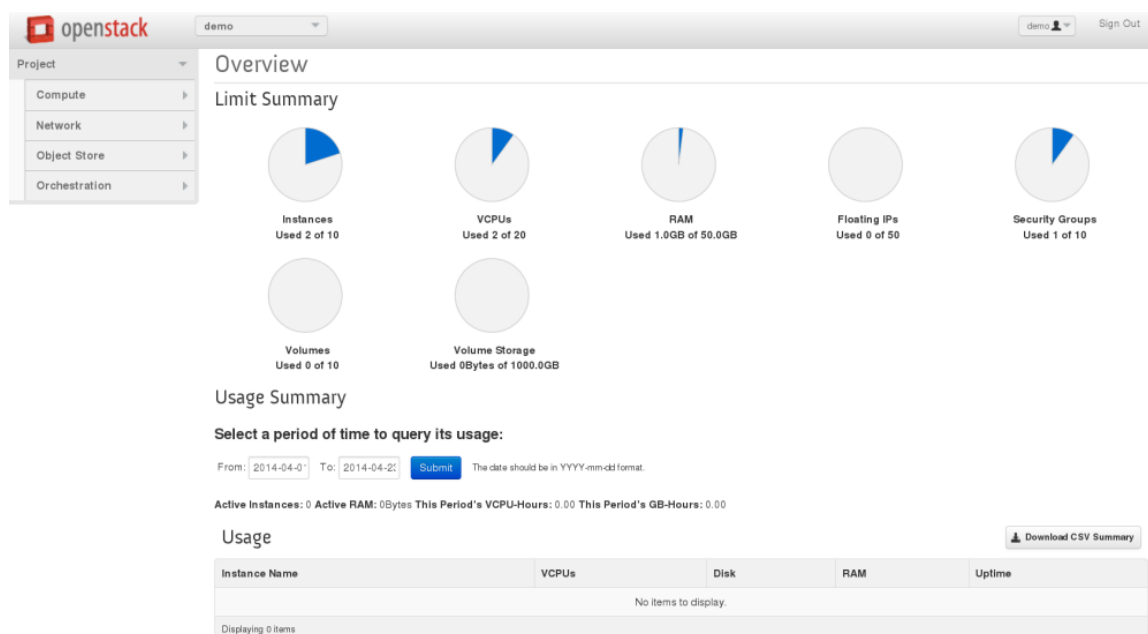


From the **Project** tab, you can access the following tabs:

| Compute tab | | |
|---|---|---|
| **Overview** | View reports for the project. | |
| **Instances** | View, launch, create a snapshot from, stop, pause, or reboot instances, or connect to them through VNC. | |
| **Volumes** | Use the following tabs to complete these tasks: | |
| | **Volumes** | View, create, edit, and delete volumes. |
| | **Volume Snapshots** | View, create, edit, and delete volume snapshots. |
| **Images** | View images and instance snapshots created by project users, plus any images that are publicly available. Create, edit, and delete images, and launch instances from images and snapshots. | |
| **Access & Security** | Use the following tabs to complete these tasks: | |
| | **Security Groups** | View, create, edit, and delete security groups and security group rules. |
| | **Key Pairs** | View, create, edit, import, and delete key pairs. |
| | **Floating IPs** | Allocate an IP address to or release it from a project. |
| | **API Access** | View API endpoints. |
| **Network** tab | | |
| **Network Topology** | View the network topology. | |
| **Networks** | Create and manage public and private networks. | |
| **Routers** | Create and manage subnets. | |
| **Object Store** tab | | |

| Compute tab | |
|---|---|
| Containers | Create and manage containers and objects. |
| **Orchestration** tab | |
| Stacks | Use the REST API to orchestrate multiple composite cloud applications. |

# OpenStack dashboard—Admin tab

Administrative users can use the **Admin** tab to view usage and to manage instances, volumes, flavors, images, projects, users, services, and quotas.

### Figure 1.2. Admin tab
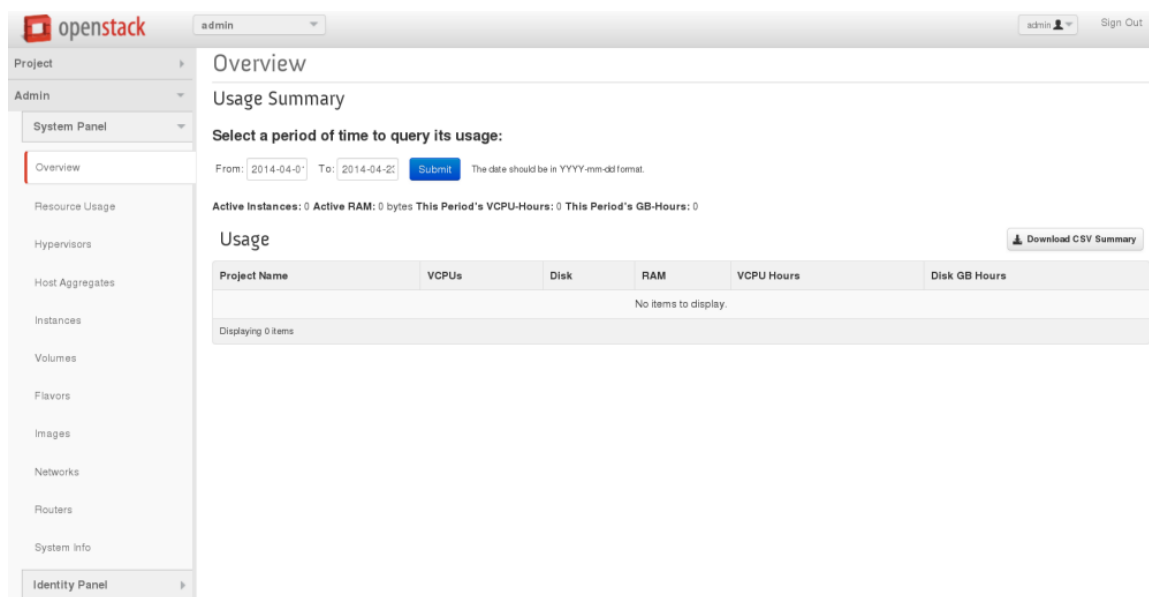


Access the following categories to complete these tasks:

| System Panel tab | |
|---|---|
| **Overview** | View basic reports. |
| **Resource Usage** | Use the following tabs to view the following usages:<br><br>**Daily Report**    View the daily report.<br><br>**Stats**    View the statistics of all resources. |
| **Hypervisors** | View the hypervisor summary. |
| **Host Aggregates** | View, create, and edit host aggregates. View the list of availability zones. |
| **Instances** | View, pause, resume, suspend, migrate, soft or hard reboot, and delete running instances that belong to users of some, but not all, projects. Also, view the log for an instance or access an instance through VNC. |
| **Volumes** | View, create, edit, and delete volumes and volume types. |
| **Flavors** | View, create, edit, view extra specifications for, and delete flavors. A flavor is size of an instance. |

| System Panel tab | |
|---|---|
| Images | View, create, edit properties for, and delete custom images. |
| Networks | View, create, edit properties for, and delete networks. |
| Routers | View, create, edit properties for, and delete routers. |
| System Info | Use the following tabs to view the service information:<br><br>**Services** — View a list of the services.<br><br>**Compute Services** — View a list of all Compute services.<br><br>**Network Agents** — View the network agents.<br><br>**Default Quotas** — View default quota values. Quotas are hard-coded in OpenStack Compute and define the maximum allowable size and number of resources. |
| **Identity Panel** tab | |
| Projects | View, create, assign users to, remove users from, and delete projects. |
| Users | View, create, enable, disable, and delete users. |

# Upload and manage images

A virtual machine image, referred to in this document simply as an image, is a single file that contains a virtual disk that has a bootable operating system installed on it. Images are used to create virtual machine instances within the cloud. For information about creating image files, see the *OpenStack Virtual Machine Image Guide*.

Depending on your role, you may have permission to upload and manage virtual machine images. Operators might restrict the upload and management of images to cloud administrators or operators only. If you have the appropriate privileges, you can use the dashboard to upload and manage images in the **admin** project.

> **Note**
>
> You can also use the **glance** and **nova** command-line clients or the Image Service and Compute APIs to manage images. See the section called "Manage images" [40].

## Upload an image

Follow this procedure to upload an image to a project.

1.  Log in to the dashboard.

2.  From the **CURRENT PROJECT** on the **Project** tab, select the appropriate project.

3.  On the **Project** tab, click **Images**.

4.  Click **Create Image**.

    The Create An Image dialog box appears.

5.  Enter the following values:

| | |
|---|---|
| **Name** | Enter a name for the image. |
| **Description** | Optionally, enter a brief description of the image. |
| **Image Source** | Choose the image source from the list. Your choices are **Image Location** and **Image File**. |
| **Image File** or **Image Location** | Based on your selection for **Image Source**, you either enter the location URL of the image in the **Image Location** field. or browse to the image file on your system and add it. |
| **Format** | Select the correct format (for example, QCOW2) for the image. |
| **Architecture** | Specify the architecture. For example, `i386` for a 32-bit architecture or `x86-64` for a 64-bit architecture. |
| **Minimum Disk (GB)** and **Minimum RAM (MB)** | Leave these optional fields empty. |
| **Public** | Select this check box to make the image public to all users with access to the current project. |
| **Protected** | Select this check box to ensure that only users with permissions can delete the image. |

6.  Click **Create Image**.

    The image is queued to be uploaded. It might take some time before the status changes from Queued to Active.

# Update an image

Follow this procedure to update an existing image.

1.  Log in to the dashboard.

2.  From the **CURRENT PROJECT** on the **Project** tab, select the appropriate project.

3.  On the **Project** tab, click **Images**.

4.  Select the image that you want to edit.

5.  In the **Actions** column, click **More** and then select **Edit** from the list.

6.  In the Update Image dialog box, you can perform the following actions:

    • Change the name of the image.

    • Select the **Public** check box to make the image public.

    • Clear the **Public** check box to make the image private.

7.  Click **Update Image**.

# Delete an image

Deletion of images is permanent and **cannot** be reversed. Only users with the appropriate permissions can delete images.

1.  Log in to the dashboard.

2.  From the **CURRENT PROJECT** on the **Project** tab, select the appropriate project.

3.  On the **Project** tab, click **Images**.

4.  Select the images that you want to delete.

5.  Click **Delete Images**.

6.  In the **Confirm Delete Image** dialog box, click **Delete Images** to confirm the deletion.

# Configure access and security for instances

Before you launch an instance, you should add security group rules to enable users to ping and use SSH to connect to the instance. Security groups are sets of IP filter rules that define networking access and are applied to all instances within a project. To do so, you either add rules to the default security group or add a new security group with rules.

Key pairs are SSH credentials that are injected into an instance when it is launched. To use key pair injection, the image that the instance is based on must contain the `cloud-init` package. Each project should have at least one key pair. For more information, see the section called "Add a key pair" [9].

If you have generated a key pair with an external tool, you can import it into OpenStack. The key pair can be used for multiple instances that belong to a project. For more information, see the section called "Import a key pair" [9].

When an instance is created in OpenStack, it is automatically assigned a fixed IP address in the network to which the instance is assigned. This IP address is permanently associated with the instance until the instance is terminated. However, in addition to the fixed IP address, a floating IP address can also be attached to an instance. Unlike fixed IP addresses, floating IP addresses are able to have their associations modified at any time, regardless of the state of the instances involved.

## Add a rule to the default security group

This procedure enables SSH and ICMP (ping) access to instances. The rules apply to all instances within a given project, and should be set for every project unless there is a reason to prohibit SSH or ICMP access to the instances.

This procedure can be adjusted as necessary to add additional security group rules to a project, if your cloud requires them.

> **Note**
>
> When adding a rule, you must specify the protocol used with the destination port or source port.

1. Log in to the dashboard, choose a project, and click **Access & Security**. The **Security Groups** tab shows the security groups that are available for this project.

2. Select the **default** security group and click **Edit Rules**.

3. To allow SSH access, click **Add Rule**.

4. In the Add Rule dialog box, enter the following values:

| Rule | SSH |
|---|---|
| Remote | CIDR |
| CIDR | 0.0.0.0/0 |

**Note**

To accept requests from a particular range of IP addresses, specify the IP address block in the **CIDR** box.

5.  Click **Add**.

    Instances will now have SSH port 22 open for requests from any IP address.

6.  To add an ICMP rule, click **Add Rule**.

7.  In the Add Rule dialog box, enter the following values:

    | Rule | All ICMP |
    | --- | --- |
    | **Direction** | Ingress |
    | **Remote** | CIDR |
    | **CIDR** | 0.0.0.0/0 |

8.  Click **Add**.

    Instances will now accept all incoming ICMP packets.

# Add a key pair

Create at least one key pair for each project.

1.  Log in to the dashboard, choose a project, and click **Access & Security**.

2.  Click the **Keypairs** tab, which shows the key pairs that are available for this project.

3.  Click **Create Keypair**.

4.  In the Create Keypair dialog box, enter a name for your key pair, and click **Create Keypair**.

5.  Respond to the prompt to download the key pair.

# Import a key pair

1.  Log in to the dashboard, choose a project, and click **Access & Security**.

2.  Click the **Keypairs** tab, which shows the key pairs that are available for this project.

3.  Click **Import Keypair**.

4.  In the Import Keypair dialog box, enter the name of your key pair, copy the public key into the **Public Key** box, and then click **Import Keypair**.

5.  Save the `*.pem` file locally.

6.  To change its permissions so that only you can read and write to the file, run the following command:

```
$ chmod 0600 yourPrivateKey.pem
```

### Note

If you are using the dashboard from a Windows computer, use PuTTYgen to load the `*.pem` file and convert and save it as `*.ppk`. For more information see the WinSCP web page for PuTTYgen.

7.    To make the key pair known to SSH, run the **ssh-add** command.

```
$ ssh-add yourPrivateKey.pem
```

The Compute database registers the public key of the key pair.

The dashboard lists the key pair on the **Access & Security** tab.

# Allocate a floating IP address to an instance

When an instance is created in OpenStack, it is automatically assigned a fixed IP address in the network to which the instance is assigned. This IP address is permanently associated with the instance until the instance is terminated.

However, in addition to the fixed IP address, a floating IP address can also be attached to an instance. Unlike fixed IP addresses, floating IP addresses can have their associations modified at any time, regardless of the state of the instances involved. This procedure details the reservation of a floating IP address from an existing pool of addresses and the association of that address with a specific instance.

1.    Log in to the dashboard, choose a project, and click **Access & Security**.

2.    Click the **Floating IPs** tab, which shows the floating IP addresses allocated to instances.

3.    Click **Allocate IP to Project**.

4.    Choose the pool from which to pick the IP address.

5.    Click **Allocate IP**.

6.    In the **Floating IPs** list, click  **Associate**.

7.    In the Manage Floating IP Associations dialog box, choose the following options:

   • The **IP Address** field is filled automatically, but you can add a new IP address by clicking the + button.

   • In the **Ports to be associated** field, select a port from the list.

      The list shows all the instances with their fixed IP addresses.

8.    Click **Associate**.

### Note

To disassociate an IP address from an instance, click the **Disassociate** button.

To release the floating IP address back into the pool of addresses, click the **More** button and select the **Release Floating IP** option.

# Launch and manage instances

Instances are virtual machines that run inside the cloud.

You can launch an instance from the following sources:

• Images uploaded to the OpenStack Image Service, as described in the section called "Upload and manage images" [6].

• Image that you have copied to a persistent volume. The instance launches from the volume, which is provided by the `cinder-volume` API through iSCSI.

# Launch an instance

When you launch an instance from an image, OpenStack creates a local copy of the image on the compute node where the instance starts.

When you launch an instance from a volume, note the following steps:

- To select the volume to from which to launch, launch an instance from an arbitrary image on the volume. The image that you select does not boot. Instead, it is replaced by the image on the volume that you choose in the next steps.

  To boot a Xen image from a volume, the image you launch in must be the same type, fully virtualized or paravirtualized, as the one on the volume.

- Select the volume or volume snapshot from which to boot. Enter a device name. Enter `vda` for KVM images or `xvda` for Xen images.

1. Log in to the dashboard, choose a project, and click **Images**.

   The dashboard shows the images that have been uploaded to OpenStack Image Service and are available for this project.

   For details on creating images, see Creating images manually in the *OpenStack Virtual Machine Image Guide*.

2. Select an image and click **Launch**.

3. In the Launch Instance dialog box, specify the following values:

<table>
<tr><td colspan="2" align="center"><strong>Details tab</strong></td></tr>
<tr><td><strong>Availability Zone</strong></td><td>By default, this value is set to the availability zone given by the cloud provider (for example, <code>us-west</code> or <code>apac-south</code>). For some cases, it could be <code>nova</code>.</td></tr>
<tr><td><strong>Instance Name</strong></td><td>Assign a name to the virtual machine.<br><br>**Note**<br><br>The name you assign here becomes the initial host name of the server. After the server is built, if you change the server name in the API or change the host name directly, the names are not updated in the dashboard.<br><br>Server names are not guaranteed to be unique when created so you could have two instances with the same host name.</td></tr>
<tr><td><strong>Flavor</strong></td><td>Specify the size of the instance to launch.<br><br>**Note**<br><br>The flavor is selected based on the size of the image selected for launching an instance. For example, while creating an image, if you have entered the value in the **Minimum RAM (MB)** field as 2048, then on selecting the image, the default flavor is **m1.small**.</td></tr>
</table>

| Details tab | |
| --- | --- |
| **Instance Count** | To launch multiple instances, enter a value greater than 1. The default is 1. |
| **Instance Boot Source** | Your options are: |
| | **Boot from image** — If you choose this option, a new field for **Image Name** displays. You can select the image from the list. |
| | **Boot from snapshot** — If you choose this option, a new field for **Instance Snapshot** displays. You can select the snapshot from the list. |
| | **Boot from volume** — If you choose this option, a new field for **Volume** displays. You can select the volume from the list. |
| | **Boot from image (creates a new volume)** — With this option, you can boot from an image and create a volume by entering the **Device Size** and **Device Name** for your volume. Click the **Delete on Terminate** option to delete the volume on terminating the instance. |
| | **Boot from volume snapshot (creates a new volume)** — Using this option, you can boot from a volume snapshot and create a new volume by choosing **Volume Snapshot** from a list and adding a **Device Name** for your volume. Click the **Delete on Terminate** option to delete the volume on terminating the instance. |
| | Since you are launching an instance from an image, **Boot from image** is chosen by default. |
| **Image Name** | This field changes based on your previous selection. Since you have chosen to launch an instance using an image, the **Image Name** field displays. Select the image name from the dropdown list. |
| Access & Security  tab | |
| **Keypair** | Specify a key pair. |
| | If the image uses a static root password or a static key set (neither is recommended), you do not need to provide a key pair to launch the instance. |
| **Security Groups** | Activate the security groups that you want to assign to the instance. |

| Details tab | |
|---|---|
| | Security groups are a kind of cloud firewall that define which incoming network traffic is forwarded to instances. For details, see the section called "Add a rule to the default security group" [8].<br><br>If you have not created any security groups, you can assign only the default security group to the instance. |
| **Networking** tab | |
| **Selected Networks** | To add a network to the instance, click the + in the **Available Networks** field. |
| **Post-Creation** tab | |
| **Customization Script** | Specify a customization script that runs after your instance launches. |
| **Advanced Options** tab | |
| **Disk Partition** | Select the type of disk partition from the dropdown list. |
| | **Automatic** — Entire disk is single partition and automatically resizes. |
| | **Manual** — Faster build times but requires manual partitioning. |

4.   Click **Launch**.

The instance starts on a compute node in the cloud.

The **Instances** tab shows the instance's name, its private and public IP addresses, size, status, task, and power state.

If you did not provide a key pair, security groups, or rules, users can access the instance only from inside the cloud through VNC. Even pinging the instance is not possible without an ICMP rule configured. To access the instance through a VNC console, see the section called "Access an instance through a console" [68].

# Connect to your instance by using SSH

To use SSH to connect to your instance, you use the downloaded keypair file.

### Note

The user name is `ubuntu` for the Ubuntu cloud images on TryStack.

1.  Copy the IP address for your instance.

2.  Use the **ssh** command to make a secure connection to the instance. For example:

    ```
    $ ssh -i MyKey.pem ubuntu@10.0.0.2
    ```

3.  At the prompt, type `yes`.

# Track usage for instances

You can track usage for instances for each project. You can track costs per month by showing metrics like number of vCPUs, disks, RAM, and uptime for all your instances.

1.  Log in to the dashboard, choose a project, and click **Overview**.

2.  To query the instance usage for a month, select a month and click **Submit**.

3.  To download a summary, click **Download CSV Summary**.

# Create an instance snapshot

1.  Log in to the dashboard, choose a project, and click **Instances**.

2.  Select the instance from which to create a snapshot.

3.  In the **Actions** column, click **Create Snapshot**.

4.  In the Create Snapshot dialog box, enter a name for the snapshot, and click **Create Snapshot**.

    The **Images** category shows the instance snapshot.

To launch an instance from the snapshot, select the snapshot and click **Launch**. Proceed with the section called "Launch an instance" [13].

# Manage an instance

1.  Log in to the dashboard, choose a project, and click **Instances**.

2.  Select an instance.

3.  In the **More** list in the **Actions** column, select the state.

    You can resize or rebuild an instance. You can also choose to view the instance console log, edit instance or the security groups. Depending on the current state of the instance, you can pause, resume, suspend, soft or hard reboot, or terminate it.

# Create and manage networks

The OpenStack Networking service provides a scalable system for managing the network connectivity within an OpenStack cloud deployment. It can easily and quickly react to changing network needs (for example, creating and assigning new IP addresses).

Networking in OpenStack is complex. This section provides the basic instructions for creating a network and a router. For detailed information about managing networks, refer to the *OpenStack Cloud Administrator Guide*.

## Create a network

1. Log in to the dashboard, choose a project, and click **Networks**.

2. Click **Create Network**.

3. In the Create Network dialog box, specify the following values.

| Network tab | |
|---|---|
| **Network Name** | Specify a name to identify the network. |
| **Subnet** tab | |
| **Create Subnet** | Select this check box to create a subnet<br><br>You do not have to specify a subnet when you create a network, but if you do not, any attached instance receives an Error status. |
| **Subnet Name** | Specify a name for the subnet. |
| **Network Address** | Specify the IP address for the subnet. |
| **IP Version** | Select IPv4 or IPv6. |
| **Gateway IP** | Specify an IP address for a specific gateway. This parameter is optional. |
| **Disable Gateway** | Select this check box to disable a gateway IP address. |
| **Subnet Detail** tab | |
| **Enable DHCP** | Select this check box to enable DHCP. |
| **Allocation Pools** | Specify IP address pools. |
| **DNS Name Servers** | Specify a name for the DNS server. |
| **Host Routes** | Specify the IP address of host routes. |

4. Click **Create**.

   The dashboard shows the network on the **Networks** tab.

## Create a router

1. Log in to the dashboard, choose a project, and click **Routers**.

2. Click **Create Router**.

3. In the Create Router dialog box, specify a name for the router and click **Create Router**.

   The new router is now displayed in the **Routers** tab.

4. Click the new router's **Set Gateway** button.

5. In the **External Network** field, specify the network to which the router will connect, and then click **Set Gateway**.

6. To connect a private network to the newly created router, perform the following steps:

    a. On the **Routers** tab, click the name of the router.

    b. On the Router Details page, click **Add Interface**.

    c. In the Add Interface dialog box, specify the following information:

| | |
|---|---|
| **Subnet** | Select a subnet. |
| **IP Address (optional)** | Enter the router interface IP address for the selected subnet.<br><br>Note: If this value is not set, then by default, the first host IP address in the subnet is used by OpenStack Networking. |

    The **Router Name** and **Router ID** fields are automatically updated.

7. Click **Add Interface**.

You have successfully created the router. You can view the new topology from the **Network Topology** tab.

# Create and manage object containers

OpenStack Object Storage provides a distributed, API-accessible storage platform that can be integrated directly into an application or used to store any type of file, including VM images, backups, archives, or media files. In the OpenStack Dashboard, you can only manage containers and objects.

In OpenStack Object Storage, containers provide storage for objects in a manner similar to a Windows folder or Linux file directory, though they cannot be nested. An object in OpenStack consists of the file to be stored in the container and any accompanying metadata.

## Create a container

1.  Log in to the dashboard, choose a project, and click **Containers**.

2.  Click **Create Container**.

3.  In the Create Container dialog box, enter a name for the container, and then click **Create Container**.

You have successfully created a container.

> **Note**
>
> To delete a container, click the **More** button and select **Delete Container**.

## Upload an object

1.  Log in to the dashboard, choose a project, and click **Containers**.

2.  Select the container in which you want to store your object.

3.  Click **Upload Object**.

    The Upload Object To Container: *<name>* dialog box is displayed. *<name>* is the name of the container to which you are uploading the object.

4.  Enter a name for the object.

5.  Browse to and select the file that you want to upload.

6.  Click **Upload Object**.

You have successfully uploaded an object to the container.

> **Note**
>
> To delete an object, click the **More** button and select **Delete Object**.

## Manage an object

### To edit an object

1.  Log in to the dashboard, choose a project, and click **Containers**.

2.    Select the container in which you want to store your object.

3.    Click **More** and choose **Edit** from the dropdown list.

      The Edit Object dialog box is displayed.

4.    Browse to and select the file that you want to upload.

5.    Click **Update Object**.

### Note

To delete an object, click the **More** button and select **Delete Object**.

## To copy an object from one container to another

1.    Log in to the dashboard, choose a project, and click **Containers**.

2.    Select the container in which you want to store your object.

3.    Click **More** and choose **Copy** from the dropdown list.

4.    In the Copy Object: launch dialog box, enter the following values:

      • **Destination Container**: Choose the destination container from the list.

      • **Path**: Specify a path in which the new copy should be stored inside of the selected container.

      • **Destination object name**: Enter a name for the object in the new container.

5.    Click **Copy Object**.

## To create a metadata-only object without a file

You can create a new object in container without a file available and can upload the file later when it is ready. This temporary object acts a place-holder for a new object, and enables the user to share object metadata and URL info in advance.

1.    Log in to the dashboard, choose a project, and click **Containers**.

2.    Select the container in which you want to store your object.

3.    Click **Upload Object**.

      The Upload Object To Container: *<name>* dialog box is displayed. *<name>* is the name of the container to which you are uploading the object.

4.    Enter a name for the object.

5.    Click **Update Object**.

## To create a pseudo-folder

Pseudo-folders are similar to folders in your desktop operating system. They are virtual collections defined by a common prefix on the object's name.

1. Log in to the dashboard, choose a project, and click **Containers**.

2. Select the container in which you want to store your object.

3. Click **Create Pseudo-folder**.

   The Create Pseudo-Folder in Container *<name>* dialog box is displayed. *<name>* is the name of the container to which you are uploading the object.

4. Enter a name for the pseudo-folder.

   A slash (/) character is used as the delimiter for pseudo-folders in the Object Store.

5. Click **Create**.

# Create and manage volumes

Volumes are block storage devices that you attach to instances to enable persistent storage. You can attach a volume to a running instance or detach a volume and attach it to another instance at any time. You can also create a snapshot from or delete a volume. Only administrative users can create volume types.

## Create a volume

1. Log in to the dashboard, choose a project, and click **Volumes**.

2. Click **Create Volume**.

   In the dialog box that opens, enter or select the following values.

| | |
|---|---|
| **Volume Name** | Specify a name for the volume. |
| **Description** | Optionally, provide a brief description for the volume. |
| **Type** | Leave this field blank. |
| **Size (GB)** | The size of the volume in gigabytes. |
| **Volume Source** | Select one of the following options: |
| | **No source, empty volume** — Creates an empty volume. <br><br> **Note** <br> An empty volume does not contain a file system or a partition table. |
| | **Snapshot** — If you choose this option, a new field for **Use snapshot as a source** displays. You can select the snapshot from the list. |
| | **Image** — If you choose this option, a new field for **Use image as a source** displays. You can select the image from the list. <br><br> Select the **Availability Zone** from the list. By default, this value is set to the availability zone given by the cloud provider (for example, `us-west` or `apac-south`). For some cases, it could be `nova`. |
| | **Volume** — If you choose this option, a new field for **Use** |

**volume as a source** dis-
plays. You can select the
volume from the list.

### Note

Options to use a snapshot or a volume as
the source for a volume are displayed only
if there are existing snapshots or volumes.

3. Click **Create Volume**.

The dashboard shows the volume on the **Volumes** tab.

# Attach a volume to an instance

After you create one or more volumes, you can attach them to instances. You can attach a
volume to one instance at a time.

1. Log in to the dashboard, choose a project, and click **Volumes**.

2. Select the volume to add to an instance and click **Edit Attachments**.

3. In the **Manage Volume Attachments** dialog box, select an instance.

4. Enter the name of the device from which the volume is accessible by the instance.

### Note

The actual device name might differ from the volume name because of hy-
pervisor settings.

5. Click **Attach Volume**.

The dashboard shows the instance to which the volume is now attached and the de-
vice name.

You can view the status of a volume in the **Volumes** tab of the dashboard. The volume is ei-
ther Available or In-Use.

Now you can log in to the instance and mount, format, and use the disk.

# Detach a volume from an instance

1. Log in to the dashboard, choose a project, and click **Volumes**.

2. Select the volume and click **Edit Attachments**.

3. Click **Detach Volume** and confirm your changes.

A message indicates whether the action was successful.

# Create a snapshot from a volume

1. Log in to the dashboard, choose a project, and click **Volumes**.

2.    Select a volume from which to create a snapshot.

3.    From the **More** list, select **Create Snapshot**.

4.    In the dialog box that opens, enter a snapshot name and a brief description.

5.    Confirm your changes.

      The dashboard shows the new volume snapshot in **Volume Snapshots** tab.

# Edit a volume

1.    Log in to the dashboard, choose a project, and click **Volumes**.

2.    From the **CURRENT PROJECT** on the **Project** tab, select the appropriate project.

3.    On the **Project** tab, click **Volumes**.

4.    Select the image that you want to edit.

5.    In the **Actions** column, click **Edit Volume**.

6.    In the **Edit Volume** dialog box, update the name and description of the image.

7.    Click **Edit Volume**.

### Note

You can extend a volume by using the **Extend Volume** option available in the **More** dropdown list and entering the new value for volume size.

# Delete a volume

When you delete an instance, the data in its attached volumes is not destroyed.

1.    Log in to the dashboard, choose a project, and click **Volumes**.

2.    Select the check boxes for the volumes that you want to delete.

3.    Click **Delete Volumes** and confirm your choice.

      A message indicates whether the action was successful.

# Launch and manage stacks

OpenStack Orchestration is a service that you can use to orchestrate multiple composite cloud applications. This service supports use of both the Amazon Web Services (AWS) CloudFormation template format through both a Query API that is compatible with CloudFormation and the native OpenStack *Heat Orchestration Template (HOT)* format through a REST API.

These flexible template languages enable application developers to describe and automate the deployment of infrastructure, services, and applications. The templates enable creation of most OpenStack resource types, such as instances, floating IP addresses, volumes, security groups, and users. The resources, once created, are referred to as stacks.

The template languages are described in the Template Guide in the Heat developer documentation.

## Launch a stack

1.  Log in to the dashboard, choose a project, and click **Stacks** in the **Orchestration** category on the **Projects** tab.

2.  Click **Launch Stack**.

3.  In the **Select Template** dialog box, specify the following values.

| | |
|---|---|
| **Template Source** | Choose the source of the template from the list. |
| **Template URL/File/Data** | Depending on the source that you selected, enter the URL, browse to the file location, or directly include the template. |
| **Environment Source** | Choose the source of the environment from the list. The environment files contain additional settings for the stack. |
| **Environment URL/File/Data** | Depending on the source that you selected, enter the URL, browse to the file location, or directly include the environment. |

4.  Click **Next**.

5.  In the **Launch Stack** dialog box, specify the following values.

| | |
|---|---|
| **Stack Name** | Enter a name to identify the stack. |
| **Creation Timeout (minutes)** | Specify the number of minutes that can elapse before the launch of the stack times out. |
| **Rollback On Failure** | Select this check box if you want if you want the service to roll back changes if the stack fails to launch. |
| **Password for user "demo"** | Specify the password that the default user uses when the stack is created. |
| **DBUsername** | Specify the name of the database user. |
| **LinuxDistribution** | Specify the Linux distribution that is used in the stack. |
| **DBRootPassword** | Specify the root password for the database. |
| **KeyName** | Specify the name of the key pair to use to log in to the stack. |

| DBName | Specify the name of the database. |
|---|---|
| DBPassword | Specify the password for the database. |
| InstanceType | Specify the flavor for the instance. |

6.　Click **Launch** to create a stack.

7.　The **Stacks** tab shows the stack.

After the stack is created, click on the stack name to see the following details:

**Topology**　　The topology of the stack.

**Overview**　　The parameters and details of the stack.

**Resources**　　The resources used by the stack.

**Events**　　The events related to the stack.

# Manage a stack

1.　Log in to the dashboard, choose a project, and click **Stacks**.

2.　Select the stack that you want to update.

3.　Click **Change Stack Template**.

4.　In the **Select Template** dialog box, select the new template source or environment source.

5.　Click **Next**.

　　The **Update Stack Parameters** window appears.

6.　Enter new values for any parameters that you want to update.

7.　Click **Update**.

# Delete a stack

When you delete a stack, you cannot undo this action.

1.　Log in to the dashboard, choose a project, and click **Stacks**.

2.　Select the stack that you want to delete.

3.　Click **Delete Stack**.

4.　In the confirmation dialog box, click **Delete Stack** to confirm the deletion.

# Create and manage databases

The Database service provides scalable and reliable cloud provisioning functionality for both relational and non-relational database engines. Users can quickly and easily use database features without the burden of handling complex administrative tasks.

## Create a database instance

**Prerequisites.** Before you create a database instance, you need to configure a default datastore and make sure you have an appropriate flavor for the type of database instance you want.

1.  **Configure a default datastore.**

    Because the dashboard does not let you choose a specific datastore to use with an instance, you need to configure a default datastore. The dashboard then uses the default datastore to create the instance.

    a.  Add the following line to `/etc/trove/trove.conf`:

    ```
    default_datastore = DATASTORE_NAME
    ```

    Replace `DATASTORE_NAME` with the name that the administrative user set when issuing the **trove-manage** command to create the datastore. You can use the trove **datastore-list** command to display the datastores that are available in your environment.

    For example, if your MySQL datastore name is set to `mysql`, your entry would look like this:

    ```
    default_datastore = mysql
    ```

    b.  Restart Database services on the controller node:

    ```
    # service trove-api restart
    # service trove-taskmanager restart
    # service trove-conductor restart
    ```

2.  **Verify flavor.**

    Make sure an appropriate flavor exists for the type of database instance you want.

**Create database instance.**    Once you have configured a default datastore and verified that you have an appropriate flavor, you can create a database instance.

1.  Log in to the dashboard, choose a project, and click **Databases**.

2.  Click **Database Instances**. This lists the instances that already exist in your environment.

3.  Click **Launch Instance**.

4.  In the **Launch Database** dialog box, specify the following values.

| Details | |
|---|---|
| Database Name | Specify a name for the database instance. |
| Flavor | Select an appropriate flavor for the instance. |
| Volume Size | Select a volume size. Volume size is expressed in GB. |
| **Initialize Databases** | |
| Initial Database | Optionally provide a comma separated list of databases to create, for example:<br><br>`database1, database2, database3` |
| Initial Admin User | Create an initial admin user. This user will have access to all the databases you create. |
| Password | Specify a password associated with the initial admin user you just named. |
| Host | Optionally, allow the user to connect only from this host. If you do not specify a host, this user will be allowed to connect from anywhere. |

5.  Click the **Launch** button. The new database instance appears in the databases list.

# Backup and restore a database

You can use Database services to backup a database and store the backup artifact in the Object Storage module. Later on, if the original database is damaged, you can use the backup artifact to restore the database. The restore process creates a database instance.

This example shows you how to back up and restore a MySQL database.

## To backup and restore a database

1.  **Backup the database instance.**

    a.  Log in to the dashboard, choose a project, and click **Databases**.

    b.  Click **Database Instances**. This displays the existing instances in your system.

    c.  Click **Create Backup**.

    d.  In the Backup Database dialog box, specify the following values:

    | Name | Specify a name for the backup. |
    |---|---|
    | Database Instance | Select the instance you want to back up. |
    | Description | Specify an optional description. |

    e.  Click **Backup**. The new backup appears in the backup list.

2.  **Restore a database instance.**

    Now assume that your original database instance is damaged and you need to restore it. You do the restore by using your backup to create a new database instance.

    a.  Log in to the dashboard, choose a project, and click **Databases**.

    b.  Click **Database Backups**. This lists the available backups.

c.   Check the backup you want to use and click **Restore Backup**.

d.   In the **Launch Database** dialog box, specify the values you want to use for the new database instance. Click the **Restore From Database** tab and make sure that this new instance is based on the correct backup.

e.   Click **Launch**. The new instance appears in the database instances list.

# 2. OpenStack command-line clients

## Table of Contents

# Overview

Each OpenStack project provides a command-line client, which enables you to access the project API through easy-to-use commands. For example, the Compute service provides a nova command-line client.

You can run the commands from the command line, or include the commands within scripts to automate tasks. If you provide OpenStack credentials, such as your user name and password, you can run these commands on any computer.

Internally, each command uses cURL command-line tools, which embed API requests. Open-Stack APIs are RESTful APIs, and use the HTTP protocol. They include methods, URIs, media types, and response codes.

OpenStack APIs are open-source Python clients, and can run on Linux or Mac OS X systems. On some client commands, you can specify a **debug** parameter to show the underlying API request for the command. This is a good way to become familiar with the OpenStack API calls.

The following table lists the command-line client for each OpenStack service with its package name and description.

## Table 2.1. OpenStack services and clients

| Service | Client | Package | Description |
|---|---|---|---|
| Block Storage | **cinder** | python-cinderclient | Create and manage volumes. |

| Service | Client | Package | Description |
|---|---|---|---|
| Compute | **nova** | python-novaclient | Create and manage images, instances, and flavors. |
| Database Service | **trove** | python-troveclient | Create and manage databases. |
| Identity | **keystone** | python-keystoneclient | Create and manage users, tenants, roles, endpoints, and credentials. |
| Image Service | **glance** | python-glanceclient | Create and manage images. |
| Networking | **neutron** | python-neutronclient | Configure networks for guest servers. This client was previously called **quantum**. |
| Object Storage | **swift** | python-swiftclient | Gather statistics, list items, update metadata, and upload, download, and delete files stored by the Object Storage service. Gain access to an Object Storage installation for ad hoc processing. |
| Orchestration | **heat** | python-heatclient | Launch stacks from templates, view details of running stacks including events and resources, and update and delete stacks. |
| Telemetry | **ceilometer** | python-ceilometer-client | Create and collect measurements across OpenStack. |
| Data Processing | **sahara** | python-saharaclient | Creates and manages Hadoop clusters on OpenStack. |
| Common client | **openstack** | python-openstack-client | Common client for the OpenStack project. |

# Install the OpenStack command-line clients

Install the prerequisite software and the Python package for each OpenStack client.

## Install the prerequisite software

Most Linux distributions include packaged versions of the command-line clients that you can install directly, see the section called "Installing from packages" [35].

If you need to install the command-line packages source packages, the following table lists the software that you need to have to run the command-line clients, and provides installation instructions as needed.

### Table 2.2. Prerequisite software

| Prerequisite | Description |
|---|---|
| Python 2.7 or later | Currently, the clients do 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. |
| | The recommended way to install setuptools on Microsoft Windows is to follow the documentation provided on the setuptools website. Another option is to use the unofficial binary installer maintained by Christoph Gohlke (http://www.lfd.uci.edu/~gohlke/pythonlibs/#setuptools). |
| pip package | To install the clients on a Linux, Mac OS X, or Microsoft Windows system, use pip. It is easy to use, ensures that you get the latest version of the clients from the Python Package Index, and lets you update or remove the packages later on. |
| | Since the installation process compiles source files, this requires the related Python development package for your operating system and distribution. |
| | Install pip through the package manager for your system: |
| | **MacOS.** |
| | ```# easy_install pip``` |
| | **Microsoft Windows.**    Ensure that the `C:\Python27\Scripts` directory is defined in the `PATH` environment variable, and use the **easy_install** command from the setuptools package: |
| | ```C:\>easy_install pip``` |
| | Another option is to use the unofficial binary installer provided by Christoph Gohlke (http://www.lfd.uci.edu/~gohlke/pythonlibs/#pip). |
| | **Ubuntu and Debian.** |
| | ```# apt-get install python-devel python-pip``` |
| | Note that extra dependencies may be required, per operating system, depending on the package being installed, such as is the case with Tempest. |
| | **Red Hat Enterprise Linux, CentOS, or Fedora.**    A packaged version enables you to use yum to install the package: |
| | ```# yum install python-dev python-pip``` |

| Prerequisite | Description |
|---|---|
| | There are also packaged versions of the clients available in RDO that enable yum to install the clients as described in the section called "Installing from packages" [35].<br><br>**SUSE Linux Enterprise Linux 11.** A packaged version available in the Open Build Service enables you to use or zypper to install the package. First, add the Open Build Service repository:<br><br>`# zypper addrepo -f obs://Cloud:OpenStack:Icehouse/SLE_11_SP3 Icehouse`<br><br>Then install pip and use it to manage client installation:<br><br>`# zypper install python-devel python-pip`<br><br>There are also packaged versions of the clients available that enable zypper to install the clients as described in the section called "Installing from packages" [35].<br><br>**openSUSE.** You can install pip and use it to manage client installation:<br><br>`# zypper install python-devel python-pip`<br><br>There are also packaged versions of the clients available that enable zypper to install the clients as described in the section called "Installing from packages" [35] |

# Install the clients

When following the instructions in this section, replace *PROJECT* with the lowercase name of the client to install, such as **nova**. Repeat for each client. The following values are valid:

- `ceilometer` - Telemetry API

- `cinder` - Block Storage API and extensions

- `glance` - Image Service API

- `heat` - Orchestration API

- `keystone` - Identity service API and extensions

- `neutron` - Networking API

- `nova` - Compute API and extensions

- `sahara` - Database Processing API

- `swift` - Object Storage API

- `trove` - Database Service API

- `openstack` - Common OpenStack client supporting multiple services

The following example shows the command for installing the nova client with **pip**.

```
# pip install python-novaclient
```

## Installing with pip

Use pip to install the OpenStack clients on a Linux, Mac OS X, or Microsoft Windows system. It is easy to use and ensures that you get the latest version of the client from the Python Package Index. Also, pip enables you to update or remove a package.

Install each client separately by using the following command:

- For Mac OS X or Linux:

```
# pip install python-PROJECTclient
```

- For Microsoft Windows:

```
C:\>pip install python-PROJECTclient
```

## Installing from packages

RDO, openSUSE and SUSE Linux Enterprise have client packages that can be installed without `pip`.

On Red Hat Enterprise Linux, CentOS, or Fedora, use **yum** to install the clients from the packaged versions available in RDO:

```
# yum install python-PROJECTclient
```

For openSUSE, use zypper to install the clients from the distribution packages Service:

```
# zypper install python-PROJECT
```

For SUSE Linux Enterprise Server, use zypper to install the clients from the distribution packages in the Open Build Service. First, add the Open Build Service repository:

```
# zypper addrepo -f obs://Cloud:OpenStack:Icehouse/SLE_11_SP3 Icehouse
```

Then you can install the packages:

```
# zypper install python-PROJECT
```

# Upgrade or remove clients

To upgrade a client, add the `--upgrade` option to the **pip install** command:

```
# pip install --upgrade python-PROJECTclient
```

To remove the a client, run the **pip uninstall** command:

```
# pip uninstall python-PROJECTclient
```

# What's next

Before you can run client commands, you must create and source the `PROJECT-openrc.sh` file to set environment variables. See the section called "Set environment variables using the OpenStack RC file" [37].

# Discover the version number for a client

Run the following command to discover the version number for a client:

```
$ PROJECT --version
```

For example, to see the version number for the **nova** client, run the following command:

```
$ nova --version
```

The version number (2.15.0 in the example) is returned.

```
2.15.0
```

# Set environment variables using the OpenStack RC file

To set the required environment variables for the OpenStack command-line clients, you must create an environment file called an OpenStack rc file, or `openrc.sh` file. If your OpenStack installation provides it, you can download the file from the OpenStack dashboard as an administrative user or any other user. This project-specific environment file contains the credentials that all OpenStack services use.

When you source the file, environment variables are set for your current shell. The variables enable the OpenStack client commands to communicate with the OpenStack services that run in the cloud.

> **Note**
>
> Defining environment variables using an environment file is not a common practice on Microsoft Windows. Environment variables are usually defined in the **Advanced** tab of the System Properties dialog box.

## Download and source the OpenStack RC file

1.  Log in to the OpenStack dashboard, choose the project for which you want to download the OpenStack RC file, and click **Access & Security**.

2.  On the API Access tab, click **Download OpenStack RC File** and save the file. The filename will be of the form *PROJECT*-`openrc.sh` where *PROJECT* is the name of the project for which you downloaded the file.

3.  Copy the *PROJECT*-`openrc.sh` file to the computer from which you want to run OpenStack commands.

    For example, copy the file to the computer from which you want to upload an image with a **glance** client command.

4.  On any shell from which you want to run OpenStack commands, source the *PROJECT*-`openrc.sh` file for the respective project.

    In the following example, the `demo-openrc.sh` file is sourced for the demo project:

    ```
    $ source demo-openrc.sh
    ```

5.  When you are prompted for an OpenStack password, enter the password for the user who downloaded the *PROJECT*-`openrc.sh` file.

## Create and source the OpenStack RC file

Alternatively, you can create the *PROJECT*-`openrc.sh` file from scratch, if for some reason you cannot download the file from the dashboard.

1.  In a text editor, create a file named *PROJECT*-`openrc.sh` file and add the following authentication information:

```
export OS_USERNAME=username
export OS_PASSWORD=password
export OS_TENANT_NAME=projectName
export OS_AUTH_URL=https://identityHost:portNumber/v2.0
# The following lines can be omitted
export OS_TENANT_ID=tenantIDString
export OS_REGION_NAME=regionName
```

2. On any shell from which you want to run OpenStack commands, source the `PROJECT-openrc.sh` file for the respective project. In this example, you source the `admin-openrc.sh` file for the `admin` project:

```
$ source admin-openrc.sh
```

> **Note**
>
> You are not prompted for the password with this method. The password lives in clear text format in the `PROJECT-openrc.sh` file. Restrict the permissions on this file to avoid security problems. You can also remove the `OS_PASSWORD` variable from the file, and use the `--password` parameter with OpenStack client commands instead.

# Override environment variable values

When you run OpenStack client commands, you can override some environment variable settings by using the options that are listed at the end of the **help** output of the various client commands. For example, you can override the `OS_PASSWORD` setting in the `PROJECT-openrc.sh` file by specifying a password on a **keystone** command, as follows:

```
$ keystone --os-password PASSWORD service-list
```

Where `PASSWORD` is your password.

# Keyring support

Keyring is a password management system available in OpenStack. You can install it using the following command:

```
$ pip install keyring
```

> **Note**
>
> Keyring is used only if `--os-use-keyring` is specified or if the environment variable `OS_USE_KEYRING=true` is defined.

A user specifies their username and password credentials to interact with OpenStack, using any client command. These credentials can be specified using various mechanisms, namely, the environment variable, or command line argument. It is not safe to specify the password using either of these methods.

For example, when you specify your password using the command-line client with the `--os-password` argument, anyone with access to your computer can view it in plain text with the `ps` field.

To avoid storing the password in plain text, you can prompt for the OpenStack password interactively. Then, the keyring can store the password and the user can safely retrieve it from their keyring. The encrypted password is stored in the `~/.open-stack-keyring.cfg` file.

# Manage images

The cloud operator assigns roles to users. Roles determine who can upload and manage images. The operator might restrict image upload and management to only cloud administrators or operators.

You can upload images through the **glance** client or the Image Service API. You can also use the **nova** client to list images, set and delete image metadata, delete images, and take a snapshot of a running instance to create an image. After you upload an image, you cannot change it.

For details about image creation, see the *Virtual Machine Image Guide*.

# List or get details for images (glance)

To get a list of images and to then get further details about a single image, use **glance image-list** and **glance image-show**.

```
$ glance image-list
+-------------------------------------+---------------------------------
+-------------+------------------+----------+--------+
| ID                                  | Name                           |
 Disk Format | Container Format | Size     | Status |
+-------------------------------------+---------------------------------
+-------------+------------------+----------+--------+
| 397e713c-b95b-4186-ad46-6126863ea0a9 | cirros-0.3.2-x86_64-uec         | ami
        | ami              | 25165824 | active |
| df430cc2-3406-4061-b635-a51c16e488ac | cirros-0.3.2-x86_64-uec-kernel  | aki
        | aki              | 4955792  | active |
| 3cf852bd-2332-48f4-9ae4-7d926d50945e | cirros-0.3.2-x86_64-uec-ramdisk | ari
        | ari              | 3714968  | active |
| 7e5142af-1253-4634-bcc6-89482c5f2e8a | myCirrosImage                   | ami
        | ami              | 14221312 | active |
+-------------------------------------+---------------------------------
+-------------+------------------+----------+--------+
```

```
$ glance image-show myCirrosImage

+-------------------------------------+---------------------------------------+
| Property                            | Value                                 |
+-------------------------------------+---------------------------------------+
| Property 'base_image_ref'           | 397e713c-b95b-4186-ad46-6126863ea0a9  |
| Property 'image_location'           | snapshot                              |
| Property 'image_state'              | available                             |
| Property 'image_type'               | snapshot                              |
| Property 'instance_type_ephemeral_gb' | 0                                   |
| Property 'instance_type_flavorid'   | 2                                     |
| Property 'instance_type_id'         | 5                                     |
| Property 'instance_type_memory_mb'  | 2048                                  |
| Property 'instance_type_name'       | m1.small                              |
| Property 'instance_type_root_gb'    | 20                                    |
| Property 'instance_type_rxtx_factor' | 1                                    |
| Property 'instance_type_swap'       | 0                                     |
| Property 'instance_type_vcpu_weight' | None                                 |
| Property 'instance_type_vcpus'      | 1                                     |
| Property 'instance_uuid'            | 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5  |
| Property 'kernel_id'                | df430cc2-3406-4061-b635-a51c16e488ac  |
| Property 'owner_id'                 | 66265572db174a7aa66eba661f58eb9e      |
| Property 'ramdisk_id'               | 3cf852bd-2332-48f4-9ae4-7d926d50945e  |
| Property 'user_id'                  | 376744b5910b4b4da7d8e6cb483b06a8      |
| checksum                            | 8e4838effa1969ad591655d6485c7ba8      |
| container_format                    | ami                                   |
| created_at                          | 2013-07-22T19:45:58                   |
| deleted                             | False                                 |
| disk_format                         | ami                                   |
| id                                  | 7e5142af-1253-4634-bcc6-89482c5f2e8a  |
| is_public                           | False                                 |
| min_disk                            | 0                                     |
| min_ram                             | 0                                     |
| name                                | myCirrosImage                         |
| owner                               | 66265572db174a7aa66eba661f58eb9e      |
| protected                           | False                                 |
| size                                | 14221312                              |
| status                              | active                                |
| updated_at                          | 2013-07-22T19:46:42                   |
```

```
+----------------------------------+-----------------------------------+
```

When viewing a list of images, you can also use **grep** to filter the list, as follows:

```
$ glance image-list | grep 'cirros'
| 397e713c-b95b-4186-ad46-6126863ea0a9 | cirros-0.3.2-x86_64-uec         | ami
         | ami         | 25165824 | active |
| df430cc2-3406-4061-b635-a51c16e488ac | cirros-0.3.2-x86_64-uec-kernel  | aki
         | aki         | 4955792  | active |
| 3cf852bd-2332-48f4-9ae4-7d926d50945e | cirros-0.3.2-x86_64-uec-ramdisk | ari
         | ari         | 3714968  | active |
```

> **Note**
>
> To store location metadata for images, which enables direct file access for a client, update the `/etc/glance/glance.conf` file with the following statements:
>
> - `show_multiple_locations = True`
>
> - `filesystem_store_metadata_file = ` *filePath*, where *filePath* points to a JSON file that defines the mount point for OpenStack images on your system and a unique ID. For example:
>
>   ```
>   [{
>       "id": "2d9bb53f-70ea-4066-a68b-67960eaae673",
>       "mountpoint": "/var/lib/glance/images/"
>   }]
>   ```
>
> After you restart the Image Service, you can use the following syntax to view the image's location information:
>
> ```
> $ glance --os-image-api-version 2 image-show imageID
> ```
>
> For example, using the image ID shown above, you would issue the command as follows:
>
> ```
> $ glance --os-image-api-version 2 image-show 2d9bb53f-70ea-4066-
> a68b-67960eaae673
> ```

# Create or update an image (glance)

To create an image, use **glance image-create**:

```
$ glance image-create imageName
```

To update an image by name or ID, use **glance image-update**:

```
$ glance image-update imageName
```

The following table lists the optional arguments that you can use with the **create** and **update** commands to modify image properties. For more information, refer to Image Service chapter in the OpenStack Command-Line Interface Reference.

| | |
|---|---|
| `--name NAME` | The name of the image. |
| `--disk-format DISK_FORMAT` | The disk format of the image. Acceptable formats are ami, ari, aki, vhd, vmdk, raw, qcow2, vdi, and iso. |

| `--container-format CONTAINER_FORMAT` | The container format of the image. Acceptable formats are ami, ari, aki, bare, and ovf. |
|---|---|
| `--owner TENANT_ID` | The tenant who should own the image. |
| `--size SIZE` | The size of image data, in bytes. |
| `--min-disk DISK_GB` | The minimum size of the disk needed to boot the image, in gigabytes. |
| `--min-ram DISK_RAM` | The minimum amount of RAM needed to boot the image, in megabytes. |
| `--location IMAGE_URL` | The URL where the data for this image resides. For example, if the image data is stored in swift, you could specify `swift://account:key@example.com/container/obj`. |
| `--file FILE` | Local file that contains the disk image to be uploaded during the update. Alternatively, you can pass images to the client through stdin. |
| `--checksum CHECKSUM` | Hash of image data to use for verification. |
| `--copy-from IMAGE_URL` | Similar to `--location` in usage, but indicates that the image server should immediately copy the data and store it in its configured image store. |
| `--is-public [True\|False]` | Makes an image accessible for all the tenants (admin-only by default). |
| `--is-protected [True\|False]` | Prevents an image from being deleted. |
| `--property KEY=VALUE` | Arbitrary property to associate with image. This option can be used multiple times. |
| `--purge-props` | Deletes all image properties that are not explicitly set in the update request. Otherwise, those properties not referenced are preserved. |
| `--human-readable` | Prints the image size in a human-friendly format. |

The following example shows the command that you would use to upload a CentOS 6.3 image in qcow2 format and configure it for public access:

```
$ glance image-create --name centos63-image --disk-format qcow2 \
  --container-format bare --is-public True --file ./centos63.qcow2
```

The following example shows how to update an existing image with a properties that describe the disk bus, the CD-ROM bus, and the VIF model:

```
$ glance image-update \
    --property hw_disk_bus=scsi \
    --property hw_cdrom_bus=ide \
    --property hw_vif_model=e1000 \
    f16-x86_64-openstack-sda
```

Currently the libvirt virtualization tool determines the disk, CD-ROM, and VIF device models based on the configured hypervisor type (`libvirt_type` in `/etc/nova/nova.conf`). For the sake of optimal performance, libvirt defaults to using virtio for both disk and VIF (NIC) models. The disadvantage of this approach is that it is not possible to run operating systems that lack virtio drivers, for example, BSD, Solaris, and older versions of Linux and Windows.

If you specify a disk or CD-ROM bus model that is not supported, see Table 2.3, "Disk and CD-ROM bus model values" [43]. If you specify a VIF model that is not supported, the instance fails to launch. See Table 2.4, "VIF model values" [43].

The valid model values depend on the `libvirt_type` setting, as shown in the following tables.

### Table 2.3. Disk and CD-ROM bus model values

| libvirt_type setting | Supported model values |
|---|---|
| qemu or kvm | • ide<br><br>• scsi<br><br>• virtio |
| xen | • ide<br><br>• xen |

### Table 2.4. VIF model values

| libvirt_type setting | Supported model values |
|---|---|
| qemu or kvm | • e1000<br><br>• ne2k_pci<br><br>• pcnet<br><br>• rtl8139<br><br>• virtio |
| xen | • e1000<br><br>• netfront<br><br>• ne2k_pci<br><br>• pcnet<br><br>• rtl8139 |
| vmware | • VirtualE1000<br><br>• VirtualPCNet32<br><br>• VirtualVmxnet |

# Create an image (nova)

You can use the **nova** client to take a snapshot of a running instance to create an image.

To minimize the potential for data loss and ensure that you create an accurate image, you should shut down the instance before you take a snapshot.

1.  Write any buffered data to disk.

    For more information, see Taking Snapshots in the *OpenStack Operations Guide*.

2.  List instances to get the server name:

    ```
    $ nova list
    +--------------------------------------+----------------+--------+------------+-------------+------------------+
    | ID                                   | Name           | Status | Task State | Power State | Networks         |
    +--------------------------------------+----------------+--------+------------+-------------+------------------+
    | 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5 | myCirrosServer | ACTIVE | None       | Running     | private=10.0.0.3 |
    +--------------------------------------+----------------+--------+------------+-------------+------------------+
    ```

    In this example, the instance is named `myCirrosServer`.

3.  Use this instance to create a snapshot:

```
$ nova image-create myCirrosServer myCirrosImage
```

The command creates a snapshot and automatically uploads the image to your reposi-
tory.

**Note**

For snapshots that you create from an instance that was booted from a
volume:

- The snapshot is based on the volume that is attached to the instance
  through the Block Storage service.

- No data is uploaded to the Image Service.

- You can find information about the snapshot in the properties of the im-
  age.

4.  Get details for your image to check its status:

```
$ nova image-show myCirrosImage
+----------------------------------+--------------------------------------+
| Property                         | Value                                |
+----------------------------------+--------------------------------------+
| metadata owner_id                | 66265572db174a7aa66eba661f58eb9e     |
| minDisk                          | 0                                    |
| metadata instance_type_name      | m1.small                             |
| metadata instance_type_id        | 5                                    |
| metadata instance_type_memory_mb | 2048                                 |
| id                               | 7e5142af-1253-4634-bcc6-89482c5f2e8a |
| metadata instance_type_root_gb   | 20                                   |
| metadata instance_type_rxtx_factor | 1                                  |
| metadata ramdisk_id              | 3cf852bd-2332-48f4-9ae4-7d926d50945e |
| metadata image_state             | available                            |
| metadata image_location          | snapshot                             |
| minRam                           | 0                                    |
| metadata instance_type_vcpus     | 1                                    |
| status                           | ACTIVE                               |
| updated                          | 2013-07-22T19:46:42Z                 |
| metadata instance_type_swap      | 0                                    |
| metadata instance_type_vcpu_weight | None                               |
| metadata base_image_ref          | 397e713c-b95b-4186-ad46-6126863ea0a9 |
| progress                         | 100                                  |
| metadata instance_type_flavorid  | 2                                    |
| OS-EXT-IMG-SIZE:size             | 14221312                             |
| metadata image_type              | snapshot                             |
| metadata user_id                 | 376744b5910b4b4da7d8e6cb483b06a8     |
| name                             | myCirrosImage                        |
| created                          | 2013-07-22T19:45:58Z                 |
| metadata instance_uuid           | 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5 |
| server                           | 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5 |
| metadata kernel_id               | df430cc2-3406-4061-b635-a51c16e488ac |
| metadata instance_type_ephemeral_gb | 0                                 |
+----------------------------------+--------------------------------------+
```

The image status changes from SAVING to ACTIVE. Only the tenant who creates the
image has access to it.

To launch an instance from your image, include the image ID and flavor ID, as in the follow-
ing example:

```
$ nova boot newServer --image 7e5142af-1253-4634-bcc6-89482c5f2e8a \
   --flavor 3
+----------------------------------+--------------------------------------+
| Property                         | Value                                |
+----------------------------------+--------------------------------------+
| OS-EXT-STS:task_state            | scheduling                           |
| image                            | myCirrosImage                        |
| OS-EXT-STS:vm_state              | building                             |
| OS-EXT-SRV-ATTR:instance_name    | instance-00000007                    |
| flavor                           | m1.medium                            |
| id                               | d7efd3e4-d375-46d1-9d57-372b6e4bdb7f |
| security_groups                  | [{u'name': u'default'}]              |
| user_id                          | 376744b5910b4b4da7d8e6cb483b06a8     |
| OS-DCF:diskConfig                | MANUAL                               |
| accessIPv4                       |                                      |
| accessIPv6                       |                                      |
```

```
| progress                             | 0                                |
| OS-EXT-STS:power_state               | 0                                |
| OS-EXT-AZ:availability_zone          | nova                             |
| config_drive                         |                                  |
| status                               | BUILD                            |
| updated                              | 2013-07-22T19:58:33Z             |
| hostId                               |                                  |
| OS-EXT-SRV-ATTR:host                 | None                             |
| key_name                             | None                             |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | None                             |
| name                                 | newServer                        |
| adminPass                            | jis88nN46RGP                     |
| tenant_id                            | 66265572db174a7aa66eba661f58eb9e |
| created                              | 2013-07-22T19:58:33Z             |
| metadata                             | {}                               |
+--------------------------------------+----------------------------------+
```

# Troubleshoot image creation

If you encounter problems in creating an image in Image Service or Compute, the following information may help you troubleshoot the creation process.

- Ensure that the version of qemu you are using is version 0.14 or later. Earlier versions of qemu result in an `unknown option -s` error message in the `nova-compute.log` file.

- Examine the `/var/log/nova-api.log` and `/var/log/nova-compute.log` log files for error messages.

# Configure access and security for instances

When you launch a virtual machine, you can inject a *key pair*, which provides SSH access to your instance. For this to work, the image must contain the `cloud-init` package.

You create at least one key pair for each project. You can use the key pair for multiple instances that belong to that project. If you generate a key pair with an external tool, you can import it into OpenStack.

If an image uses a static root password or a static key set—neither is recommended—you must not provide a key pair when you launch the instance.

A *security group* is a named collection of network access rules that you use to limit the types of traffic that have access to instances. When you launch an instance, you can assign one or more security groups to it. If you do not create security groups, new instances are automatically assigned to the default security group, unless you explicitly specify a different security group.

The associated *rules* in each security group control the traffic to instances in the group. Any incoming traffic that is not matched by a rule is denied access by default. You can add rules to or remove rules from a security group, and you can modify rules for the default and any other security group.

You can modify the rules in a security group to allow access to instances through different ports and protocols. For example, you can modify rules to allow access to instances through SSH, to ping instances, or to allow UDP traffic; for example, for a DNS server running on an instance. You specify the following parameters for rules:

• **Source of traffic**. Enable traffic to instances from either IP addresses inside the cloud from other group members or from all IP addresses.

• **Protocol**. Choose TCP for SSH, ICMP for pings, or UDP.

• **Destination port on virtual machine**. Define a port range. To open a single port only, enter the same value twice. ICMP does not support ports; instead, you enter values to define the codes and types of ICMP traffic to be allowed.

Rules are automatically enforced as soon as you create or modify them.

> ### Note
>
> Instances that use the default security group cannot, by default, be accessed from any IP address outside of the cloud. If you want those IP addresses to access the instances, you must modify the rules for the default security group.
>
> You can also assign a floating IP address to a running instance to make it accessible from outside the cloud. See the section called "Manage IP addresses" [61].

# Add a key pair

You can generate a key pair or upload an existing public key.

1. To generate a key pair, run the following command:

   ```
   $ nova keypair-add KEY_NAME > MY_KEY.pem
   ```

   The command generates a key pair with the name that you specify for KEY_NAME, writes the private key to the .pem file that you specify, and registers the public key at the Nova database.

2. To set the permissions of the .pem file so that only you can read and write to it, run the following command:

   ```
   $ chmod 600 MY_KEY.pem
   ```

# Import a key pair

1. If you have already generated a key pair and the public key is located at ~/.ssh/id_rsa.pub, run the following command to upload the public key:

   ```
   $ nova keypair-add --pub_key ~/.ssh/id_rsa.pub KEY_NAME
   ```

   The command registers the public key at the Nova database and names the key pair the name that you specify for KEY_NAME.

2. To ensure that the key pair has been successfully imported, list key pairs as follows:

   ```
   $ nova keypair-list
   ```

# Create and manage security groups

1. To list the security groups for the current project, including descriptions, enter the following command:

   ```
   $ nova secgroup-list
   ```

2. To create a security group with a specified name and description, enter the following command:

   ```
   $ nova secgroup-create SECURITY_GROUP_NAME GROUP_DESCRIPTION
   ```

3. To delete a specified group, enter the following command:

   ```
   $ nova secgroup-delete SECURITY_GROUP_NAME
   ```

   ### Note

   You cannot delete the default security group for a project. Also, you cannot delete a security group that is assigned to a running instance.

# Create and manage security group rules

Modify security group rules with the **nova secgroup-*-rule** commands. Before you begin, source the OpenStack RC file. For details, see the section called "Set environment variables using the OpenStack RC file" [37].

1.  To list the rules for a security group, run the following command:

    ```
    $ nova secgroup-list-rules SECURITY_GROUP_NAME
    ```

2.  To allow SSH access to the instances, choose one of the following options:

    *   Allow access from all IP addresses, specified as IP subnet 0.0.0.0/0 in CIDR notation:

        ```
        $ nova secgroup-add-rule SECURITY_GROUP_NAME tcp 22 22 0.0.0.0/0
        ```

    *   Allow access only from IP addresses from other security groups (source groups) to access the specified port:

        ```
        $ nova secgroup-add-group-rule --ip_proto tcp --from_port 22 \
              --to_port 22 SECURITY_GROUP_NAME SOURCE_GROUP_NAME
        ```

3.  To allow pinging of the instances, choose one of the following options:

    *   Allow pinging from all IP addresses, specified as IP subnet 0.0.0.0/0 in CIDR notation:

        ```
        $ nova secgroup-add-rule SECURITY_GROUP_NAME icmp -1 -1 0.0.0.0/0
        ```

        This allows access to all codes and all types of ICMP traffic.

    *   Allow only members of other security groups (source groups) to ping instances:

        ```
        $ nova secgroup-add-group-rule --ip_proto icmp --from_port -1 \
              --to_port -1 SECURITY_GROUP_NAME SOURCE_GROUP_NAME
        ```

4.  To allow access through a UDP port, such as allowing access to a DNS server that runs on a VM, choose one of the following options:

    *   Allow UDP access from IP addresses, specified as IP subnet 0.0.0.0/0 in CIDR notation:

        ```
        $ nova secgroup-add-rule SECURITY_GROUP_NAME udp 53 53 0.0.0.0/0
        ```

    *   Allow only IP addresses from other security groups (source groups) to access the specified port:

        ```
        $ nova secgroup-add-group-rule --ip_proto udp --from_port 53 \
              --to_port 53 SECURITY_GROUP_NAME SOURCE_GROUP_NAME
        ```

## Delete a security group rule

To delete a security group rule, specify the same arguments that you used to create the rule.

For example, to delete the security group rule that permits SSH access from all IP addresses, run the following command.

```
$ nova secgroup-delete-rule SECURITY_GROUP_NAME tcp 22 22 0.0.0.0/0
```

# Launch instances

Instances are virtual machines that run inside the cloud.

Before you can launch an instance, gather the following parameters:

- The **instance source** can be an image, snapshot, or block storage volume that contains an image or snapshot.

- A **name** for your instance.

- The **flavor** for your instance, which defines the compute, memory, and storage capacity of nova computing instances. A flavor is an available hardware configuration for a server. It defines the size of a virtual server that can be launched.

- Any **user data** files. A user data file is a special key in the metadata service that holds a file that cloud-aware applications in the guest instance can access. For example, one application that uses user data is the cloud-init system, which is an open-source package from Ubuntu that is available on various Linux distributions and that handles early initialization of a cloud instance.

- Access and security credentials, which include one or both of the following credentials:

  - A **key pair** for your instance, which are SSH credentials that are injected into images when they are launched. For the key pair to be successfully injected, the image must contain the `cloud-init` package. Create at least one key pair for each project. If you already have generated a key pair with an external tool, you can import it into OpenStack. You can use the key pair for multiple instances that belong to that project.

  - A **security group** that defines which incoming network traffic is forwarded to instances. Security groups hold a set of firewall policies, known as *security group rules*.

- If needed, you can assign a **floating (public) IP address** to a running instance.

- You can also attach a block storage device, or **volume**, for persistent storage.

> ### Note
>
> Instances that use the default security group cannot, by default, be accessed from any IP address outside of the cloud. If you want those IP addresses to access the instances, you must modify the rules for the default security group.
>
> You can also assign a floating IP address to a running instance to make it accessible from outside the cloud. See the section called "Manage IP addresses" [61].

After you gather the parameters that you need to launch an instance, you can launch it from an image or a volume. You can launch an instance directly from one of the available OpenStack images or from an image that you have copied to a persistent volume. The OpenStack Image Service provides a pool of images that are accessible to members of different projects.

# Gather parameters to launch an instance

Before you begin, source the OpenStack RC file.

1. List the available flavors:

```
$ nova flavor-list
```

Note the ID of the flavor that you want to use for your instance:

```
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor | Is_Public |
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
| 1  | m1.tiny   | 512       | 0    | 0         |      | 1     | 1.0         | True      |
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0         | True      |
| 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0         | True      |
| 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0         | True      |
| 5  | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0         | True      |
+----+-----------+-----------+------+-----------+------+-------+-------------+-----------+
```

2. List the available images:

```
$ nova image-list
```

Note the ID of the image from which you want to boot your instance:

```
+--------------------------------------+------------------------------+--------+--------+
| ID                                   | Name                         | Status | Server |
+--------------------------------------+------------------------------+--------+--------+
| 397e713c-b95b-4186-ad46-6126863ea0a9 | cirros-0.3.2-x86_64-uec      | ACTIVE |        |
| df430cc2-3406-4061-b635-a51c16e488ac | cirros-0.3.2-x86_64-uec-kernel | ACTIVE |        |
| 3cf852bd-2332-48f4-9ae4-7d926d50945e | cirros-0.3.2-x86_64-uec-ramdisk | ACTIVE |        |
+--------------------------------------+------------------------------+--------+--------+
```

You can also filter the image list by using **grep** to find a specific image, as follows:

```
$ nova image-list | grep 'kernel'
```

```
| df430cc2-3406-4061-b635-a51c16e488ac | cirros-0.3.2-x86_64-uec-kernel | ACTIVE |        |
```

3. List the available security groups:

### Note

If you are an admin user, specify the `--all-tenants` parameter to list groups for all tenants.

```
$ nova secgroup-list --all-tenants
```

Note the ID of the security group that you want to use for your instance:

```
+----+---------+-------------+----------------------------------+
| Id | Name    | Description | Tenant_ID                        |
+----+---------+-------------+----------------------------------+
| 2  | default | default     | 66265572db174a7aa66eba661f58eb9e |
| 1  | default | default     | b70d90d65e464582b6b2161cf3603ced |
+----+---------+-------------+----------------------------------+
```

If you have not created any security groups, you can assign the instance to only the default security group.

You can view rules for a specified security group:

```
$ nova secgroup-list-rules default
```

4. List the available key pairs and note the name of the key pair that you use for SSH access.

```
$ nova keypair-list
```

```
$ nova keypair-list
```

# Launch an instance from an image

1. After you gather required parameters, run the following command to launch an in-stance. Specify the server name, flavor ID, and image ID.

   Optionally, you can provide a key name for access control and a security group for se-curity. You can also include metadata key and value pairs. For example, you can add a description for your server by providing the `--meta description="My Server"` parameter.

   You can pass user data in a local file at instance launch by using the `--user-data USER-DATA-FILE` parameter.

   ```
   $ nova boot --flavor FLAVOR_ID --image IMAGE_ID --key-name KEY_NAME \
     --user-data USER_DATA_FILE --security-groups SEC_GROUP --meta KEY=VALUE
   \
     INSTANCE_NAME
   ```

   The following command launches the `MyCirrosServer` instance with the `m1.small` flavor (ID of `1`), `cirros-0.3.2-x86_64-uec` image (ID of `397e713c-b95b-4186-ad46-6126863ea0a9`), `default` security group, `KeyPair01` key, and a user data file called `cloudinit.file`:

   ```
   $ nova boot --flavor 1 --image 397e713c-b95b-4186-ad46-6126863ea0a9 \
     --security-groups default --key-name KeyPair01 --user-data cloudinit.
   file \
     myCirrosServer
   ```

   Depending on the parameters that you provide, the command returns a list of server properties.

   A status of `BUILD` indicates that the instance has started, but is not yet online.

   A status of `ACTIVE` indicates that the instance is active.

   ```
   +-----------------------------------+--------------------------------------+
   | Property                          | Value                                |
   +-----------------------------------+--------------------------------------+
   | OS-EXT-STS:task_state             | scheduling                           |
   | image                             | cirros-0.3.2-x86_64-uec              |
   | OS-EXT-STS:vm_state               | building                             |
   | OS-EXT-SRV-ATTR:instance_name     | instance-00000002                    |
   | flavor                            | m1.small                             |
   | id                                | b3cdc6c0-85a7-4904-ae85-71918f734048 |
   | security_groups                   | [{u'name': u'default'}]              |
   | user_id                           | 376744b5910b4b4da7d8e6cb483b06a8     |
   | OS-DCF:diskConfig                 | MANUAL                               |
   | accessIPv4                        |                                      |
   | accessIPv6                        |                                      |
   | progress                          | 0                                    |
   | OS-EXT-STS:power_state            | 0                                    |
   | OS-EXT-AZ:availability_zone       | nova                                 |
   | config_drive                      |                                      |
   | status                            | BUILD                                |
   | updated                           | 2013-07-16T16:25:34Z                 |
   | hostId                            |                                      |
   | OS-EXT-SRV-ATTR:host              | None                                 |
   | key_name                          | None                                 |
   | OS-EXT-SRV-ATTR:hypervisor_hostname | None                               |
   | name                              | myCirrosServer                       |
   | adminPass                         | tVs5pL8HcPGw                         |
   | tenant_id                         | 66265572db174a7aa66eba661f58eb9e     |
   | created                           | 2013-07-16T16:25:34Z                 |
   | metadata                          | {u'KEY': u'VALUE'}                   |
   +-----------------------------------+--------------------------------------+
   ```

   Copy the server ID value from the `id` field in the output. You use this ID to get details for or delete your server.

Copy the administrative password value from the `adminPass` field. You use this value to log in to your server.

> **Note**
>
> You can also place arbitrary local files into the instance file system at creation time by using the `--file <dst-path=src-path>` option. You can store up to five files. For example, if you have a special authorized keys file named `special_authorized_keysfile` that you want to put on the instance rather than using the regular SSH key injection, you can use the `--file` option as shown in the following example:
>
> ```
> $ nova boot --image ubuntu-cloudimage --flavor 1 vm-name \
>   --file /root/.ssh/authorized_keys=special_authorized_keysfile
> ```

2. Check if the instance is online:

```
$ nova list
```

The list shows the ID, name, status, and private (and if assigned, public) IP addresses for all instances in the project to which you belong:

```
+--------------------------------------+---------------------+--------+------------+-------------+------------------+
| ID                                   | Name                | Status | Task State | Power State | Networks         |
+--------------------------------------+---------------------+--------+------------+-------------+------------------+
| 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5 | myCirrosServer      | ACTIVE | None       | Running     | private=10.0.0.3 |
| 8a99547e-7385-4ad1-ae50-4ecfaaad5f42 | myInstanceFromVolume| ACTIVE | None       | Running     | private=10.0.0.4 |
+--------------------------------------+---------------------+--------+------------+-------------+------------------+
```

If the status for the instance is ACTIVE, the instance is online.

To view the available options for the **nova list** command, run the following command:

```
$ nova help list
```

> **Note**
>
> If you did not provide a key pair, security groups, or rules, you can access the instance only from inside the cloud through VNC. Even pinging the instance is not possible.

# Launch an instance from a volume

You can boot instances from a volume instead of an image.

To complete these tasks, use these parameters on the nova **boot** command:

| Task | nova boot parameter | See |
|------|---------------------|-----|
| Boot an instance from an image and attach a non-bootable volume. | `--block-device` | the section called "Boot instance from image and attach non-bootable volume" [55] |
| Create a volume from an image and boot an instance from that volume. | `--block-device` | the section called "Create volume from image and boot instance" [57] |
| Boot from an existing source image, volume, or snapshot. | `--block-device` | the section called "Create volume from image and boot instance" [57] |
| Attach a swap disk to an instance. | `--swap` | the section called "Attach swap or ephemeral disk to an instance" [60] |
| Attach an ephemeral disk to an instance. | `--ephemeral` | the section called "Attach swap or ephemeral disk to an instance" [60] |

### Note

To attach a volume to a running instance, see the section called "Attach a volume to an instance" [110].

## Boot instance from image and attach non-bootable volume

Create a non-bootable volume and attach that volume to an instance that you boot from an image.

To create a non-bootable volume, do not create it from an image. The volume must be entirely empty with no partition table and no file system.

1.  Create a non-bootable volume:

```
$ cinder create --display-name my-volume 8
+-------------------------------+--------------------------------------+
|            Property           |                Value                 |
+-------------------------------+--------------------------------------+
|          attachments          |                  []                  |
|        availability_zone      |                 nova                 |
|            bootable           |                false                 |
|           created_at          |      2014-05-09T16:33:11.000000       |
|          description          |                 None                 |
|           encrypted           |                False                 |
|              id               | d620d971-b160-4c4e-8652-2513d74e2080 |
|            metadata           |                  {}                  |
|              name             |              my-volume               |
|     os-vol-host-attr:host     |                 None                 |
|  os-vol-mig-status-attr:migstat |                None                |
|  os-vol-mig-status-attr:name_id |                None                |
|    os-vol-tenant-attr:tenant_id |     ccef9e62b1e645df98728fb2b3076f27 |
|              size             |                  8                   |
|          snapshot_id          |                 None                 |
```

```
|             source_volid            |                 None                  |
|                status               |               creating                |
|               user_id               |   fef060ae7bfd4024b3edb97dff59017a    |
|             volume_type             |                 None                  |
+-------------------------------------+---------------------------------------+
```

2. List volumes:

```
$ cinder list
+-------------------------------------+-----------+----------+------
+-------------+----------+-------------+
|                ID                   |  Status   |   Name   | Size |
 Volume Type | Bootable | Attached to |
+-------------------------------------+-----------+----------+------
+-------------+----------+-------------+
| d620d971-b160-4c4e-8652-2513d74e2080 | available | my-volume |  8   |
  None     | false    |             |
+-------------------------------------+-----------+----------+------
+-------------+----------+-------------+
```

3. Boot an instance from an image and attach the empty volume to the instance:

```
$ nova boot --flavor 2 --image 98901246-af91-43d8-b5e6-a4506aa8f369 \
        --block-device source=volume,id=d620d971-
b160-4c4e-8652-2513d74e2080,dest=volume,shutdown=preserve \
        myInstanceWithVolume
+------------------------------------
+-----------------------------------------------------------------+
| Property                           | Value
                                     |
+------------------------------------
+-----------------------------------------------------------------+
| OS-DCF:diskConfig                  | MANUAL
                                     |
| OS-EXT-AZ:availability_zone        | nova
                                     |
| OS-EXT-SRV-ATTR:host               | -
                                     |
| OS-EXT-SRV-ATTR:hypervisor_hostname | -
                                     |
| OS-EXT-SRV-ATTR:instance_name      | instance-00000004
                                     |
| OS-EXT-STS:power_state             | 0
                                     |
| OS-EXT-STS:task_state              | scheduling
                                     |
| OS-EXT-STS:vm_state                | building
                                     |
| OS-SRV-USG:launched_at             | -
                                     |
| OS-SRV-USG:terminated_at           | -
                                     |
| accessIPv4                         |
                                     |
| accessIPv6                         |
                                     |
| adminPass                          | ZaiYeC8iucgU
                                     |
| config_drive                       |
                                     |
```

```
| created                               | 2014-05-09T16:34:50Z
                                        |
| flavor                                | m1.small (2)
                                        |
| hostId                                |
                                        |
| id                                    | 1e1797f3-1662-49ff-ae8c-
a77e82ee1571                            |
| image                                 | cirros-0.3.1-x86_64-uec
 (98901246-af91-43d8-b5e6-a4506aa8f369) |
| key_name                              | -
                                        |
| metadata                              | {}
                                        |
| name                                  | myInstanceWithVolume
                                        |
| os-extended-volumes:volumes_attached  | [{"id": "d620d971-
b160-4c4e-8652-2513d74e2080"}]          |
| progress                              | 0
                                        |
| security_groups                       | default
                                        |
| status                                | BUILD
                                        |
| tenant_id                             | ccef9e62b1e645df98728fb2b3076f27
                                        |
| updated                               | 2014-05-09T16:34:51Z
                                        |
| user_id                               | fef060ae7bfd4024b3edb97dff59017a
                                        |
+------------------------------------
+------------------------------------------------------------+
```

## Create volume from image and boot instance

You can create a volume from an existing image, volume, or snapshot. This procedure shows you how to create a volume from an image, and use the volume to boot an instance.

1.  List the available images:

```
$ nova image-list
+------------------------------------+--------------------------------
+--------+--------+
| ID                                 | Name                          |
 Status | Server |
+------------------------------------+--------------------------------
+--------+--------+
| 484e05af-a14d-4567-812b-28122d1c2260 | Fedora-x86_64-20-20131211.1-sda |
 ACTIVE |        |
| 98901246-af91-43d8-b5e6-a4506aa8f369 | cirros-0.3.1-x86_64-uec       |
 ACTIVE |        |
| b6e95589-7eb2-4171-8bab-d225d9262c73 | cirros-0.3.1-x86_64-uec-kernel |
 ACTIVE |        |
| c90893ea-e732-40ac-a23d-4e36b2082c35 | cirros-0.3.1-x86_64-uec-ramdisk |
 ACTIVE |        |
+------------------------------------+--------------------------------
+--------+--------+
```

Note the ID of the image that you want to use to create a volume.

2.  List the available flavors:

```
$ nova flavor-list
+-----+-----------+-----------+------+-----------+------+-------
+-------------+-----------+
| ID  | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs |
 RXTX_Factor | Is_Public |
+-----+-----------+-----------+------+-----------+------+-------
+-------------+-----------+
| 1   | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0
    | True      |
| 2   | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0
    | True      |
| 3   | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0
    | True      |
| 4   | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0
    | True      |
| 42  | m1.nano   | 64        | 0    | 0         |      | 1     | 1.0
    | True      |
| 451 | m1.heat   | 1024      | 0    | 0         |      | 2     | 1.0
    | True      |
| 5   | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0
    | True      |
| 84  | m1.micro  | 128       | 0    | 0         |      | 1     | 1.0
    | True      |
+-----+-----------+-----------+------+-----------+------+-------
+-------------+-----------+
```

Note the ID of the flavor that you want to use to create a volume.

3.  To create a bootable volume from an image and launch an instance from this volume, use the $--block-device$ parameter.

For example:

```
$ nova boot --flavor FLAVOR --block-device source=SOURCE,id=ID,dest=DEST,
size=SIZE,shutdown=PRESERVE,bootindex=INDEX NAME
```

The parameters are:

*   $--flavor\ FLAVOR$. The flavor ID or name.

*   $--block-device$
    source=$SOURCE$,id=$ID$,dest=$DEST$,size=$SIZE$,shutdown=$PRESERVE$,bootindex=$INDEX$

    | | |
    |---|---|
    | **source=$SOURCE$** | The type of object used to create the block device. Valid values are `volume`, `snapshot`, `image`, and `blank`. |
    | **id=$ID$** | The ID of the source object. |
    | **dest=$DEST$** | The type of the target virtual device. Valid values are `volume` and `local`. |
    | **size=$SIZE$** | The size of the volume that is created. |

**shutdown={preserve|remove}**    What to do with the volume when the instance is deleted. `preserve` does not delete the volume. `remove` deletes the volume.

**bootindex=*INDEX***    Orders the boot disks. Use `0` to boot from this volume.

- *NAME*. The name for the server.

4. Create a bootable volume from an image, before the instance boots. The volume is not deleted when the instance is terminated:

```
$ nova boot --flavor 2 \
    --block-device source=image,id=e0b7734d-2331-42a3-b19e-067adc0da17d,
dest=volume,size=10,shutdown=preserve,bootindex=0 \
    myInstanceFromVolume
+-------------------------------------
+----------------------------------------------------+
| Property                            | Value
                    |
+-------------------------------------
+----------------------------------------------------+
| OS-EXT-STS:task_state               | scheduling
                    |
| image                               | Attempt to boot from volume - no
 image supplied |
| OS-EXT-STS:vm_state                 | building
                    |
| OS-EXT-SRV-ATTR:instance_name       | instance-00000003
                    |
| OS-SRV-USG:launched_at              | None
                    |
| flavor                              | m1.small
                    |
| id                                  | 2e65c854-dba9-4f68-8f08-
fe332e546ecc                |
| security_groups                     | [{u'name': u'default'}]
                    |
| user_id                             | 352b37f5c89144d4ad0534139266d51f
                    |
| OS-DCF:diskConfig                   | MANUAL
                    |
| accessIPv4                          |
                    |
| accessIPv6                          |
                    |
| progress                            | 0
                    |
| OS-EXT-STS:power_state              | 0
                    |
| OS-EXT-AZ:availability_zone         | nova
                    |
| config_drive                        |
                    |
| status                              | BUILD
                    |
| updated                             | 2014-02-02T13:29:54Z
                    |
```

```
| hostId                                  |
                    |
| OS-EXT-SRV-ATTR:host                    | None
                    |
| OS-SRV-USG:terminated_at                | None
                    |
| key_name                                | None
                    |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | None
                    |
| name                                    | myInstanceFromVolume
                    |
| adminPass                               | TzjqyGsRcJo9
                    |
| tenant_id                               | f7ac731cc11f40efbc03a9f9e1d1d21f
                    |
| created                                 | 2014-02-02T13:29:53Z
                    |
| os-extended-volumes:volumes_attached | []
                    |
| metadata                                | {}
                    |
+--------------------------------------
+--------------------------------------------------+
```

5.  List volumes to see the bootable volume and its attached `myInstanceFromVolume` instance:

```
$ cinder list
+--------------------------------------+--------+--------------+------
-------------+----------+--------------------------------------+
|                 ID                   | Status | Display Name | Size |
 Volume Type | Bootable |             Attached to              |
+--------------------------------------+--------+--------------+------
-------------+----------+--------------------------------------+
| 2fff50ab-1a9c-4d45-ae60-1d054d6bc868 | in-use |              | 10   |
   None    |   true   | 2e65c854-dba9-4f68-8f08-fe332e546ecc |
+--------------------------------------+--------+--------------+------
-------------+----------+--------------------------------------+
```

## Attach swap or ephemeral disk to an instance

Use the nova **boot** `--swap` parameter to attach a swap disk on boot or the nova **boot** `--ephemeral` parameter to attach an ephemeral disk on boot. When you terminate the instance, both disks are deleted.

Boot an instance with a 512 MB swap disk and 2 GB ephemeral disk:

```
$ nova boot --flavor FLAVOR --image IMAGE_ID --swap 512 --ephemeral size=
2 NAME
```

### Note

The flavor defines the maximum swap and ephemeral disk size. You cannot exceed these maximum values.

# Manage instances and hosts

Instances are virtual machines that run inside the cloud on physical compute nodes. The Compute service manages instances. A host is the node on which a group of instances resides.

This section describes how to perform the different tasks involved in instance management, such as adding floating IP addresses, stopping and starting instances, and terminating instances. This section also discusses node management tasks.

## Manage IP addresses

Each instance has a private, fixed IP address and can also have a public, or floating, address. Private IP addresses are used for communication between instances, and public addresses are used for communication with networks outside the cloud, including the Internet.

When you launch an instance, it is automatically assigned a private IP address that stays the same until you explicitly terminate the instance. Rebooting an instance has no effect on the private IP address.

A pool of floating IP addresses, configured by the cloud administrator, is available in Open-Stack Compute. The project quota defines the maximum number of floating IP addresses that you can allocate to the project. After you allocate a floating IP address to a project, you can:

- Associate the floating IP address with an instance of the project. Only one floating IP address can be allocated to an instance at any given time.

- Disassociate a floating IP address from an instance in the project.

- Delete a floating IP from the project; deleting a floating IP automatically deletes that IP's associations.

Use the **nova floating-ip-\*** commands to manage floating IP addresses.

### List floating IP address information

- To list all pools that provide floating IP addresses, run:

```
$ nova floating-ip-pool-list
+--------+
| name   |
+--------+
| public |
| test   |
+--------+
```

> **Note**
>
> If this list is empty, the cloud administrator must configure a pool of floating IP addresses.

- To list all floating IP addresses that are allocated to the current project, run:

```
$ nova floating-ip-list
+-------------+--------------------------------------+---------+--------+
| Ip          | Instance Id                          | Fixed Ip | Pool  |
+-------------+--------------------------------------+---------+--------+
| 172.24.4.225 | 4a60ff6a-7a3c-49d7-9515-86ae501044c6 | 10.0.0.2 | public |
| 172.24.4.226 | None                                 | None    | public |
+-------------+--------------------------------------+---------+--------+
```

For each floating IP address that is allocated to the current project, the command out-
puts the floating IP address, the ID for the instance to which the floating IP address is as-
signed, the associated fixed IP address, and the pool from which the floating IP address
was allocated.

## Associate floating IP addresses

You can assign a floating IP address to a project and to an instance.

1.  Run the following command to allocate a floating IP address to the current project. By
    default, the floating IP address is allocated from the `public` pool. The command out-
    puts the allocated IP address.

    ```
    $ nova floating-ip-create
    +-------------+-------------+---------+--------+
    | Ip          | Instance Id | Fixed Ip | Pool  |
    +-------------+-------------+---------+--------+
    | 172.24.4.225 | None        | None    | public |
    +-------------+-------------+---------+--------+
    ```

    ### Note

    If more than one IP address pool is available, you can specify the pool from
    which to allocate the IP address, using the pool's name. For example, to al-
    locate a floating IP address from the `test` pool, run:

    ```
    $ nova floating-ip-create test
    ```

2.  List all project instances with which a floating IP address could be associated:

    ```
    $ nova list
    +--------------------------------------+------+---------+------------
    +-------------+-----------------+
    | ID                                   | Name | Status  | Task State |
     Power State | Networks        |
    +--------------------------------------+------+---------+------------
    +-------------+-----------------+
    | d5c854f9-d3e5-4fce-94d9-3d9f9f8f2987 | VM1  | ACTIVE  | -          |
     Running     | private=10.0.0.3 |
    | 42290b01-0968-4313-831f-3b489a63433f | VM2  | SHUTOFF | -          |
     Shutdown    | private=10.0.0.4 |
    +--------------------------------------+------+---------+------------
    +-------------+-----------------+
    ```

3.  Associate an IP address with an instance in the project, as follows:

    ```
    $ nova floating-ip-associate INSTANCE_NAME_OR_ID FLOATING_IP_ADDRESS
    ```

    For example:

```
$ nova floating-ip-associate VM1 172.24.4.225
```

Notice that the instance is now associated with two IP addresses:

```
$ nova list
+------------------------------------+------+--------+------------
+-----------+------------------------------+
| ID                                 | Name | Status | Task State |
 Power State | Networks                  |
+------------------------------------+------+--------+------------
+-------------+------------------------------+
| d5c854f9-d3e5-4fce-94d9-3d9f9f8f2987 | VM1 | ACTIVE  | -         |
 Running     | private=10.0.0.3, 172.24.4.225 |
| 42290b01-0968-4313-831f-3b489a63433f | VM2 | SHUTOFF | -         |
 Shutdown    | private=10.0.0.4            |
+------------------------------------+------+--------+------------
+-------------+------------------------------+
```

After you associate the IP address and configure security group rules for the instance, the instance is publicly available at the floating IP address.

> **Note**
>
> If an instance is connected to multiple networks, you can associate a floating IP address with a specific fixed IP address using the optional `--fixed-address` parameter:
>
> ```
> $ nova floating-ip-associate --fixed-
> address FIXED_IP_ADDRESS INSTANCE_NAME_OR_ID FLOATING_IP_ADDRESS
> ```

## Disassociate floating IP addresses

1.  Release a floating IP address from an instance, as follows:

    ```
    $ nova floating-ip-disassociate INSTANCE_NAME_OR_ID FLOATING_IP_ADDRESS
    ```

2.  Release the floating IP address from the current project, as follows:

    ```
    $ nova floating-ip-delete FLOATING_IP_ADDRESS
    ```

    The IP address is returned to the pool of IP addresses that is available for all projects. If the IP address is still associated with a running instance, it is automatically disassociated from that instance.

# Change the size of your server

You change the size of a server by changing its flavor.

1.  Show information about your server, including its size, which is shown as the value of the flavor property.

    ```
    $ nova show myCirrosServer
          +-----------------------------------
    +----------------------------------------------------------+
    | Property                              | Value
                          |
    ```

```
+-----------------------------------
+----------------------------------------------------------------+
| status                              | ACTIVE
                              |
| updated                             | 2013-07-18T15:08:20Z
                              |
| OS-EXT-STS:task_state               | None
                              |
| OS-EXT-SRV-ATTR:host                | devstack
                              |
| key_name                            | None
                              |
| image                               | cirros-0.3.2-x86_64-uec (397e713c-
b95b-4186-ad46-6126863ea0a9) |
| private network                     | 10.0.0.3
                              |
| hostId                              |
 6e1e69b71ac9b1e6871f91e2dfc9a9b9ceca0f05db68172a81d45385        |
| OS-EXT-STS:vm_state                 | active
                              |
| OS-EXT-SRV-ATTR:instance_name       | instance-00000005
                              |
| OS-EXT-SRV-ATTR:hypervisor_hostname | devstack
                              |
| flavor                              | m1.small (2)
                              |
| id                                  | 84c6e57d-a6b1-44b6-81eb-
fcb36afd31b5                         |
| security_groups                     | [{u'name': u'default'}]
                              |
| user_id                             | 376744b5910b4b4da7d8e6cb483b06a8
                              |
| name                                | myCirrosServer
                              |
| created                             | 2013-07-18T15:07:59Z
                              |
| tenant_id                           | 66265572db174a7aa66eba661f58eb9e
                              |
| OS-DCF:diskConfig                   | MANUAL
                              |
| metadata                            | {u'description': u'Small test
 image', u'creator': u'joecool'}   |
| accessIPv4                          |
                              |
| accessIPv6                          |
                              |
| progress                            | 0
                              |
| OS-EXT-STS:power_state              | 1
                              |
| OS-EXT-AZ:availability_zone         | nova
                              |
| config_drive                        |
                              |
+-----------------------------------
+----------------------------------------------------------------+
```

The size (flavor) of the server is `m1.small (2)`.

2. List the available flavors with the following command:

```
$ nova flavor-list
+----+-----------+-----------+------+-----------+------+-------
+-------------+-----------+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs |
 RXTX_Factor | Is_Public |
+----+-----------+-----------+------+-----------+------+-------
+-------------+-----------+
| 1  | m1.tiny   | 512       | 0    | 0         |      | 1     | 1.0
    | True      |
| 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0
    | True      |
| 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0
    | True      |
| 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0
    | True      |
| 5  | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0
    | True      |
+----+-----------+-----------+------+-----------+------+-------
+-------------+-----------+
```

3.  To resize the server, pass the server ID or name and the new flavor to the **nova resize** command. Include the `--poll` parameter to report the resize progress.

```
$ nova resize myCirrosServer 4 --poll
```

```
Instance resizing... 100% complete
Finished
```

4.  Show the status for your server:

```
$ nova list
+--------------------------------------+---------------+--------
+----------------------------------------+
| ID                                   | Name          | Status |
 Networks                              |
+--------------------------------------+---------------+--------
+----------------------------------------+
| 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5 | myCirrosServer | RESIZE |
 private=172.16.101.6, public=10.4.113.6 |
+--------------------------------------+---------------+--------
+----------------------------------------+
```

When the resize completes, the status becomes VERIFY_RESIZE.

5.  Confirm the resize:

```
$ nova resize-confirm 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5
```

The server status becomes ACTIVE.

6.  If the resize fails or does not work as expected, you can revert the resize:

```
$ nova resize-revert 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5
```

The server status becomes ACTIVE.

# Search for an instance using IP address

You can search for an instance using the IP address parameter, `--ip`, with the **nova list** command.

```
$ nova list --ip IP_ADDRESS
```

The following example shows the results of a search on `10.0.0.4`.

```
$ nova list --ip 10.0.0.4
+------------------------------------+---------------------+--------
+-----------+------------+-----------------+
| ID                                 | Name                | Status | Task
 State | Power State | Networks        |
+------------------------------------+---------------------+--------
+-----------+------------+-----------------+
| 8a99547e-7385-4ad1-ae50-4ecfaaad5f42 | myInstanceFromVolume | ACTIVE | None
      | Running     | private=10.0.0.4 |
+------------------------------------+---------------------+--------
+-----------+------------+-----------------+
```

# Stop and start an instance

Use one of the following methods to stop and start an instance.

## Pause and unpause an instance

• To pause an instance, run the following command:

```
$ nova pause INSTANCE_NAME
```

This command stores the state of the VM in RAM. A paused instance continues to run in a frozen state.

• To unpause the instance, run the following command:

```
$ nova unpause INSTANCE_NAME
```

## Suspend and resume an instance

Administrative users might want to suspend an instance if it is infrequently used or to perform system maintenance. When you suspend an instance, its VM state is stored on disk, all memory is written to disk, and the virtual machine is stopped. Suspending an instance is similar to placing a device in hibernation.

### Note

Suspending an instance will not free allocated memory or virtual CPU resources. To release them, delete the instance.

• To initiate a hypervisor-level suspend operation, run the following command:

```
$ nova suspend INSTANCE_NAME
```

• To resume a suspended instance, run the following command:

```
$ nova resume INSTANCE_NAME
```

## Shelve and unshelve an instance

Shelving is useful if you have an instance that you are not using, but would like retain in your list of servers. For example, you can stop an instance at the end of a work week, and resume work again at the start of the next week. All associated data and resources are kept; however, anything still in memory is not retained. If a shelved instance is no longer needed, it can also be entirely removed.

You can complete the following shelving tasks:

**Shelve an instance**

Shuts down the instance, and stores it together with associated data and resources (a snapshot is taken if not volume backed). Anything in memory is lost. Use the following command:

```
$ nova shelve SERVERNAME
```

**Unshelve an instance**

Restores the instance:

```
$ nova unshelve SERVERNAME
```

**Remove a shelved instance**

Removes the instance from the server; data and resource associations are deleted. If an instance is no longer needed, you can move that instance off the hypervisor in order to minimize resource usage:

```
$ nova shelve-offload SERVERNAME
```

# Reboot an instance

You can soft or hard reboot a running instance. A soft reboot attempts a graceful shut down and restart of the instance. A hard reboot power cycles the instance.

• By default, when you reboot a server, it is a soft reboot.

```
$ nova reboot SERVER
```

To perform a hard reboot, pass the `--hard` parameter, as follows:

```
$ nova reboot --hard SERVER
```

# Delete an instance

When you no longer need an instance, you can delete it.

1. List all instances:

```
$ nova list
+------------------------------------+----------------------+--------
+-----------+-------------+-----------------+
| ID                                 | Name                 | Status |
 Task State | Power State | Networks        |
```

```
+--------------------------------------+---------------------+--------
-----------+------------+-----------------+
| 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5 | myCirrosServer      | ACTIVE |
 None         | Running     | private=10.0.0.3 |
| 8a99547e-7385-4ad1-ae50-4ecfaaad5f42 | myInstanceFromVolume | ACTIVE |
 None         | Running     | private=10.0.0.4 |
| d7efd3e4-d375-46d1-9d57-372b6e4bdb7f | newServer           | ERROR  |
 None         | NOSTATE     |                  |
+--------------------------------------+---------------------+--------
-----------+------------+-----------------+
```

2.  Run the **nova delete** command to delete the instance. The following example shows deletion of the `newServer` instance, which is in `ERROR` state:

    ```
    $ nova delete newServer
    ```

    The command does not notify that your server was deleted.

3.  To verify that the server was deleted, run the **nova list** command:

    ```
    $ nova list
    +--------------------------------------+---------------------+--------
    -----------+------------+-----------------+
    | ID                                   | Name                | Status |
     Task State | Power State | Networks        |
    +--------------------------------------+---------------------+--------
    -----------+------------+-----------------+
    | 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5 | myCirrosServer      | ACTIVE |
     None         | Running     | private=10.0.0.3 |
    | 8a99547e-7385-4ad1-ae50-4ecfaaad5f42 | myInstanceFromVolume | ACTIVE |
     None         | Running     | private=10.0.0.4 |
    +--------------------------------------+---------------------+--------
    -----------+------------+-----------------+
    ```

    The deleted instance does not appear in the list.

# Access an instance through a console

In order to see the console output of an instance, regardless of whether or not the console log has output, either VNC or SPICE is used. This allows for relaying keyboard and mouse activity to and from the instance in question.

There are three common remote console access methods in use with OpenStack:

**novnc**    An in-browser VNC client implemented using HTML5 Canvas and WebSockets

**spice**    A complete in-browser client solution for interaction with virtualized instances

**xvpvnc**   A Java client offering console access to an instance

Example:

To access an instance through a remote console, run the following command:

```
$ nova get-vnc-console INSTANCE_NAME VNC_TYPE
```

The command returns a URL from which you can access your instance:

```
+--------
+------------------------------------------------------------------------------
+
| Type   | Url
        |
+--------
+------------------------------------------------------------------------------
+
| xvpvnc | http://192.168.5.96:6081/console?token=c83ae3a3-15c4-4890-8d45-
aefb494a8d6c |
+--------
+------------------------------------------------------------------------------
+
```

*VNC_TYPE* can be replaced by any of the above values as connection types.

When using SPICE to view the console of an instance, a browser plugin can be used directly on the instance page, or the **get-vnc-console** command can be used with it, as well, by returning a token-authenticated address such as the one above.

For further information and comparisons (including security considerations, please see the security guide.

# Manage bare-metal nodes

The bare-metal driver for OpenStack Compute manages provisioning of physical hardware by using common cloud APIs and tools such as Orchestration (Heat). The use case for this driver is for single tenant clouds such as a high-performance computing cluster or for deploying OpenStack itself.

If you use the bare-metal driver, you must create a network interface and add it to a bare-metal node. Then, you can launch an instance from a bare-metal image.

### Note

Development efforts are focused on moving the driver out of the Compute code base in the Icehouse release.

You can list and delete bare-metal nodes. When you delete a node, any associated network interfaces are removed. You can list and remove network interfaces that are associated with a bare-metal node.

## Commands

The following commands can be used to manage bare-metal nodes.

- **baremetal-interface-add**. Adds a network interface to a bare-metal node.

- **baremetal-interface-list**. Lists network interfaces associated with a bare-metal node.

- **baremetal-interface-remove**. Removes a network interface from a bare-metal node.

- **baremetal-node-create**. Creates a bare-metal node.

- **baremetal-node-delete**. Removes a bare-metal node and any associated interfaces.

- **baremetal-node-list**. Lists available bare-metal nodes.

- **baremetal-node-show**. Shows information about a bare-metal node.

## Create a bare-metal node

When you create a bare-metal node, your PM address, user name, and password should match those that are configured in your hardware's BIOS/IPMI configuration.

```
$ nova baremetal-node-create --pm_address PM_ADDRESS --pm_user PM_USERNAME \
  --pm_password PM_PASSWORD $(hostname -f) 1 512 10 aa:bb:cc:dd:ee:ff
```

The following example shows the command and results from creating a node with the PM address `1.2.3.4`, the PM user name `ipmi`, and password `ipmi`.

```
$ nova baremetal-node-create --pm_address 1.2.3.4 --pm_user ipmi \
  --pm_password ipmi $(hostname -f) 1 512 10 aa:bb:cc:dd:ee:ff
+-----------------+-------------------+
| Property        | Value             |
+-----------------+-------------------+
| instance_uuid   | None              |
| pm_address      | 1.2.3.4           |
| interfaces      | []                |
| prov_vlan_id    | None              |
| cpus            | 1                 |
| memory_mb       | 512               |
| prov_mac_address| aa:bb:cc:dd:ee:ff |
| service_host    | ubuntu            |
| local_gb        | 10                |
| id              | 1                 |
| pm_user         | ipmi              |
| terminal_port   | None              |
+-----------------+-------------------+
```

## Add a network interface to the node:

For each NIC on the node, you must create an interface, specifying the interface's MAC address.

```
$ nova baremetal-interface-add 1 aa:bb:cc:dd:ee:ff
+-------------+-------------------+
| Property    | Value             |
+-------------+-------------------+
| datapath_id | 0                 |
| id          | 1                 |
| port_no     | 0                 |
| address     | aa:bb:cc:dd:ee:ff |
+-------------+-------------------+
```

## Launch an instance from a bare-metal image:

A bare-metal instance is an instance created directly on a physical machine without any virtualization layer running underneath it. Nova retains power control via IPMI. In some situations, Nova may retain network control via Neutron and OpenFlow.

```
$ nova boot --image my-baremetal-image --flavor my-baremetal-flavor test
```

```
+----------------------------+------------------------------------+
| Property                   | Value                              |
+----------------------------+------------------------------------+
| status                     | BUILD                              |
| id                         | cc302a8f-cd81-484b-89a8-b75eb3911b1b |
+----------------------------+------------------------------------+

... wait for instance to become active ...
```

> **Note**
>
> Set the `--availability_zone` parameter to specify which zone or node to
> use to start the server. Separate the zone from the host name with a comma.
> For example:
>
> ```
> $ nova boot --availability_zone zone:HOST,NODE
> ```
>
> `host` is optional for the `--availability_zone` parameter. You can specify
> simply `zone:,node`. You must still use the comma.

## List bare-metal nodes and interfaces:

Use the **nova baremetal-node-list** command to view all bare-metal nodes and interfaces.
When a node is in use, its status includes the UUID of the instance that runs on it:

```
$ nova baremetal-node-list
+----+--------+------+-----------+---------+-------------------+------
-----------+-------------+-------------+---------------+
| ID | Host   | CPUs | Memory_MB | Disk_GB | MAC Address       | VLAN | PM
 Address | PM Username | PM Password | Terminal Port |
+----+--------+------+-----------+---------+-------------------+------
-----------+-------------+-------------+---------------+
| 1  | ubuntu | 1    | 512       | 10      | aa:bb:cc:dd:ee:ff | None | 1.2.3.
4    | ipmi        |             | None          |
+----+--------+------+-----------+---------+-------------------+------
-----------+-------------+-------------+---------------+
```

## Show details for a bare-metal node:

Use the **nova baremetal-node-list** command to view the details for a bare-metal node.

```
$ nova baremetal-node-show 1
+-----------------+------------------------------------+
| Property        | Value                              |
+-----------------+------------------------------------+
| instance_uuid   | cc302a8f-cd81-484b-89a8-b75eb3911b1b |
| pm_address      | 1.2.3.4                            |
| interfaces      |
  [{u'datapath_id': u'0', u'id': 1, u'port_no': 0, u'address':
u'aa:bb:cc:dd:ee:ff'}] |
| prov_vlan_id    | None                               |
| cpus            | 1                                  |
| memory_mb       | 512                                |
| prov_mac_address | aa:bb:cc:dd:ee:ff                 |
| service_host    | ubuntu                             |
| local_gb        | 10                                 |
| id              | 1                                  |
| pm_user         | ipmi                               |
| terminal_port   | None                               |
```

```
+-----------------+-------------------------------------+
```

# Show usage statistics for hosts and instances

You can show basic statistics on resource usage for hosts and instances.

> **Note**
>
> For more sophisticated monitoring, see the ceilometer project. You can also use tools, such as Ganglia or Graphite, to gather more detailed data.

## Show host usage statistics

The following examples show the host usage statistics for a host called `devstack`.

• List the hosts and the nova-related services that run on them:

```
$ nova host-list
+----------+------------+----------+
| host_name | service    | zone     |
+----------+------------+----------+
| devstack  | conductor  | internal |
| devstack  | compute    | nova     |
| devstack  | cert       | internal |
| devstack  | network    | internal |
| devstack  | scheduler  | internal |
| devstack  | consoleauth | internal |
+----------+------------+----------+
```

• Get a summary of resource usage of all of the instances running on the host:

```
$ nova host-describe devstack
 +----------+----------------------------------+-----+----------+---------
+
| HOST      | PROJECT                          | cpu | memory_mb | disk_gb |
+----------+----------------------------------+-----+----------+---------+
| devstack  | (total)                          | 2   | 4003     | 157     |
| devstack  | (used_now)                       | 3   | 5120     | 40      |
| devstack  | (used_max)                       | 3   | 4608     | 40      |
| devstack  | b70d90d65e464582b6b2161cf3603ced | 1   | 512      | 0       |
| devstack  | 66265572db174a7aa66eba661f58eb9e | 2   | 4096     | 40      |
+----------+----------------------------------+-----+----------+---------+
```

The `cpu` column shows the sum of the virtual CPUs for instances running on the host.

The `memory_mb` column shows the sum of the memory (in MB) allocated to the instances that run on the host.

The `disk_gb` column shows the sum of the root and ephemeral disk sizes (in GB) of the instances that run on the host.

The row that has the value `used_now` in the `PROJECT` column shows the sum of the resources allocated to the instances that run on the host, plus the resources allocated to the virtual machine of the host itself.

The row that has the value `used_max` row in the `PROJECT` column shows the sum of the resources allocated to the instances that run on the host.

> **Note**
>
> These values are computed by using information about the flavors of the in-
> stances that run on the hosts. This command does not query the CPU usage,
> memory usage, or hard disk usage of the physical host.

## Show instance usage statistics

• Get CPU, memory, I/O, and network statistics for an instance.

1. List instances:

```
$ nova list
+--------------------------------------+--------------------+--------
+------------+-------------+------------------+
| ID                                   | Name               | Status |
 Task State | Power State | Networks         |
+--------------------------------------+--------------------+--------
+------------+-------------+------------------+
| 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5 | myCirrosServer     | ACTIVE |
 None       | Running     | private=10.0.0.3 |
| 8a99547e-7385-4ad1-ae50-4ecfaaad5f42 | myInstanceFromVolume | ACTIVE |
 None       | Running     | private=10.0.0.4 |
+--------------------------------------+--------------------+--------
+------------+-------------+------------------+
```

2. Get diagnostic statistics:

```
$ nova diagnostics myCirrosServer
+-----------------+----------------+
| Property        | Value          |
+-----------------+----------------+
| vnet1_rx        | 1210744        |
| cpu0_time       | 19624610000000 |
| vda_read        | 0              |
| vda_write       | 0              |
| vda_write_req   | 0              |
| vnet1_tx        | 863734         |
| vnet1_tx_errors | 0              |
| vnet1_rx_drop   | 0              |
| vnet1_tx_packets| 3855           |
| vnet1_tx_drop   | 0              |
| vnet1_rx_errors | 0              |
| memory          | 2097152        |
| vnet1_rx_packets| 5485           |
| vda_read_req    | 0              |
| vda_errors      | -1             |
+-----------------+----------------+
```

• Get summary statistics for each tenant:

```
$ nova usage-list
Usage from 2013-06-25 to 2013-07-24:
+------------------------------+-----------+--------------+----------
+--------------+
| Tenant ID                    | Instances | RAM MB-Hours | CPU Hours |
 Disk GB-Hours |
+------------------------------+-----------+--------------+----------
+--------------+
```

```
| b70d90d65e464582b6b2161cf3603ced | 1            | 344064.44    | 672.00     |
 0.00           |
| 66265572db174a7aa66eba661f58eb9e | 3            | 671626.76    | 327.94     |
 6558.86        |
+----------------------------------+------------+--------------+-----------
+--------------+
```

# Provide user data to instances

A *user data* file is a special key in the metadata service that holds a file that cloud-aware applications in the guest instance can access. For example, one application that uses user data is the cloud-init system, which is an open-source package from Ubuntu that is available on various Linux distributions and which handles early initialization of a cloud instance.

You can place user data in a local file and pass it through the `--user-data <user-data-file>` parameter at instance creation:

```
$ nova boot --image ubuntu-cloudimage --flavor 1 --user-data mydata.file
```

# Use snapshots to migrate instances

To use snapshots to migrate instances from OpenStack projects to clouds, complete these steps.

1. In the source project, perform the following steps:

    1. Create a snapshot of the instance.

    2. Download the snapshot as an image.

2. In the destination project, perform the following steps:

    1. Import the snapshot to the new environment.

    2. Boot a new instance from the snapshot.

    ### Note

    Some cloud providers allow only administrators to perform this task.

## Create a snapshot of the instance

1. Shut down the source VM before you take the snapshot to ensure that all data is flushed to disk. If necessary, list the instances to view get the instance name.

```
$ nova list
+--------------------------------------+------------+--------
+-----------------------------+
| ID                                   | Name       | Status | Networks
                    |
+--------------------------------------+------------+--------
+-----------------------------+
| c41f3074-c82a-4837-8673-fa7e9fea7e11 | myInstance | ACTIVE | private=10.
0.0.3            |
+--------------------------------------+------------+--------
+-----------------------------+
```

```
$ nova stop example
```

2. Use the **nova list** command to confirm that the instance shows a `SHUTOFF` status.

```
$ nova list
+--------------------------------------+------------+---------
+-----------------------------+
| ID                                   | Name       | Status  | Networks
                    |
+--------------------------------------+------------+---------
+-----------------------------+
| c41f3074-c82a-4837-8673-fa7e9fea7e11 | myInstance | SHUTOFF | private=
10.0.0.3            |
+--------------------------------------+------------+---------
+-----------------------------+
```

3. Use the **nova image-create** command to take a snapshot. Use the **nova image-list** command to check the status until the status is `ACTIVE`:

```
$ nova image-create --poll myInstance myInstanceSnapshot
Instance snapshotting... 50% complete
```

```
$ nova image-list
+------------------------------------+------------------------------
+--------+--------+
| ID                                 | Name
 Status | Server |
+------------------------------------+------------------------------
+--------+--------+
| 657ebb01-6fae-47dc-986a-e49c4dd8c433 | cirros-0.3.2-x86_64-uec          |
 ACTIVE |        |
| 72074c6d-bf52-4a56-a61c-02a17bf3819b | cirros-0.3.2-x86_64-uec-kernel   |
 ACTIVE |        |
| 3c5e5f06-637b-413e-90f6-ca7ed015ec9e | cirros-0.3.2-x86_64-uec-ramdisk |
 ACTIVE |        |
| f30b204e-1ce6-40e7-b8d9-b353d4d84e7d | myInstanceSnapshot               |
 ACTIVE |        |
+------------------------------------+------------------------------
+--------+--------+
```

# Download the snapshot as an image

1. Get the image ID:

```
$ nova image-list
+------------------------------------+------------------+--------
+------------------------------------+
| ID                                 | Name             | Status |
 Server                             |
+------------------------------------+------------------+--------
+------------------------------------+
| f30b204e-1ce6-40e7-b8d9-b353d4d84e7d | myInstanceSnapshot| ACTIVE |
 c41f3074-c82a-4837-8673-fa7e9fea7e11 |
+------------------------------------+------------------+--------
+------------------------------------+
```

2. Download the snapshot by using the image ID that was returned in the previous step:

```
$ glance image-download --file snapshot.raw f30b204e-1ce6-40e7-b8d9-
b353d4d84e7d
```

### Note

The **glance image-download** command requires the image ID and cannot use the image name.

Ensure there is sufficient space on the destination file system for the image file.

3. Make the image available to the new environment, either through HTTP or with direct upload to a machine (**scp**).

# Import the snapshot to new environment

In the new project or cloud environment, import the snapshot:

```
$ glance image-create --copy-from IMAGE_URL
```

# Boot a new instance from the snapshot

In the new project or cloud environment, use the snapshot to create the new instance:

```
$ nova boot --flavor m1.tiny --image myInstanceSnapshot myNewInstance
```

# Store metadata on a configuration drive

You can configure OpenStack to write metadata to a special configuration drive that attaches to the instance when it boots. The instance can mount this drive and read files from it to get information that is normally available through the metadata service. This metadata is different from the user data.

One use case for using the configuration drive is to pass a networking configuration when you do not use DHCP to assign IP addresses to instances. For example, you might pass the IP address configuration for the instance through the configuration drive, which the instance can mount and access before you configure the network settings for the instance.

Any modern guest operating system that is capable of mounting an ISO 9660 or VFAT file system can use the configuration drive.

# Requirements and guidelines

To use the configuration drive, you must follow the following requirements for the compute host and image.

### Compute host requirements

* The following hypervisors support the configuration drive: libvirt, XenServer, Hyper-V, and VMware.

* To use configuration drive with libvirt, XenServer, or VMware, you must first install the genisoimage package on each compute host. Otherwise, instances do not boot properly.

  Use the `mkisofs_cmd` flag to set the path where you install the genisoimage program. If genisoimage is in same path as the `nova-compute` service, you do not need to set this flag.

* To use configuration drive with Hyper-V, you must set the `mkisofs_cmd` value to the full path to an `mkisofs.exe` installation. Additionally, you must set the `qemu_img_cmd` value in the `hyperv` configuration section to the full path to an `qemu-img` command installation.

### Image requirements

* An image built with a recent version of the cloud-init package can automatically access metadata passed through the configuration drive. The cloud-init package version 0.7.1 works with Ubuntu and Fedora based images, such as Red Hat Enterprise Linux.

* If an image does not have the cloud-init package installed, you must customize the image to run a script that mounts the configuration drive on boot, reads the data from the drive, and takes appropriate action such as adding the public key to an account. See the section called "Configuration drive contents" [81] for details about how data is organized on the configuration drive.

* If you use Xen with a configuration drive, use the `xenapi_disable_agent` configuration parameter to disable the agent.

**Guidelines**

- Do not rely on the presence of the EC2 metadata in the configuration drive, because this content might be removed in a future release. For example, do not rely on files in the `ec2` directory.

- When you create images that access configuration drive data and multiple directories are under the `openstack` directory, always select the highest API version by date that your consumer supports. For example, if your guest image supports the 2012-03-05, 2012-08-05, and 2013-04-13 versions, try 2013-04-13 first and fall back to a previous version if 2013-04-13 is not present.

# Enable and access the configuration drive

1. To enable the configuration drive, pass the `--config-drive true` parameter to the **nova boot** command.

   The following example enables the configuration drive and passes user data, two files, and two key/value metadata pairs, all of which are accessible from the configuration drive:

   ```
   $ nova boot --config-drive true --image my-image-name --key-name mykey --
   flavor 1 --user-data ./my-user-data.txt myinstance --file /etc/network/
   interfaces=/home/myuser/instance-interfaces --file known_hosts=/home/
   myuser/.ssh/known_hosts --meta role=webservers --meta essential=false
   ```

   You can also configure the Compute service to always create a configuration drive by setting the following option in the `/etc/nova/nova.conf` file:

   ```
   force_config_drive=true
   ```

   > **Note**
   >
   > If a user passes the `--config-drive true` flag to the **nova boot** command, an administrator cannot disable the configuration drive.

2. If your guest operating system supports accessing disk by label, you can mount the configuration drive as the `/dev/disk/by-label/`*`configurationDriveVolume-`*`Label` device. In the following example, the configuration drive has the `config-2` volume label.

   ```
   # mkdir -p /mnt/config
   # mount /dev/disk/by-label/config-2 /mnt/config
   ```

   > **Note**
   >
   > Ensure that you use at least version 0.3.1 of CirrOS for configuration drive support.

   If your guest operating system does not use `udev`, the `/dev/disk/by-label` directory is not present.

You can use the **blkid** command to identify the block device that corresponds to the configuration drive. For example, when you boot the CirrOS image with the `m1.tiny` flavor, the device is `/dev/vdb`:

```
# blkid -t LABEL="config-2" -odevice
```

```
/dev/vdb
```

Once identified, you can mount the device:

```
# mkdir -p /mnt/config
# mount /dev/vdb /mnt/config
```

## Configuration drive contents

In this example, the contents of the configuration drive are as follows:

```
ec2/2009-04-04/meta-data.json
ec2/2009-04-04/user-data
ec2/latest/meta-data.json
ec2/latest/user-data
openstack/2012-08-10/meta_data.json
openstack/2012-08-10/user_data
openstack/content
openstack/content/0000
openstack/content/0001
openstack/latest/meta_data.json
openstack/latest/user_data
```

The files that appear on the configuration drive depend on the arguments that you pass to the **nova boot** command.

## OpenStack metadata format

The following example shows the contents of the `open-stack/2012-08-10/meta_data.json` and `openstack/latest/meta_data.json` files. These files are identical. The file contents are formatted for readability.

```
{
    "availability_zone": "nova",
    "files": [
        {
            "content_path": "/content/0000",
            "path": "/etc/network/interfaces"
        },
        {
            "content_path": "/content/0001",
            "path": "known_hosts"
        }
    ],
    "hostname": "test.novalocal",
    "launch_index": 0,
    "name": "test",
    "meta": {
        "role": "webservers",
        "essential": "false"
    },
```

```
    "public_keys": {
        "mykey": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAAgQDBqUfVvCSez0/
Wfpd8dLLgZXV9GtXQ7hnMN+Z0OWQUyebVEHey1CXuin0uY1cAJMhUq8j98SiW
+cU0sU4J3x5l2+xi1bodDm1BtFWVeLIOQINpfV1n8fKjHB
+ynPpe1F6tMDvrFGUlJs44t30BrujMXBe8Rq44cCk6wqyjATA3rQ== Generated by Nova\n"
    },
    "uuid": "83679162-1378-4288-a2d4-70e13ec132aa"
}
```

Note the effect of the `--file /etc/network/interfaces=/home/myuser/in-stance-interfaces` argument that was passed to the **nova boot** command. The contents of this file are contained in the `openstack/content/0000` file on the con-figuration drive, and the path is specified as `/etc/network/interfaces` in the `meta_data.json` file.

# EC2 metadata format

The following example shows the contents of the `ec2/2009-04-04/meta-data.json` and the `ec2/latest/meta-data.json` files. These files are identical. The file contents are formatted to improve readability.

```
{
    "ami-id": "ami-00000001",
    "ami-launch-index": 0,
    "ami-manifest-path": "FIXME",
    "block-device-mapping": {
        "ami": "sda1",
        "ephemeral0": "sda2",
        "root": "/dev/sda1",
        "swap": "sda3"
    },
    "hostname": "test.novalocal",
    "instance-action": "none",
    "instance-id": "i-00000001",
    "instance-type": "m1.tiny",
    "kernel-id": "aki-00000002",
    "local-hostname": "test.novalocal",
    "local-ipv4": null,
    "placement": {
        "availability-zone": "nova"
    },
    "public-hostname": "test.novalocal",
    "public-ipv4": "",
    "public-keys": {
        "0": {
            "openssh-key": "ssh-rsa
 AAAAB3NzaC1yc2EAAAADAQABAAAAgQDBqUfVvCSez0/
Wfpd8dLLgZXV9GtXQ7hnMN+Z0OWQUyebVEHey1CXuin0uY1cAJMhUq8j98SiW
+cU0sU4J3x5l2+xi1bodDm1BtFWVeLIOQINpfV1n8fKjHB
+ynPpe1F6tMDvrFGUlJs44t30BrujMXBe8Rq44cCk6wqyjATA3rQ== Generated by Nova\n"
        }
    },
    "ramdisk-id": "ari-00000003",
    "reservation-id": "r-7lfps8wj",
    "security-groups": [
        "default"
    ]
}
```

## User data

The `openstack/2012-08-10/user_data`, `openstack/latest/user_data`, `ec2/2009-04-04/user-data`, and `ec2/latest/user-data` file are present only if the `--user-data` flag and the contents of the user data file are passed to the **nova boot** command.

## Configuration drive format

The default format of the configuration drive as an ISO 9660 file system. To explicitly specify the ISO 9660 format, add the following line to the `/etc/nova/nova.conf` file:

```
config_drive_format=iso9660
```

By default, you cannot attach the configuration drive image as a CD drive instead of as a disk drive. To attach a CD drive, add the following line to the `/etc/nova/nova.conf` file:

```
config_drive_cdrom=true
```

For legacy reasons, you can configure the configuration drive to use VFAT format instead of ISO 9660. It is unlikely that you would require VFAT format because ISO 9660 is widely supported across operating systems. However, to use the VFAT format, add the following line to the `/etc/nova/nova.conf` file:

```
config_drive_format=vfat
```

If you choose VFAT, the configuration drive is 64 MB.

# Create and manage networks

Before you run commands, set the following environment variables:

```
export OS_USERNAME=admin
export OS_PASSWORD=password
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://localhost:5000/v2.0
```

## Create networks

1. List the extensions of the system:

```
$ neutron ext-list -c alias -c name

+-----------------+--------------------------+
| alias           | name                     |
+-----------------+--------------------------+
| agent_scheduler | Agent Schedulers         |
| binding         | Port Binding             |
| quotas          | Quota management support |
| agent           | agent                    |
| provider        | Provider Network         |
| router          | Neutron L3 Router        |
| lbaas           | LoadBalancing service    |
| extraroute      | Neutron Extra Route      |
+-----------------+--------------------------+
```

2. Create a network:

```
$ neutron net-create net1

Created a new network:
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| id                        | 2d627131-c841-4e3a-ace6-f2dd75773b6d |
| name                      | net1                                 |
| provider:network_type     | vlan                                 |
| provider:physical_network | physnet1                             |
| provider:segmentation_id  | 1001                                 |
| router:external           | False                                |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tenant_id                 | 3671f46ec35e4bbca6ef92ab7975e463     |
+---------------------------+--------------------------------------+
```

### Note

Some fields of the created network are invisible to non-admin users.

3. Create a network with specified provider network type:

```
$ neutron net-create net2 --provider:network-type local

Created a new network:
```

```
+--------------------------+-------------------------------------+
| Field                    | Value                               |
+--------------------------+-------------------------------------+
| admin_state_up           | True                                |
| id                       | 524e26ea-fad4-4bb0-b504-1ad0dc770e7a |
| name                     | net2                                |
| provider:network_type    | local                               |
| provider:physical_network|                                     |
| provider:segmentation_id |                                     |
| router:external          | False                               |
| shared                   | False                               |
| status                   | ACTIVE                              |
| subnets                  |                                     |
| tenant_id                | 3671f46ec35e4bbca6ef92ab7975e463    |
+--------------------------+-------------------------------------+
```

Just as shown previously, the unknown option `--provider:network-type` is used to create a `local` provider network.

# Create subnets

- Create a subnet:

```
$ neutron subnet-create net1 192.168.2.0/24 --name subnet1
```

```
Created a new subnet:
+------------------+--------------------------------------------------+
| Field            | Value                                            |
+------------------+--------------------------------------------------+
| allocation_pools | {"start": "192.168.2.2", "end": "192.168.2.254"} |
| cidr             | 192.168.2.0/24                                   |
| dns_nameservers  |                                                  |
| enable_dhcp      | True                                             |
| gateway_ip       | 192.168.2.1                                      |
| host_routes      |                                                  |
| id               | 15a09f6c-87a5-4d14-b2cf-03d97cd4b456             |
| ip_version       | 4                                                |
| name             | subnet1                                          |
| network_id       | 2d627131-c841-4e3a-ace6-f2dd75773b6d             |
| tenant_id        | 3671f46ec35e4bbca6ef92ab7975e463                 |
+------------------+--------------------------------------------------+
```

The **subnet-create** command has the following positional and optional parameters:

- The name or ID of the network to which the subnet belongs.

  In this example, `net1` is a positional argument that specifies the network name.

- The CIDR of the subnet.

  In this example, `192.168.2.0/24` is a positional argument that specifies the CIDR.

- The subnet name, which is optional.

  In this example, `--name subnet1` specifies the name of the subnet.

For information and examples on more advanced use of neutron's `subnet` subcommand, see the Cloud Administrator Guide.

# Create routers

1.  Create a router:

    ```
    $ neutron router-create router1
    ```

    ```
    Created a new router:
    +----------------------+--------------------------------------+
    | Field                | Value                                |
    +----------------------+--------------------------------------+
    | admin_state_up       | True                                 |
    | external_gateway_info |                                     |
    | id                   | 6e1f11ed-014b-4c16-8664-f4f615a3137a |
    | name                 | router1                              |
    | status               | ACTIVE                               |
    | tenant_id            | 7b5970fbe7724bf9b74c245e66b92abf     |
    +----------------------+--------------------------------------+
    ```

    Take note of the unique router identifier returned, this will be required in subsequent steps.

2.  Link the router to the external provider network:

    ```
    $ neutron router-gateway-set ROUTER NETWORK
    ```

    Replace *ROUTER* with the unique identifier of the router, replace *NETWORK* with the unique identifier of the external provider network.

3.  Link the router to the subnet:

    ```
    $ neutron router-interface-add ROUTER SUBNET
    ```

    Replace *ROUTER* with the unique identifier of the router, replace *SUBNET* with the unique identifier of the subnet.

# Create ports

1.  Create a port with specified IP address:

    ```
    $ neutron port-create net1 --fixed-ip ip_address=192.168.2.40
    ```

    ```
    Created a new port:
    +----------------------
    +-----------------------------------------------------------------------------------
    +
    | Field                | Value
                          |
    +----------------------
    +-----------------------------------------------------------------------------------
    +
    | admin_state_up       | True
                          |
    | binding:capabilities | {"port_filter": false}
                          |
    | binding:vif_type     | ovs
                          |
    | device_id            |
                          |
    ```

```
| device_owner         |
                                  |
| fixed_ips            | {"subnet_id": "15a09f6c-87a5-4d14-
b2cf-03d97cd4b456", "ip_address": "192.168.2.40"} |
| id                   | f7a08fe4-e79e-4b67-bbb8-a5002455a493
                                  |
| mac_address          | fa:16:3e:97:e0:fc
                                  |
| name                 |
                                  |
| network_id           | 2d627131-c841-4e3a-ace6-f2dd75773b6d
                                  |
| status               | DOWN
                                  |
| tenant_id            | 3671f46ec35e4bbca6ef92ab7975e463
                                  |
+---------------------
+-----------------------------------------------------------------------------
+
```

In the previous command, `net1` is the network name, which is a positional argument.
`--fixed-ip ip_address=192.168.2.40` is an option, which specifies the port's
fixed IP address we wanted.

> ### Note
>
> When creating a port, you can specify any unallocated IP in the subnet
> even if the address is not in a pre-defined pool of allocated IP addresses
> (set by your cloud provider).

2. Create a port without specified IP address:

```
$ neutron port-create net1
```

```
Created a new port:
+---------------------
+-----------------------------------------------------------------------------
+
| Field| Value
                                  |
+---------------------
+-----------------------------------------------------------------------------
+
| admin_state_up       | True
                                  |
| binding:capabilities | {"port_filter": false}
                                  |
| binding:vif_type     | ovs
                                  |
| device_id            |
                                  |
| device_owner         |
                                  |
| fixed_ips            | {"subnet_id": "15a09f6c-87a5-4d14-
b2cf-03d97cd4b456", "ip_address": "192.168.2.2"} |
| id                   | baf13412-2641-4183-9533-de8f5b91444c
                                  |
| mac_address          | fa:16:3e:f6:ec:c7
                                  |
```

```
| name                   |
                                     |
| network_id             | 2d627131-c841-4e3a-ace6-f2dd75773b6d
                                     |
| status                 | DOWN
                                     |
| tenant_id              | 3671f46ec35e4bbca6ef92ab7975e463
                                     |
+--------------------
+-----------------------------------------------------------------------------
+
```

**Note**

Note that the system allocates one IP address if you do not specify an IP address in the **neutron port-create** command.

3. Query ports with specified fixed IP addresses:

```
$ neutron port-list --fixed-ips ip_address=192.168.2.2 ip_address=192.168.
2.40

+------------------------------------+------+------------------
+-----------------------------------------------------------------------------
+
| id                                 | name | mac_address      |
 fixed_ips
         |
+------------------------------------+------+------------------
+-----------------------------------------------------------------------------
+
| baf13412-2641-4183-9533-de8f5b91444c |      | fa:16:3e:f6:ec:c7 |
 {"subnet_id": "15a09f6c-87a5-4d14-b2cf-03d97cd4b456", "ip_address": "192.
168.2.2"} |
| f7a08fe4-e79e-4b67-bbb8-a5002455a493 |      | fa:16:3e:97:e0:fc |
 {"subnet_id": "15a09f6c-87a5-4d14-b2cf-03d97cd4b456", "ip_address": "192.
168.2.40"} |
+------------------------------------+------+------------------
+-----------------------------------------------------------------------------
+
```

`--fixed-ips ip_address=192.168.2.2 ip_address=192.168.2.40` is one unknown option.

**How to find unknown options?** The unknown options can be easily found by watching the output of `create_xxx` or `show_xxx` command. For example, in the port creation command, we see the fixed_ips fields, which can be used as an unknown option.

# Manage objects and containers

The OpenStack Object Storage service provides the **swift** client, which is a command-line interface (CLI). Use this client to list objects and containers, upload objects to containers, and download or delete objects from containers. You can also gather statistics and update metadata for accounts, containers, and objects.

This client is based on the native swift client library, `client.py`, which seamlessly re-authenticates if the current token expires during processing, retries operations multiple times, and provides a processing concurrency of 10.

## Create and manage containers

- To create a container, run the following command and replace *CONTAINER* with the name of your container.

  ```
  $ swift post CONTAINER
  ```

- To list all containers, run the following command:

  ```
  $ swift list
  ```

- To check the status of containers, run the following command:

  ```
  $ swift stat
  ```

  ```
  Account: AUTH_7b5970fbe7724bf9b74c245e77c03bcg
  Containers: 2
  Objects: 3
  Bytes: 268826
  Accept-Ranges: bytes
  X-Timestamp: 1392683866.17952
  Content-Type: text/plain; charset=utf-8
  ```

  You can also use the **swift stat** command with the *ACCOUNT* or *CONTAINER* names as parameters.

  ```
  $ swift stat CONTAINER
  ```

  ```
  Account: AUTH_7b5970fbe7724bf9b74c245e77c03bcg
  Container: storage1
  Objects: 2
  Bytes: 240221
  Read ACL:
  Write ACL:
  Sync To:
  Sync Key:
  Accept-Ranges: bytes
  X-Timestamp: 1392683866.20180
  Content-Type: text/plain; charset=utf-8
  ```

## Manage access

- Users have roles on accounts. For example, a user with the admin role has full access to all containers and objects in an account. You can set access control lists (ACLs) at the container level and support lists for read and write access, which you set with the `X-Container-Read` and `X-Container-Write` headers.

To give a user read access, use the **swift post** command with the $-r$ parameter. To give a user write access, use the $-w$ parameter.

The following example enables the `testuser` user to read objects in the container:

```
$ swift post -r 'testuser'
```

You can also use this command with a list of users.

• If you use StaticWeb middleware to enable Object Storage to serve public web content, use `.r:`, followed by a list of allowed referrers.

The following command gives object access to all referring domains:

```
$ swift post -r '.r:*'
```

# Manage objects

• To upload an object to a container, run the following command:

```
$ swift upload CONTAINER OBJECT_FILENAME
```

To upload in chunks, for large files, run the following command:

```
$ swift upload -S CHUNK_SIZE CONTAINER OBJECT_FILENAME
```

• To check the status of the object, run the following command:

```
$ swift stat CONTAINER OBJECT_FILENAME
```

```
Account: AUTH_7b5970fbe7724bf9b74c245e77c03bcg
Container: storage1
Object: images
Content Type: application/octet-stream
Content Length: 211616
Last Modified: Tue, 18 Feb 2014 00:40:36 GMT
ETag: 82169623d55158f70a0d720f238ec3ef
Meta Orig-Filename: images.jpg
Accept-Ranges: bytes
X-Timestamp: 1392684036.33306
```

• To list the objects in a container, run the following command:

```
$ swift list CONTAINER
```

• To download an object from a container, run the following command:

```
$ swift download CONTAINER OBJECT_FILENAME
```

# Environment variables required to run examples

To run the cURL command examples for the Object Storage API requests, set these environment variables:

**publicURL**    The public URL that is the HTTP endpoint from where you can access Object Storage. It includes the Object Storage API version number and your account name. For example, `https://23.253.72.207/v1/my_account`.

**token**            The authentication token for Object Storage.

To obtain these values, run the **swift stat -v** command.

As shown in this example, the public URL appears in the `StorageURL` field, and the token appears in the `Auth Token` field:

```
StorageURL: https://23.253.72.207/v1/my_account
Auth Token: {token}
Account: my_account
Containers: 2
Objects: 3
Bytes: 47
Meta Book: MobyDick
X-Timestamp: 1389453423.35964
X-Trans-Id: txee55498935404a2caad89-0052dd3b77
Content-Type: text/plain; charset=utf-8
Accept-Ranges: bytes
```

# Object versioning

You can store multiple versions of your content so that you can recover from unintended overwrites. Object versioning is an easy way to implement version control, which you can use with any type of content.

### Note

You cannot version a large-object manifest file, but the large-object manifest file can point to versioned segments.

We strongly recommended that you put non-current objects in a different container than the container where current object versions reside.

### To enable and use object versioning

1.  To enable object versioning, ask your cloud provider to set the `allow_versions` option to `TRUE` in the container configuration file.

2.  Create an `archive` container to store older versions of objects:

    ```
    # curl -i $publicURL/archive -X PUT -H "Content-Length: 0" -H "X-Auth-
    Token: $token"
    ```

    ```
    HTTP/1.1 201 Created
    Content-Length: 0
    Content-Type: text/html; charset=UTF-8
    X-Trans-Id: tx46f8c29050834d88b8d7e-0052e1859d
    Date: Thu, 23 Jan 2014 21:11:57 GMT
    ```

3.  Create a `current` container to store current versions of objects.

    Include the `X-Versions-Location` header. This header defines the container that holds the non-current versions of your objects. You must UTF-8-encode and then URL-encode the container name before you include it in the `X-Versions-Location` header. This header enables object versioning for all objects in the `current` container. Changes to objects in the `current` container automatically create non-current versions in the `archive` container.

```
# curl -i $publicURL/current -X PUT -H "Content-Length: 0" -H "X-Auth-
Token: $token" -H "X-Versions-Location: archive"
```

```
HTTP/1.1 201 Created
Content-Length: 0
Content-Type: text/html; charset=UTF-8
X-Trans-Id: txb91810fb717347d09eec8-0052e18997
Date: Thu, 23 Jan 2014 21:28:55 GMT
```

4.  Create the first version of an object in the `current` container:

```
# curl -i $publicURL/current/my_object --data-binary 1 -X PUT -H "Content-
Length: 0" -H "X-Auth-Token: $token"
```

```
HTTP/1.1 201 Created
Last-Modified: Thu, 23 Jan 2014 21:31:22 GMT
Content-Length: 0
Etag: d41d8cd98f00b204e9800998ecf8427e
Content-Type: text/html; charset=UTF-8
X-Trans-Id: tx5992d536a4bd4fec973aa-0052e18a2a
Date: Thu, 23 Jan 2014 21:31:22 GMT
```

Nothing is written to the non-current version container when you initially **PUT** an object in the `current` container. However, subsequent **PUT** requests that edit an object trigger the creation of a version of that object in the `archive` container.

These non-current versions are named as follows:

```
<length><object_name><timestamp>
```

Where `length` is the 3-character, zero-padded hexadecimal character length of the object, `<object_name>` is the object name, and `<timestamp>` is the time when the object was initially created as a current version.

5.  Create a second version of the object in the `current` container:

```
# curl -i $publicURL/current/my_object --data-binary 2 -X PUT -H "Content-
Length: 0" -H "X-Auth-Token: $token"
```

```
HTTP/1.1 201 Created
Last-Modified: Thu, 23 Jan 2014 21:41:32 GMT
Content-Length: 0
Etag: d41d8cd98f00b204e9800998ecf8427e
Content-Type: text/html; charset=UTF-8
X-Trans-Id: tx468287ce4fc94eada96ec-0052e18c8c
Date: Thu, 23 Jan 2014 21:41:32 GMT
```

6.  Issue a **GET** request to a versioned object to get the current version of the object. You do not have to do any request redirects or metadata lookups.

List older versions of the object in the `archive` container:

```
# curl -i $publicURL/archive?prefix=009my_object -X GET -H "X-Auth-Token:
 $token"
```

```
HTTP/1.1 200 OK
Content-Length: 30
X-Container-Object-Count: 1
Accept-Ranges: bytes
```

```
X-Timestamp: 1390513280.79684
X-Container-Bytes-Used: 0
Content-Type: text/plain; charset=utf-8
X-Trans-Id: tx9a441884997542d3a5868-0052e18d8e
Date: Thu, 23 Jan 2014 21:45:50 GMT

009my_object/1390512682.92052
```

### Note

A **POST** request to a versioned object updates only the metadata for the object and does not create a new version of the object. New versions are created only when the content of the object changes.

7. Issue a **DELETE** request to a versioned object to remove the current version of the object and replace it with the next-most current version in the non-current container.

```
# curl -i $publicURL/current/my_object -X DELETE -H "X-Auth-Token: $token"
```

```
HTTP/1.1 204 No Content
Content-Length: 0
Content-Type: text/html; charset=UTF-8
X-Trans-Id: tx006d944e02494e229b8ee-0052e18edd
Date: Thu, 23 Jan 2014 21:51:25 GMT
```

List objects in the `archive` container to show that the archived object was moved back to the `current` container:

```
# curl -i $publicURL/archive?prefix=009my_object -X GET -H "X-Auth-Token:
 $token"
```

```
HTTP/1.1 204 No Content
Content-Length: 0
X-Container-Object-Count: 0
Accept-Ranges: bytes
X-Timestamp: 1390513280.79684
X-Container-Bytes-Used: 0
Content-Type: text/html; charset=UTF-8
X-Trans-Id: tx044f2a05f56f4997af737-0052e18eed
Date: Thu, 23 Jan 2014 21:51:41 GMT
```

This next-most current version carries with it any metadata last set on it. If you want to completely remove an object and you have five versions of it, you must **DELETE** it five times.

8. To disable object versioning for the `current` container, remove its `X-Versions-Location` metadata header by sending an empty key value.

```
# curl -i $publicURL/current -X PUT -H "Content-Length: 0" -H "X-Auth-
Token: $token" -H "X-Versions-Location: "
```

```
HTTP/1.1 202 Accepted
Content-Length: 76
Content-Type: text/html; charset=UTF-8
X-Trans-Id: txe2476de217134549996d0-0052e19038
Date: Thu, 23 Jan 2014 21:57:12 GMT

<html><h1>Accepted</h1><p>The request is accepted for processing.</p></
html>
```

# Serialized response formats

By default, the Object Storage API uses a `text/plain` response format. In addition, both JSON and XML data serialization response formats are supported.

**Note**

To run the cURL command examples, you must export environment variables.

To define the response format, use one of these methods:

| Method | Description |
|---|---|
| `format=`*`format`* query parameter | Append this parameter to the URL for a **GET** request, where *`format`* is `json` or `xml`. |
| `Accept` request header | Include this header in the **GET** request. The valid header values are: |
| | **text/plain** — Plain text response format. The default. |
| | **application/json**text — JSON data serialization response format. |
| | **application/xml**or `text/xml` — XML data serialization response format. |

### Example 2.1. JSON example with format query parameter

For example, this request uses the *`format`* query parameter to ask for a JSON response:

```
# curl -i $publicURL?format=json -X GET -H "X-Auth-Token: $token"
```

```
HTTP/1.1 200 OK
Content-Length: 96
X-Account-Object-Count: 1
X-Timestamp: 1389453423.35964
X-Account-Meta-Subject: Literature
X-Account-Bytes-Used: 14
X-Account-Container-Count: 2
Content-Type: application/json; charset=utf-8
Accept-Ranges: bytes
X-Trans-Id: tx274a77a8975c4a66aeb24-0052d95365
Date: Fri, 17 Jan 2014 15:59:33 GMT
```

Object Storage lists container names with additional information in JSON format:

```
[
    {
        "count":0,
        "bytes":0,
        "name":"janeausten"
    },
    {
        "count":1,
        "bytes":14,
        "name":"marktwain"
    }
]
```

### Example 2.2. XML example with Accept header

This request uses the `Accept` request header to ask for an XML response:

```
# curl -i $publicURL -X GET -H "X-Auth-Token: $token" -H "Accept: application/
xml; charset=utf-8"
```

```
HTTP/1.1 200 OK
Content-Length: 263
X-Account-Object-Count: 3
X-Account-Meta-Book: MobyDick
X-Timestamp: 1389453423.35964
X-Account-Bytes-Used: 47
X-Account-Container-Count: 2
Content-Type: application/xml; charset=utf-8
Accept-Ranges: bytes
X-Trans-Id: txf0b4c9727c3e491694019-0052e03420
Date: Wed, 22 Jan 2014 21:12:00 GMT
```

Object Storage lists container names with additional information in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<account name="AUTH_73f0aa26640f4971864919d0eb0f0880">
    <container>
        <name>janeausten</name>
        <count>2</count>
        <bytes>33</bytes>
    </container>
    <container>
        <name>marktwain</name>
        <count>1</count>
        <bytes>14</bytes>
    </container>
</account>
```

The remainder of the examples in this guide use standard, non-serialized responses. However, all **GET** requests that perform list operations accept the *format* query parameter or `Accept` request header.

# Large objects

By default, the content of an object cannot be greater than 5 GB. However, you can use a number of smaller objects to construct a large object. The large object is comprised of two types of objects:

• **Segment objects** store the object content. You can divide your content into segments, and upload each segment into its own segment object. Segment objects do not have any special features. You create, update, download, and delete segment objects just as you would normal objects.

• A **manifest object** links the segment objects into one logical large object. When you download a manifest object, Object Storage concatenates and returns the contents of the segment objects in the response body of the request. This behavior extends to the response headers returned by **GET** and **HEAD** requests. The `Content-Length` response header value is the total size of all segment objects. Object Storage calculates the `ETag` response header value by taking the `ETag` value of each segment, concatenating them together, and returning the MD5 checksum of the result. The manifest object types are:

**Static large objects**    The manifest object content is an ordered list of the names of the segment objects in JSON format. See the section called "Static large objects" [96].

**Dynamic large objects**    The manifest object has no content but it has a `X-Object-Manifest` metadata header. The value of this header is `{container}/{prefix}`, where `{container}` is the name of the container where the segment objects are stored, and `{prefix}` is a string that all segment objects have in common. See the section called "Dynamic large objects" [97].

### Note

If you make a **COPY** request by using a manifest object as the source, the new object is a normal, and not a segment, object. If the total size of the source segment objects exceeds 5 GB, the **COPY** request fails. However, you can make a duplicate of the manifest object and this new object can be larger than 5 GB.

## Static large objects

To create a static large object, divide your content into pieces and create (upload) a segment object to contain each piece.

You must record the `ETag` response header that the **PUT** operation returns. Alternatively, you can calculate the MD5 checksum of the segment prior to uploading and include this in the `ETag` request header. This ensures that the upload cannot corrupt your data.

List the name of each segment object along with its size and MD5 checksum in order.

Create a manifest object. Include the `?multipart-manifest=put` query string at the end of the manifest object name to indicate that this is a manifest object.

The body of the **PUT** request on the manifest object comprises a json list, where each element contains the following attributes:

**path**        The container and object name in the format: `{container-name}/{object-name}`.

**etag**        The MD5 checksum of the content of the segment object. This value must match the `ETag` of that object.

**size_bytes**  The size of the segment object. This value must match the `Content-Length` of that object.

### Example 2.3. Static large object manifest list

This example shows three segment objects. You can use several containers and the object names do not have to conform to a specific pattern, in contrast to dynamic large objects.

```
[
    {
        "path": "mycontainer/objseg1",
```

```
        "etag": "0228c7926b8b642dfb29554cd1f00963",
        "size_bytes": 1468006
    },
    {
        "path": "mycontainer/pseudodir/seg-obj2",
        "etag": "5bfc9ea51a00b790717eeb934fb77b9b",
        "size_bytes": 1572864
    },
    {
        "path": "other-container/seg-final",
        "etag": "b9c3da507d2557c1ddc51f27c54bae51",
        "size_bytes": 256
    }
]
```

The `Content-Length` request header must contain the length of the json content—not the length of the segment objects. However, after the **PUT** operation completes, the `Content-Length` metadata is set to the total length of all the object segments. A similar situation applies to the `ETag`. If used in the **PUT** operation, it must contain the MD5 checksum of the json content. The `ETag` metadata value is then set to be the MD5 checksum of the concatenated `ETag` values of the object segments. You can also set the `Content-Type` request header and custom object metadata.

When the **PUT** operation sees the *?multipart-manifest=put* query parameter, it reads the request body and verifies that each segment object exists and that the sizes and ETags match. If there is a mismatch, the **PUT**operation fails.

If everything matches, the manifest object is created. The `X-Static-Large-Object` metadata is set to `true` indicating that this is a static object manifest.

Normally when you perform a **GET** operation on the manifest object, the response body contains the concatenated content of the segment objects. To download the manifest list, use the *?multipart-manifest=get* query parameter. The resulting list is not formatted the same as the manifest you originally used in the **PUT** operation.

If you use the **DELETE** operation on a manifest object, the manifest object is deleted. The segment objects are not affected. However, if you add the *?multipart-manifest=delete* query parameter, the segment objects are deleted and if all are successfully deleted, the manifest object is also deleted.

To change the manifest, use a **PUT** operation with the *?multipart-manifest=put* query parameter. This request creates a manifest object. You can also update the object metadata in the usual way.

## Dynamic large objects

You must segment objects that are larger than 5 GB before you can upload them. You then upload the segment objects like you would any other object and create a dynamic large manifest object. The manifest object tells Object Storage how to find the segment objects that comprise the large object. The segments remain individually addressable, but retrieving the manifest object streams all the segments concatenated. There is no limit to the number of segments that can be a part of a single large object.

To ensure the download works correctly, you must upload all the object segments to the same container and ensure that each object name is prefixed in such a way that it sorts

in the order in which it should be concatenated. You also create and upload a manifest file. The manifest file is a zero-byte file with the extra `X-Object-Manifest {contain-er}/{prefix}` header, where `{container}` is the container the object segments are in and `{prefix}` is the common prefix for all the segments. You must UTF-8-encode and then URL-encode the container and common prefix in the `X-Object-Manifest` header.

It is best to upload all the segments first and then create or update the manifest. With this method, the full object is not available for downloading until the upload is complete. Also, you can upload a new set of segments to a second location and update the manifest to point to this new location. During the upload of the new segments, the original manifest is still available to download the first set of segments.

### Example 2.4. Upload segment of large object request: HTTP

```
PUT /{api_version}/{account}/{container}/{object} HTTP/1.1
Host: storage.clouddrive.com
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
ETag: 8a964ee2a5e88be344f36c22562a6486
Content-Length: 1
X-Object-Meta-PIN: 1234
```

No response body is returned. A status code of $2nn$ (between 200 and 299, inclusive) indicates a successful write; status 411 Length Required denotes a missing `Content-Length` or `Content-Type` header in the request. If the MD5 checksum of the data written to the storage system does NOT match the (optionally) supplied ETag value, a 422 Unprocessable Entity response is returned.

You can continue uploading segments, like this example shows, prior to uploading the manifest.

### Example 2.5. Upload next segment of large object request: HTTP

```
PUT /{api_version}/{account}/{container}/{object} HTTP/1.1
Host: storage.clouddrive.com
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
ETag: 8a964ee2a5e88be344f36c22562a6486
Content-Length: 1
X-Object-Meta-PIN: 1234
```

Next, upload the manifest you created that indicates the container where the object segments reside. Note that uploading additional segments after the manifest is created causes the concatenated object to be that much larger but you do not need to recreate the manifest file for subsequent additional segments.

### Example 2.6. Upload manifest request: HTTP

```
PUT /{api_version}/{account}/{container}/{object} HTTP/1.1
Host: storage.clouddrive.com
X-Auth-Token: eaaafd18-0fed-4b3a-81b4-663c99ec1cbb
Content-Length: 0
X-Object-Meta-PIN: 1234
X-Object-Manifest: {container}/{prefix}
```

### Example 2.7. Upload manifest response: HTTP

```
[...]
```

The `Content-Type` in the response for a **GET** or **HEAD** on the manifest is the same as the `Content-Type` set during the **PUT** request that created the manifest. You can change the `Content-Type` by reissuing the **PUT** request.

## Comparison of static and dynamic large objects

While static and dynamic objects have similar behavior, this table describes their differences:

### Table 2.5. Static and dynamic large objects

|  | Static large object | Dynamic large object |
| --- | --- | --- |
| End-to-end integrity | Assured. The list of segments includes the MD5 checksum (`ETag`) of each segment. You cannot upload the manifest object if the `ETag` in the list differs from the uploaded segment object. If a segment is somehow lost, an attempt to download the manifest object results in an error. | Not guaranteed. The eventual consistency model means that although you have uploaded a segment object, it might not appear in the container listing until later. If you download the manifest before it appears in the container, it does not form part of the content returned in response to a **GET** request. |
| Upload order | You must upload the segment objects before you upload the manifest object. | You can upload manifest and segment objects in any order. You are recommended to upload the manifest object after the segments in case a premature download of the manifest occurs. However, this is not enforced. |
| Removal or addition of segment objects | You cannot add or remove segment objects from the manifest. However, you can create a completely new manifest object of the same name with a different manifest list. | You can upload new segment objects or remove existing segments. The names must simply match the {prefix} supplied in `X-Object-Manifest`. |
| Segment object size and number | Segment objects must be at least 1 MB in size (by default). The final segment object can be any size. At most, 1000 segments are supported (by default). | Segment objects can be any size. |
| Segment object container name | The manifest list includes the container name of each object. Segment objects can be in different containers. | All segment objects must be in the same container. |
| Manifest object metadata | The object has `X-Static-Large-Object` set to `true`. You do not set this metadata directly. Instead the system sets it when you **PUT** a static manifest object. | The `X-Object-Manifest` value is the {container}/{prefix}, which indicates where the segment objects are located. You supply this request header in the **PUT** operation. |
| Copying the manifest object | Include the `?multipart-manifest=get` query string in the **COPY** request. The new object contains the same manifest as the original. The segment objects are not copied. Instead, both the original and new manifest objects share the same set of segment objects. | The **COPY** operation does not create a manifest object. To duplicate a manifest object, use the **GET** operation to read the value of `X-Object-Manifest` and use this value in the `X-Object-Manifest` request header in a **PUT** operation. This creates a new manifest object that shares the same set of segment objects as the original manifest object. |

## Page through large lists of containers or objects

If you have a large number of containers or objects, you can use the *marker*, *limit*, and *end_marker* parameters to control how many items are returned in a list and where the list starts or ends.

*marker*          When you request a list of containers or objects, Object Storage returns a maximum of 10,000 names for each request. To get subsequent names, you must make another request with the *marker* parameter. Set the

marker parameter to the name of the last item returned in the previous list. You must URL-encode the *marker* value before you send the HTTP request. Object Storage returns a maximum of 10,000 names starting after the last item returned.

*limit*　　　　　To return fewer than 10,000 names, use the *limit* parameter. If the number of names returned equals the specified *limit* (or 10,000 if you omit the *limit* parameter), you can assume there are more names to list. If the number of names in the list is exactly divisible by the *limit* value, the last request has no content.

*end_marker*　Limits the result set to names that are less than the *end_marker* parameter value. You must URL-encode the *end_marker* value before you send the HTTP request.

### To page through a large list of containers

Assume the following list of container names:

```
apples
bananas
kiwis
oranges
pears
```

1.  Use a *limit* of two:

    ```
    # curl -i $publicURL/?limit=2 -X GET -H "X-Auth-Token: $token"
    ```

    ```
    apples
    bananas
    ```

    Because two container names are returned, there are more names to list.

2.  Make another request with a *marker* parameter set to the name of the last item returned:

    ```
    # curl -i $publicURL/?limit=2&amp;marker=bananas -X GET -H "X-Auth-Token:
     $token"
    ```

    ```
    kiwis
    oranges
    ```

    Again, two items are returned, and there might be more.

3.  Make another request with a *marker* of the last item returned:

    ```
    # curl -i $publicURL/?limit=2&amp;marker=oranges -X GET -H "X-Auth-Token:
     $token"
    ```

    ```
    pears
    ```

    You receive a one-item response, which is fewer than the *limit* number of names. This indicates that this is the end of the list.

4.  Use the *end_marker* parameter to limit the result set to object names that are less than the *end_marker* parameter value:

```
# curl -i $publicURL/?end_marker=oranges -X GET -H "X-Auth-Token: $token"
```

```
apples
bananas
kiwis
```

You receive a result set of all container names before the `end-marker` value.

# Auto-extract archive files

Use the auto-extract archive feature to upload a tar archive file.

The Object Storage system extracts files from the archive file and creates an object.

## Auto-extract archive request

To upload an archive file, make a **PUT** request. Add the `extract-archive=format` query parameter to indicate that you are uploading a tar archive file instead of normal content.

Valid values for the `format` variable are `tar`, `tar.gz`, or `tar.bz2`.

The path you specify in the **PUT** request is used for the location of the object and the prefix for the resulting object names.

In the **PUT** request, you can specify the path for:

• An account

• Optionally, a specific container

• Optionally, a specific object prefix

For example, if the first object in the tar archive is `/home/file1.txt` and you specify the `/v1/12345678912345/mybackup/castor/` path, the operation creates the `castor/home/file1.txt` object in the `mybackup` container in the `12345678912345` account.

## Create an archive for auto-extract

You must use the tar utility to create the tar archive file.

You can upload regular files but you cannot upload other items (for example, empty directories or symbolic links).

You must UTF-8-encode the member names.

The archive auto-extract feature supports these formats:

• The POSIX.1-1988 Ustar format.

• The GNU tar format. Includes the long name, long link, and sparse extensions.

• The POSIX.1-2001 pax format.

Use gzip or bzip2 to compress the archive.

Use the `extract-archive` query parameter to specify the format. Valid values for this parameter are `tar`, `tar.gz`, or `tar.bz2`.

## Auto-extract archive response

When Object Storage processes the request, it performs multiple sub-operations. Even if all sub-operations fail, the operation returns a 201 `Created` status. Some sub-operations might succeed while others fail: Examine the response body to determine the results of each auto-extract archive sub-operation.

You can set the `Accept` request header to one of these values to define the response format:

**text/plain**    Formats response as plain text. If you omit the `Accept` header, `text/plain` is the default.

**application/json**    Formats response as JSON.

**application/xml**    Formats response as XML.

**text/xml**    Formats response as XML.

The following auto-extract archive files example shows a `text/plain` response body where no failures occurred:

```
Number Files Created: 10
Errors:
```

The following auto-extract archive files example shows a `text/plain` response where some failures occurred. In this example, the Object Storage system is configured to reject certain character strings so that the 400 Bad Request error occurs for any objects that use the restricted strings.

```
Number Files Created: 8
Errors:
/v1/12345678912345/mycontainer/home/xx%3Cyy, 400 Bad Request
/v1/12345678912345/mycontainer/../image.gif, 400 Bad Request
```

The following example shows the failure response in `application/json` format.

```
{
    "Number Files Created":1,
    "Errors":[
        [
            "/v1/12345678912345/mycontainer/home/xx%3Cyy",
            "400 Bad Request"
        ],
        [
            "/v1/12345678912345/mycontainer/../image.gif",
            "400 Bad Request"
        ]
    ]
}
```

# Create and manage stacks

The Orchestration module enables you to orchestrate multiple composite cloud applications. This module supports use of both the Amazon Web Services (AWS) CloudFormation template format through both a Query API that is compatible with CloudFormation and the native OpenStack *Heat Orchestration Template (HOT)* format through a REST API.

These flexible template languages enable application developers to describe and automate the deployment of infrastructure, services, and applications. The templates enable creation of most OpenStack resource types, such as instances, floating IP addresses, volumes, security groups, and users. The resources, once created, are referred to as stacks.

The template languages are described in the Template Guide in the Heat developer documentation.

# Create a stack from an example template file

- To create a stack, or template, from an example template file, run the following command:

```
$ heat stack-create mystack --template-file /PATH_TO_HEAT_TEMPLATES/
WordPress_Single_Instance.template
    --parameters "InstanceType=m1.
large;DBUsername=USERNAME;DBPassword=PASSWORD;KeyName=HEAT_KEY;LinuxDistribution=
F17"
```

The `--parameters` values that you specify depend on the parameters that are defined in the template. If a website hosts the template file, you can specify the URL with the `--template-url` parameter instead of the `--template-file` parameter.

The command returns the following output:

```
+--------------------------------------+--------------+--------------------+---------------------+
| id                                   | stack_name   | stack_status       | creation_time       |
+--------------------------------------+--------------+--------------------+---------------------+
| 4c712026-dcd5-4664-90b8-0915494c1332 | mystack      | CREATE_IN_PROGRESS | 2013-04-03T23:22:08Z |
+--------------------------------------+--------------+--------------------+---------------------+
```

- You can also use the **template-validate** command to validate a template file without creating a stack from it.

> ### Note
>
> Previous versions of the heat client used **validate** instead of **template-validate**, but it has been deprecated in favor of **template-validate**.

To do so, run the following command:

```
$ heat template-validate --template-file /PATH_TO_HEAT_TEMPLATES/
WordPress_Single_Instance.template
```

If validation fails, the response returns an error message.

# Get information about stacks

To explore the state and history of a particular stack, you can run a number of commands.

• To see which stacks are visible to the current user, run the following command:

```
$ heat stack-list
+------------------------------------+--------------+-----------------
+---------------------+
| id                                 | stack_name   | stack_status    |
 creation_time        |
+------------------------------------+--------------+-----------------
+---------------------+
| 4c712026-dcd5-4664-90b8-0915494c1332 | mystack      | CREATE_COMPLETE |
 2013-04-03T23:22:08Z |
| 7edc7480-bda5-4e1c-9d5d-f567d3b6a050 | my-otherstack | CREATE_FAILED   |
 2013-04-03T23:28:20Z |
+------------------------------------+--------------+-----------------
+---------------------+
```

• To show the details of a stack, run the following command:

```
$ heat stack-show mystack
```

• A stack consists of a collection of resources. To list the resources and their status, run the following command:

```
$ heat resource-list mystack
+-------------------+-------------------+-----------------
+---------------------+
| logical_resource_id | resource_type     | resource_status | updated_time
         |
+-------------------+-------------------+-----------------
+---------------------+
| WikiDatabase       | AWS::EC2::Instance | CREATE_COMPLETE |
 2013-04-03T23:25:56Z |
+-------------------+-------------------+-----------------
+---------------------+
```

• To show the details for a specific resource in a stack, run the following command:

```
$ heat resource-show mystack WikiDatabase
```

• Some resources have associated metadata which can change throughout the life cycle of a resource. Show the metadata by running the following command:

```
$ heat resource-metadata mystack WikiDatabase
```

• A series of events is generated during the life cycle of a stack. To display life cycle events, run the following command::

```
$ heat event-list mystack
+-------------------+----+-----------------------+-----------------
+---------------------+
| logical_resource_id | id | resource_status_reason | resource_status |
 event_time        |
```

```
+--------------------+----+----------------------+----------------
+--------------------+
| WikiDatabase       | 1  | state changed        | IN_PROGRESS    |
 2013-04-03T23:22:09Z |
| WikiDatabase       | 2  | state changed        | CREATE_COMPLETE |
 2013-04-03T23:25:56Z |
+--------------------+----+----------------------+----------------
+--------------------+
```

• To show the details for a particular event, run the following command:

```
$ heat event-show WikiDatabase 1
```

# Update a stack

To update an existing stack from a modified template file, run a command like the following command:

```
$ heat stack-update mystack --template-file /path/to/heat/templates/
WordPress_Single_Instance_v2.template
     --parameters "InstanceType=m1.large;DBUsername=wp;DBPassword=
verybadpassword;KeyName=heat_key;LinuxDistribution=F17"
+-----------------------------------+--------------+-----------------
+--------------------+
| id                                | stack_name   | stack_status    |
 creation_time        |
+-----------------------------------+--------------+-----------------
+--------------------+
| 4c712026-dcd5-4664-90b8-0915494c1332 | mystack      | UPDATE_COMPLETE |
 2013-04-03T23:22:08Z |
| 7edc7480-bda5-4e1c-9d5d-f567d3b6a050 | my-otherstack | CREATE_FAILED   |
 2013-04-03T23:28:20Z |
+-----------------------------------+--------------+-----------------
+--------------------+
```

Some resources are updated in-place, while others are replaced with new resources.

# Measure cloud resources

Telemetry measures cloud resources in OpenStack. It collects data related to billing. Currently, this metering service is available through only the **ceilometer** command-line client.

To model data, Telemetry uses the following abstractions:

**Meter**        Measures a specific aspect of resource usage, such as the existence of a running instance, or ongoing performance, such as the CPU utilization for an instance. Meters exist for each type of resource. For example, a separate `cpu_util` meter exists for each instance. The life cycle of a meter is decoupled from the existence of its related resource. The meter persists after the resource goes away.

A meter has the following attributes:

- String name

- A unit of measurement

- A type, which indicates whether values increase monotonically (cumulative), are interpreted as a change from the previous value (delta), or are stand-alone and relate only to the current duration (gauge)

**Sample**        An individual data point that is associated with a specific meter. A sample has the same attributes as the associated meter, with the addition of time stamp and value attributes. The value attribute is also known as the sample *volume*.

**Statistic**        A set of data point aggregates over a time duration. (In contrast, a sample represents a single data point.) The Telemetry service employs the following aggregation functions:

- **count**. The number of samples in each period.

- **max**. The maximum number of sample volumes in each period.

- **min**. The minimum number of sample volumes in each period.

- **avg**. The average of sample volumes over each period.

- **sum**. The sum of sample volumes over each period.

**Alarm**        A set of rules that define a monitor and a current state, with edge-triggered actions associated with target states. Alarms provide user-oriented Monitoring-as-a-Service and a general purpose utility for OpenStack. Orchestration auto scaling is a typical use case. Alarms follow a tristate model of `ok`, `alarm`, and `insufficient data`. For conventional threshold-oriented alarms, a static threshold value and comparison operator govern state transitions. The comparison operator compares a selected meter statistic against an evaluation window of configurable length into the recent past.

This example uses the **heat** client to create an auto-scaling stack and the **ceilometer** client to measure resources.

1. Create an auto-scaling stack by running the following command. The `-f` option speci-
   fies the name of the stack template file, and the `-P` option specifies the `KeyName` pa-
   rameter as `heat_key`.

```
$ heat stack-create -f cfn/F17/AutoScalingCeilometer.yaml -P "KeyName=
heat_key"
```

2. List the heat resources that were created:

```
$ heat resource-list
+------------------------+---------------------------------------
+----------------+----------------------+
| resource_name          | resource_type                         |
resource_status | updated_time         |
+------------------------+---------------------------------------
+----------------+----------------------+
| CfnUser                | AWS::IAM::User                        |
CREATE_COMPLETE | 2013-10-02T05:53:41Z |
| WebServerKeys          | AWS::IAM::AccessKey                    |
CREATE_COMPLETE | 2013-10-02T05:53:42Z |
| LaunchConfig           | AWS::AutoScaling::LaunchConfiguration  |
CREATE_COMPLETE | 2013-10-02T05:53:43Z |
| ElasticLoadBalancer    | AWS::ElasticLoadBalancing::LoadBalancer |
UPDATE_COMPLETE | 2013-10-02T05:55:58Z |
| WebServerGroup         | AWS::AutoScaling::AutoScalingGroup     |
CREATE_COMPLETE | 2013-10-02T05:55:58Z |
| WebServerScaleDownPolicy | AWS::AutoScaling::ScalingPolicy      |
CREATE_COMPLETE | 2013-10-02T05:56:00Z |
| WebServerScaleUpPolicy   | AWS::AutoScaling::ScalingPolicy      |
CREATE_COMPLETE | 2013-10-02T05:56:00Z |
| CPUAlarmHigh           | OS::Ceilometer::Alarm                 |
CREATE_COMPLETE | 2013-10-02T05:56:02Z |
| CPUAlarmLow            | OS::Ceilometer::Alarm                 |
CREATE_COMPLETE | 2013-10-02T05:56:02Z |
+------------------------+---------------------------------------
+----------------+----------------------+
```

3. List the alarms that are set:

```
$ ceilometer alarm-list
+--------------------------------------+----------------------------
+------------------+---------+------------
+------------------------------+
| Alarm ID                             | Name                       |
 State             | Enabled | Continuous | Alarm condition
 |
+--------------------------------------+----------------------------
+------------------+---------+------------
+------------------------------+
| 4f896b40-0859-460b-9c6a-b0d329814496 | as-CPUAlarmLow-i6qqgkf2fubs  |
 insufficient data | True    | False      | cpu_util < 15.0 during 1x 60s
 |
| 75d8ecf7-afc5-4bdc-95ff-19ed9ba22920 | as-CPUAlarmHigh-sf4muyfruy5m |
 insufficient data | True    | False      | cpu_util > 50.0 during 1x 60s
 |
+--------------------------------------+----------------------------
+------------------+---------+------------+------------------------------+
```

4. List the meters that are set:

```
$ ceilometer meter-list
+------------------------+------------+----------
+----------------------------------------------------------------------
+------------------------------------+
| Name                   | Type       | Unit     | Resource ID
            | User ID                          | Project ID
        |
+------------------------+------------+----------
+----------------------------------------------------------------------
+------------------------------------+
| cpu                    | cumulative | ns       | 3965b41b-81b0-4386-
bea5-6ec37c8841c1 | d1a2996d3b1f4e0e8645ba9650308011 |
 bf03bf32e3884d489004ac995ff7a61c |
| cpu                    | cumulative | ns       | 62520a83-73c7-4084-
be54-275fe770ef2c | d1a2996d3b1f4e0e8645ba9650308011 |
 bf03bf32e3884d489004ac995ff7a61c |
| cpu_util               | gauge      | %        | 3965b41b-81b0-4386-
bea5-6ec37c8841c1 | d1a2996d3b1f4e0e8645ba9650308011 |
 bf03bf32e3884d489004ac995ff7a61c |
+------------------------+------------+----------
+----------------------------------------------------------------------
+------------------------------------+
```

5.  List samples:

```
$ ceilometer sample-list -m cpu_util
+------------------------------------+----------+-------+--------------
+------+--------------------+
| Resource ID                        | Name     | Type  | Volume
 | Unit | Timestamp          |
+------------------------------------+----------+-------+--------------
+------+--------------------+
| 3965b41b-81b0-4386-bea5-6ec37c8841c1 | cpu_util | gauge | 3.98333333333
 | %    | 2013-10-02T10:50:12 |
+------------------------------------+----------+-------+--------------
+------+--------------------+
```

6.  View statistics:

```
$ ceilometer statistics -m cpu_util
+--------+--------------------+--------------------+-------
+--------------+--------------+--------------+---------------
+----------+--------------------+--------------------+
| Period | Period Start       | Period End         | Count | Min
   | Max          | Sum          | Avg          | Duration | Duration
 Start    | Duration End       |
+--------+--------------------+--------------------+-------
+--------------+--------------+--------------+---------------
+----------+--------------------+--------------------+
| 0      | 2013-10-02T10:50:12 | 2013-10-02T10:50:12 | 1     | 3.
98333333333 | 3.98333333333 | 3.98333333333 | 3.98333333333 | 0.0      |
 2013-10-02T10:50:12 | 2013-10-02T10:50:12 |
+--------+--------------------+--------------------+-------
+--------------+--------------+--------------+---------------
+----------+--------------------+--------------------+
```

# Manage volumes

A volume is a detachable block storage device, similar to a USB hard drive. You can attach a volume to only one instance. To create and manage volumes, you use a combination of **no-va** and **cinder** client commands.

## Create a volume

This example creates a my-new-volume volume based on an image.

1. List images, and note the ID of the image that you want to use for your volume:

```
$ nova image-list
```

```
+--------------------------------------+-------------------------------+--------
+--------------------------------------+
| ID                                   | Name                          | Status | Server
                                      |
+--------------------------------------+-------------------------------+--------
+--------------------------------------+
| 397e713c-b95b-4186-ad46-6126863ea0a9 | cirros-0.3.2-x86_64-uec       | ACTIVE |
                                      |
| df430cc2-3406-4061-b635-a51c16e488ac | cirros-0.3.2-x86_64-uec-kernel | ACTIVE |
                                      |
| 3cf852bd-2332-48f4-9ae4-7d926d50945e | cirros-0.3.2-x86_64-uec-ramdisk | ACTIVE |
                                      |
| 7e5142af-1253-4634-bcc6-89482c5f2e8a | myCirrosImage                 | ACTIVE | 84c6e57d-
a6b1-44b6-81eb-fcb36afd31b5 |
| 89bcd424-9d15-4723-95ec-61540e8a1979 | mysnapshot                    | ACTIVE | f51ebd07-
c33d-4951-8722-1df6aa8afaa4 |
+--------------------------------------+-------------------------------+--------
+--------------------------------------+
```

2. List the availability zones, and note the ID of the availability zone in which you want to create your volume:

```
$ cinder availability-zone-list
```

```
+------+-----------+
| Name |   Status  |
+------+-----------+
| nova | available |
+------+-----------+
```

3. Create a volume with 8 GB of space, and specify the availability zone and image:

```
$ cinder create 8 --display-name my-new-volume --image-id 397e713c-
b95b-4186-ad46-6126863ea0a9 --availability-zone nova
```

```
+---------------------+--------------------------------------+
|       Property      |                Value                 |
+---------------------+--------------------------------------+
|     attachments     |                  []                  |
|  availability_zone  |                 nova                 |
|       bootable      |                false                 |
|      created_at     |      2013-07-25T17:02:12.472269       |
| display_description |                 None                 |
|     display_name    |             my-new-volume            |
|          id         | 573e024d-5235-49ce-8332-be1576d323f8 |
|       image_id      | 397e713c-b95b-4186-ad46-6126863ea0a9 |
|       metadata      |                  {}                  |
|         size        |                  8                   |
|     snapshot_id     |                 None                 |
|     source_volid    |                 None                 |
|        status       |               creating               |
|     volume_type     |                 None                 |
+---------------------+--------------------------------------+
```

4.   To verify that your volume was created successfully, list the available volumes:

```
$ cinder list
```

```
+--------------------------------------+-----------+-----------------+------+-------------
+----------+-------------+
|                  ID                  |  Status   |  Display Name   | Size | Volume Type |
 Bootable | Attached to |
+--------------------------------------+-----------+-----------------+------+-------------
+----------+-------------+
| 573e024d-5235-49ce-8332-be1576d323f8 | available |  my-new-volume  |  8   |     None    |
 true     |             |
| bd7cf584-45de-44e3-bf7f-f7b50bf235e3 | available | my-bootable-vol |  8   |     None    |
 true     |             |
+--------------------------------------+-----------+-----------------+------+-------------
+----------+-------------+
```

If your volume was created successfully, its status is `available`. If its status is `error`, you might have exceeded your quota.

# Attach a volume to an instance

1.   Attach your volume to a server, specifying the server ID and the volume ID:

```
$ nova volume-attach 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5
 573e024d-5235-49ce-8332-be1576d323f8 /dev/vdb
```

```
+----------+--------------------------------------+
| Property | Value                                |
+----------+--------------------------------------+
| device   | /dev/vdb                             |
| serverId | 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5 |
| id       | 573e024d-5235-49ce-8332-be1576d323f8 |
| volumeId | 573e024d-5235-49ce-8332-be1576d323f8 |
+----------+--------------------------------------+
```

Note the ID of your volume.

2.   Show information for your volume:

```
$ cinder show 573e024d-5235-49ce-8332-be1576d323f8
```

The output shows that the volume is attached to the server with ID `84c6e57d-a6b1-44b6-81eb-fcb36afd31b5`, is in the nova availability zone, and is bootable.

```
+----------------------------
+----------------------------------------------------------------------------------------------------------
+
|          Property        |
                                    Value
                                       |
+----------------------------
+----------------------------------------------------------------------------------------------------------
+
|        attachments       |                 [{u'device': u'/dev/vdb', u'server_id': u'84c6e57d-
a6b1-44b6-81eb-fcb36afd31b5', u'id': u'573e024d-5235-49ce-8332-be1576d323f8', u'volume_id':
 u'573e024d-5235-49ce-8332-be1576d323f8'}]             |
|      availability_zone   |
                                    nova
                                       |
|          bootable        |
                                    true
                                       |
|         created_at       |
                           2013-07-25T17:02:12.000000
                                       |
|     display_description  |
                                    None
                                       |
```

```
|       display_name        |
                          my-new-volume
                                            |
|           id              |
                 573e024d-5235-49ce-8332-be1576d323f8
                                            |
|        metadata           |
                               {}
                                            |
|    os-vol-host-attr:host  |
                            devstack
                                            |
| os-vol-tenant-attr:tenant_id |
                     66265572db174a7aa66eba661f58eb9e
                                            |
|          size             |
                                8
                                            |
|       snapshot_id         |
                              None
                                            |
|       source_volid        |
                              None
                                            |
|         status            |
                             in-use
                                            |
|   volume_image_metadata   | {u'kernel_id': u'df430cc2-3406-4061-b635-
a51c16e488ac', u'image_id': u'397e713c-b95b-4186-ad46-6126863ea0a9', u'ramdisk_id':
 u'3cf852bd-2332-48f4-9ae4-7d926d50945e', u'image_name': u'cirros-0.3.2-x86_64-uec'} |
|       volume_type         |
                              None
                                            |
+---------------------------
+-----------------------------------------------------------------------------------------------
+
```

# Resize a volume

1. To resize your volume, you must first detach it from the server.

   To detach the volume from your server, pass the server ID and volume ID to the following command:

   ```
   $ nova volume-detach 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5
    573e024d-5235-49ce-8332-be1576d323f8
   ```

   The **volume-detach** command does not return any output.

2. List volumes:

   ```
   $ cinder list

   +------------------------------------+-----------+----------------+------+-------------
   +----------+-------------+
   |                ID                  |  Status   |  Display Name  | Size | Volume Type |
   Bootable | Attached to |
   +------------------------------------+-----------+----------------+------+-------------
   +----------+-------------+
   | 573e024d-5235-49ce-8332-be1576d323f8 | available |  my-new-volume | 8    |     None    |
   true    |             |
   | bd7cf584-45de-44e3-bf7f-f7b50bf235e3 | available | my-bootable-vol | 8    |     None    |
   true    |             |
   +------------------------------------+-----------+----------------+------+-------------
   +----------+-------------+
   ```

   Note that the volume is now available.

3. Resize the volume by passing the volume ID and the new size (a value greater than the old one) as parameters:

```
$ cinder extend 573e024d-5235-49ce-8332-be1576d323f8 10
```

The **extend** command does not return any output.

# Delete a volume

1. To delete your volume, you must first detach it from the server.

   To detach the volume from your server and check for the list of existing volumes, see steps 1 and 2 in .

2. Delete the volume using either the volume name or ID:

   ```
   $ cinder delete my-new-volume
   ```

   The delete command does not return any output.

3. List the volumes again, and note that the status of your volume is `deleting`:

   ```
   $ cinder list

   +------------------------------------+-----------+----------------+------+-------------
   +----------+-------------+
   |                 ID                 |  Status   |  Display Name  | Size | Volume Type |
    Bootable | Attached to |
   +------------------------------------+-----------+----------------+------+-------------
   +----------+-------------+
   | 573e024d-5235-49ce-8332-be1576d323f8 | deleting  |  my-new-volume |  8   |     None    |
    true     |             |
   | bd7cf584-45de-44e3-bf7f-f7b50bf235e3 | available | my-bootable-vol|  8   |     None    |
    true     |             |
   +------------------------------------+-----------+----------------+------+-------------
   +----------+-------------+
   ```

   When the volume is fully deleted, it disappears from the list of volumes:

   ```
   $ cinder list

   +------------------------------------+-----------+----------------+------+-------------
   +----------+-------------+
   |                 ID                 |  Status   |  Display Name  | Size | Volume Type |
    Bootable | Attached to |
   +------------------------------------+-----------+----------------+------+-------------
   +----------+-------------+
   | bd7cf584-45de-44e3-bf7f-f7b50bf235e3 | available | my-bootable-vol|  8   |     None    |
    true     |             |
   +------------------------------------+-----------+----------------+------+-------------
   +----------+-------------+
   ```

# Transfer a volume

You can transfer a volume from one owner to another by using the **cinder transfer\*** commands. The volume donor, or original owner, creates a transfer request and sends the created transfer ID and authorization key to the volume recipient. The volume recipient, or new owner, accepts the transfer by using the ID and key.

## Note

The procedure for volume transfer is intended for tenants (both the volume donor and recipient) within the same cloud.

Use cases include:

- Create a custom bootable volume or a volume with a large data set and transfer it to a customer.

- For bulk import of data to the cloud, the data ingress system creates a new Block Storage volume, copies data from the physical device, and transfers device ownership to the end user.

# Create a volume transfer request

1. While logged in as the volume donor, list the available volumes:

   ```
   $ cinder list
   ```

   ```
   +------------------------------------+-----------+--------------+------
   -------------+----------+-------------+
   |                 ID                 |  Status   | Display Name | Size |
    Volume Type | Bootable | Attached to |
   +------------------------------------+-----------+--------------+------
   -------------+----------+-------------+
   | 72bfce9f-cacf-477a-a092-bf57a7712165 |   error   |     None     |  1   |
       None    |  false   |             |
   | a1cdace0-08e4-4dc7-b9dc-457e9bcfe25f | available |     None     |  1   |
       None    |  false   |             |
   +------------------------------------+-----------+--------------+------
   -------------+----------+-------------+
   ```

2. As the volume donor, request a volume transfer authorization code for a specific volume:

   ```
   $ cinder transfer-create volumeID
   ```

   The volume must be in an `available` state or the request will be denied. If the transfer request is valid in the database (that is, it has not expired or been deleted), the volume is placed in an `awaiting transfer` state. For example:

   ```
   $ cinder transfer-create a1cdace0-08e4-4dc7-b9dc-457e9bcfe25f
   ```

   The output shows the volume transfer ID in the `id` row and the authorization key.

   ```
   +------------+--------------------------------------+
   |  Property  |                Value                 |
   +------------+--------------------------------------+
   |  auth_key  |           b2c8e585cbc68a80           |
   | created_at |      2013-10-14T15:20:10.121458      |
   |     id     | 6e4e9aa4-bed5-4f94-8f76-df43232f44dc |
   |    name    |                 None                 |
   | volume_id  | a1cdace0-08e4-4dc7-b9dc-457e9bcfe25f |
   +------------+--------------------------------------+
   ```

   ### Note

   Optionally, you can specify a name for the transfer by using the `--display-name` `displayName` parameter.

3. Send the volume transfer ID and authorization key to the new owner (for example, by email).

4. View pending transfers:

```
$ cinder transfer-list
```

```
+--------------------------------------
+---------------------------------------+------+
|                   ID                 |              VolumeID
      |  Name  |
+--------------------------------------
+---------------------------------------+------+
| 6e4e9aa4-bed5-4f94-8f76-df43232f44dc | a1cdace0-08e4-4dc7-
b9dc-457e9bcfe25f | None |
+--------------------------------------
+---------------------------------------+------+
```

5. After the volume recipient, or new owner, accepts the transfer, you can see that the transfer is no longer available:

```
$ cinder transfer-list
```

```
+----+-----------+------+
| ID | Volume ID | Name |
+----+-----------+------+
+----+-----------+------+
```

# Accept a volume transfer request

1. As the volume recipient, you must first obtain the transfer ID and authorization key from the original owner.

2. Display the transfer request details by using the ID:

```
$ cinder transfer-show transferID
```

For example:

```
$ cinder transfer-show 6e4e9aa4-bed5-4f94-8f76-df43232f44dc
```

```
+------------+--------------------------------------+
|  Property  |                 Value                |
+------------+--------------------------------------+
| created_at |      2013-10-14T15:20:10.000000      |
|     id     | 6e4e9aa4-bed5-4f94-8f76-df43232f44dc |
|    name    |                 None                 |
| volume_id  | a1cdace0-08e4-4dc7-b9dc-457e9bcfe25f |
+-----------+---------------------------------------+
```

3. Accept the request:

```
$ cinder transfer-accept transferID authKey
```

For example:

```
$ cinder transfer-accept 6e4e9aa4-bed5-4f94-8f76-df43232f44dc
 b2c8e585cbc68a80
```

```
+-----------+--------------------------------------+
|  Property |                 Value                |
+-----------+--------------------------------------+
|     id    | 6e4e9aa4-bed5-4f94-8f76-df43232f44dc |
|    name   |                 None                 |
```

```
| volume_id | a1cdace0-08e4-4dc7-b9dc-457e9bcfe25f |
+-----------+--------------------------------------+
```

> **Note**
>
> If you do not have a sufficient quota for the transfer, the transfer is re-
> fused.

## Delete a volume transfer

1. List available volumes and their statuses:

```
$ cinder list
```

```
+--------------------------------------+------------------+--------------
+------+-------------+----------+-------------+
|                  ID                  |      Status      | Display Name
 | Size | Volume Type | Bootable | Attached to |
+--------------------------------------+------------------+--------------
+------+-------------+----------+-------------+
| 72bfce9f-cacf-477a-a092-bf57a7712165 |      error       |     None
 |  1   |     None    |  false   |             |
| a1cdace0-08e4-4dc7-b9dc-457e9bcfe25f | awaiting-transfer |     None
 |  1   |     None    |  false   |             |
+--------------------------------------+------------------+--------------
+------+-------------+----------+-------------+
```

2. Find the matching transfer ID:

```
$ cinder transfer-list
```

```
+--------------------------------------
+--------------------------------------+------+
|                  ID                  |               VolumeID
     | Name |
+--------------------------------------
+--------------------------------------+------+
| a6da6888-7cdf-4291-9c08-8c1f22426b8a | a1cdace0-08e4-4dc7-
b9dc-457e9bcfe25f | None |
+--------------------------------------
+--------------------------------------+------+
```

3. Delete the volume:

```
$ cinder transfer-delete transferID
```

For example:

```
$ cinder transfer-delete a6da6888-7cdf-4291-9c08-8c1f22426b8a
```

4. Verify that transfer list is now empty and that the volume is again available for trans-
fer:

```
$ cinder transfer-list
```

```
+----+-----------+------+
| ID | Volume ID | Name |
+----+-----------+------+
+----+-----------+------+
```

```
$ cinder list
```

```
+-------------------------------------+-----------+--------------+------
-------------+----------+-------------+
|                 ID                  |  Status   | Display Name | Size |
  Volume Type | Bootable | Attached to |
+-------------------------------------+-----------+--------------+------
-------------+----------+-------------+
| 72bfce9f-cacf-477a-a092-bf57a7712165 |  error    |     None     |  1   |
      None    |  false   |             |
| a1cdace0-08e4-4dc7-b9dc-457e9bcfe25f | available |     None     |  1   |
      None    |  false   |             |
+-------------------------------------+-----------+--------------+------
-------------+----------+-------------+
```

# Set a volume to read-only access

To give multiple users shared, secure access to the same data, you can set a volume to read-only access.

Run the following command to set a volume to read-only access:

```
$ cinder readonly-mode-update VOLUME BOOLEAN
```

*VOLUME* is the ID of the target volume and *BOOLEAN* is a flag that enables read-only or read/write access to the volume.

The following values for *BOOLEAN* are valid:

• `true`. Sets the read-only flag in the volume. When you attach the volume to an instance, the instance checks for this flag to determine whether to restrict volume access to read-only.

• `false`. Sets the volume to read/write access.

# Create and manage databases

The Database service provides scalable and reliable cloud provisioning functionality for both relational and non-relational database engines. Users can quickly and easily use database features without the burden of handling complex administrative tasks.

# Create and access a database

Assume that you have installed the Database service and populated your data store with images for the type and versions of databases that you want, and that you can create and access a database.

This example shows you how to create and access a MySQL 5.5 database.

### To create and access a database

1. **Determine which flavor to use for your database**

   When you create a database instance, you must specify a nova flavor. The flavor indicates various characteristics of the instance, such as RAM, root volume size, and so on. The default nova flavors are not sufficient to create database instances. You might need to create or obtain some new nova flavors that work for databases.

   The first step is to list flavors by using the nova **flavor-list** command.

   Here are the default flavors, although you may have additional custom flavors in your environment:

   ```
   $ nova flavor-list
   +----+-----------+-----------+------+-----------+------+-------
   +-------------+-----------+
   | ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs |
    RXTX_Factor | Is_Public |
   +----+-----------+-----------+------+-----------+------+-------
   +-------------+-----------+
   | 1  | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0
      | True      |
   | 2  | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0
      | True      |
   | 3  | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0
      | True      |
   | 4  | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0
      | True      |
   | 5  | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0
      | True      |
   +----+-----------+-----------+------+-----------+------+-------
   +-------------+-----------+
   ```

   Now take a look at the minimum requirements for various database instances:

   | Database | RAM (MB) | Disk (GB) | VCPUs |
   |----------|----------|-----------|-------|
   | MySQL | 512 | 5 | 1 |
   | Cassandra | 2048 | 5 | 1 |
   | MongoDB | 1024 | 5 | 1 |

| Database | RAM (MB) | Disk (GB) | VCPUs |
|----------|----------|-----------|-------|
| Redis    | 512      | 5         | 1     |

- If you have a custom flavor that meets the needs of the database that you want to create, proceed to Step 2 [118] and use that flavor.

- If your environment does not have a suitable flavor, an administrative user must create a custom flavor by using the nova **flavor-create** command.

**MySQL example.** This example creates a flavor that you can use with a MySQL database. This example has the following attributes:

- Flavor name: `mysql_minimum`

- Flavor ID: You must use an ID that is not already in use. In this example, IDs 1 through 5 are in use, so use ID `6`.

- RAM: `512`

- Root volume size in GB: `5`

- Virtual CPUs: `1`

```
$ nova flavor-create mysql-minimum 6 512 5 1
+----+--------------+-----------+------+-----------+------+-------
+-------------+-----------+
| ID | Name         | Memory_MB | Disk | Ephemeral | Swap | VCPUs |
 RXTX_Factor | Is_Public |
+----+--------------+-----------+------+-----------+------+-------
+-------------+-----------+
| 6  | mysql-minimum | 512      | 5    | 0         |      | 1     | 1.0
        | True      |
+----+--------------+-----------+------+-----------+------+-------
+-------------+-----------+
```

2. **Create a database instance**

   This example creates a database instance with the following characteristics:

   - Name of the instance: `mysql_instance_1`

   - Database flavor: `6`

   In addition, this command specifies these options for the instance:

   - A volume size of `5` (5 GB).

   - The `myDB` database.

   - The database is based on the `mysql` data store and the `mysql-5.5` datastore_version.

   - The `userA` user with the `password` password.

```
$ trove create mysql_instance_1 6 --size 5 --databases myDB \
    --users userA:password --datastore_version mysql-5.5 \
```

```
    --datastore mysql
+------------------
+--------------------------------------------------------------------------------
t--------------------------------------------------------------------------------
+
|      Property    |
                                              Value

 |
+------------------
+--------------------------------------------------------------------------------
+
|      created     |
                                     2014-05-29T21:26:21

 |
|     datastore    |
                          {u'version': u'mysql-5.5', u'type': u'mysql'}

 |
| datastore_version |
                                        mysql-5.5

 |
|       flavor      | {u'id': u'6', u'links': [{u'href': u'https://
controller:8779/v1.0/46d0bc4fc32e4b9e8520f8fc62199f58/flavors/6', u'rel':
 u'self'}, {u'href': u'https://controller:8779/flavors/6', u'rel':
 u'bookmark'}]} |
|         id        |
                              5599dad6-731e-44df-bb60-488da3da9cfe

 |
|        name       |
                                     mysql_instance_1

 |
|       status      |
                                          BUILD

 |
|       updated     |
                                     2014-05-29T21:26:21

 |
|       volume      |
                                        {u'size': 5}

 |
+------------------
+--------------------------------------------------------------------------------
+
```

3.  **Get the IP address of the database instance**

    First, use the trove **list** command to list all instances and their IDs:

    ```
    $ trove list
    +-----------------------------------+------------------+-----------
    +------------------+--------+-----------+------+
    ```

```
|                    id                    |       name       |  datastore |
 datastore_version | status | flavor_id | size |
+------------------------------------------+------------------+-----------
+-------------------+--------+-----------+------+
| 5599dad6-731e-44df-bb60-488da3da9cfe | mysql_instance_1 |    mysql    |
   mysql-5.5        | BUILD  |     6     |  5   |
+------------------------------------------+------------------+-----------
+-------------------+--------+-----------+------+
```

This command returns the instance ID of your new instance.

You can now pass in the instance ID with the trove **show** command to get the IP address of the instance. In this example, replace *INSTANCE_ID* with 5599dad6-731e-44df-bb60-488da3da9cfe.

```
$ trove show INSTANCE_ID
+-------------------+--------------------------------------+
|     Property      |                Value                 |
+-------------------+--------------------------------------+
|      created      |          2014-05-29T21:26:21         |
|     datastore     |                mysql                 |
| datastore_version |              mysql-5.5               |
|      flavor       |                  6                   |
|        id         | 5599dad6-731e-44df-bb60-488da3da9cfe |
|        ip         |             172.16.200.2             |
|       name        |           mysql_instance_1           |
|      status       |                BUILD                 |
|      updated      |          2014-05-29T21:26:54         |
|      volume       |                  5                   |
+-------------------+--------------------------------------+
```

This command returns the IP address of the database instance.

4.  **Access the new database**

    You can now access the new database you just created (myDB) by using typical database access commands. In this MySQL example, replace *IP_ADDRESS* with 172.16.200.2.

    ```
    $ mysql -u userA -ppassword -h IP_ADDRESS myDB
    ```

# Backup and restore a database

You can use Database services to backup a database and store the backup artifact in the Object Storage module. Later on, if the original database is damaged, you can use the backup artifact to restore the database. The restore process creates a database instance.

This example shows you how to back up and restore a MySQL database.

## To backup and restore a database

1.  **Backup the database instance**

    As background, assume that you have created a database instance with the following characteristics:

    • Name of the database instance: guest1

- Flavor ID: `10`

- Root volume size: `2`

- Databases: `db1` and `db2`

- Users: The `user1` user with the `password` password

First, get the ID of the `guest1` database instance by using the trove **list** command:

```
$ trove list
+-------------------------------------+--------+-----------
+------------------+--------+-----------+------+
|                id                   |  name  | datastore |
 datastore_version | status | flavor_id | size |
+-------------------------------------+--------+-----------
+------------------+--------+-----------+------+
| 97b4b853-80f6-414f-ba6f-c6f455a79ae6 | guest1 |   mysql   |    mysql-5.
5     | ACTIVE |    10     |  2   |
+-------------------------------------+--------+-----------
+------------------+--------+-----------+------+
```

Back up the database instance by using the trove **backup-create** command. In this example, the backup is called `backup1`. In this example, replace *INSTANCE_ID* with `97b4b853-80f6-414f-ba6f-c6f455a79ae6`:

> **Note**
>
> This command syntax pertains only to python-troveclient version 1.0.6 and later. Earlier versions require you to pass in the backup name as the first argument.

```
$ trove backup-create INSTANCE_ID backup1
+-------------+--------------------------------------+
|  Property   |                Value                 |
+-------------+--------------------------------------+
|   created   |         2014-03-18T17:09:07          |
| description |                 None                 |
|     id      | 8af30763-61fd-4aab-8fe8-57d528911138 |
| instance_id | 97b4b853-80f6-414f-ba6f-c6f455a79ae6 |
| locationRef |                 None                 |
|    name     |               backup1                |
|  parent_id  |                 None                 |
|    size     |                 None                 |
|   status    |                 NEW                  |
|   updated   |         2014-03-18T17:09:07          |
+-------------+--------------------------------------+
```

Note that the command returns both the ID of the original instance (`instance_id`) and the ID of the backup artifact (`id`).

Later on, use the trove **backup-list** command to get this information:

```
$ trove backup-list
+-------------------------------------
+-------------------------------------+--------+-----------+-----------
+--------------------+
```

```
|                    id                    |              instance_id
     |   name   |   status   | parent_id |        updated        |
+-----------------------------------
+-----------------------------------------+--------+----------+-----------
+-------------------+
| 8af30763-61fd-4aab-8fe8-57d528911138 | 97b4b853-80f6-414f-ba6f-
c6f455a79ae6 | backup1 | COMPLETED |    None    | 2014-03-18T17:09:11 |
+-----------------------------------
+-----------------------------------------+--------+----------+-----------
+-------------------+
```

You can get additional information about the backup by using the
trove **backup-show** command and passing in the *BACKUP_ID*, which is
8af30763-61fd-4aab-8fe8-57d528911138.

```
$ trove backup-show BACKUP_ID
+-------------
+-----------------------------------------------------------------------------------
+
|   Property  |
     Value
 |
+-------------
+-----------------------------------------------------------------------------------
+
|   created   |
 2014-03-18T17:09:07
    |
| description |
      None
 |
|     id      |
 8af30763-61fd-4aab-8fe8-57d528911138
             |
| instance_id |
 97b4b853-80f6-414f-ba6f-c6f455a79ae6
             |
| locationRef | http://10.0.0.1:8080/v1/
AUTH_626734041baa4254ae316de52a20b390/database_backups/
8af30763-61fd-4aab-8fe8-57d528911138.xbstream.gz.enc |
|     name    |
     backup1
 |
|  parent_id  |
      None
 |
|     size    |
     0.17
 |
|    status   |
    COMPLETED
 |
|   updated   |
 2014-03-18T17:09:11
    |
+-------------
+-----------------------------------------------------------------------------------
+
```

2. **Restore a database instance**

Now assume that your guest1 database instance is damaged and you need to restore it. In this example, you use the trove **create** command to create a new database instance called guest2.

- You specify that the new guest2 instance has the same flavor (10) and the same root volume size (2) as the original guest1 instance.

- You use the --backup argument to indicate that this new instance is based on the backup artifact identified by *BACKUP_ID*. In this example, replace *BACKUP_ID* with 8af30763-61fd-4aab-8fe8-57d528911138.

```
$ trove create guest2 10 --size 2 --backup BACKUP_ID
+------------------
+----------------------------------------------------------------------------
+
|      Property     |
                                                Value

         |
+------------------
+----------------------------------------------------------------------------
+
|      created      |
                                       2014-03-18T17:12:03

         |
|     datastore     |
                              {u'version': u'mysql-5.5', u'type': u'mysql'}

         |
| datastore_version |
                                          mysql-5.5

         |
|       flavor      | {u'id': u'10', u'links': [{u'href': u'https://10.
125.1.135:8779/v1.0/626734041baa4254ae316de52a20b390/flavors/10', u'rel':
 u'self'}, {u'href': u'https://10.125.1.135:8779/flavors/10', u'rel':
 u'bookmark'}]} |
|        id         |
                                ac7a2b35-a9b4-4ff6-beac-a1bcee86d04b

         |
|       name        |
                                          guest2

         |
|      status       |
                                          BUILD

         |
|      updated      |
                                       2014-03-18T17:12:03

         |
|      volume       |
                                        {u'size': 2}
```

```
       |
+------------------
+-------------------------------------------------------------------------------------------
+
```

3. **Verify backup**

   Now check that the new `guest2` instance has the same characteristics as the original
   `guest1` instance.

   Start by getting the ID of the new `guest2` instance.

   ```
   $ trove list
   +--------------------------------------+--------+-----------
   +------------------+--------+-----------+------+
   |                  id                  |  name  | datastore |
    datastore_version | status | flavor_id | size |
   +--------------------------------------+--------+-----------
   +------------------+--------+-----------+------+
   | 97b4b853-80f6-414f-ba6f-c6f455a79ae6 | guest1 |   mysql   |   mysql-5.
   5     | ACTIVE |    10     |  2   |
   | ac7a2b35-a9b4-4ff6-beac-a1bcee86d04b | guest2 |   mysql   |   mysql-5.
   5     | ACTIVE |    10     |  2   |
   +--------------------------------------+--------+-----------
   +------------------+--------+-----------+------+
   ```

   Use the trove **show** command to display information about the new guest2 in-
   stance. Pass in guest2's *INSTANCE_ID*, which is `ac7a2b35-a9b4-4ff6-beac-`
   `a1bcee86d04b`.

   ```
   $ trove show INSTANCE_ID
   +-------------------+--------------------------------------+
   |     Property      |                Value                 |
   +-------------------+--------------------------------------+
   |      created      |          2014-03-18T17:12:03         |
   |     datastore     |                mysql                 |
   | datastore_version |               mysql-5.5              |
   |       flavor      |                  10                  |
   |        id         | ac7a2b35-a9b4-4ff6-beac-a1bcee86d04b |
   |        ip         |               10.0.0.3               |
   |       name        |                guest2                |
   |      status       |                ACTIVE                |
   |      updated      |          2014-03-18T17:12:06         |
   |      volume       |                  2                   |
   |    volume_used    |                 0.18                 |
   +-------------------+--------------------------------------+
   ```

   Note that the data store, flavor ID, and volume size have the same values as in the
   original `guest1` instance.

   Use the trove **database-list** command to check that the original databases (`db1` and
   `db2`) are present on the restored instance.

   ```
   $ trove database-list INSTANCE_ID
   +--------------------+
   |        name        |
   +--------------------+
   |        db1         |
   ```

```
|         db2        |
| performance_schema |
|         test       |
+--------------------+
```

Use the trove **user-list** command to check that the original user (`user1`) is present on the restored instance.

```
$ trove user-list INSTANCE_ID
+--------+------+-----------+
|  name  | host | databases |
+--------+------+-----------+
| user1  |  %   |  db1, db2 |
+--------+------+-----------+
```

4. **Notify users**

   Tell the users who were accessing the now-disabled `guest1` database instance that they can now access `guest2`. Provide them with `guest2`'s name, IP address, and any other information they might need. (You can get this information by using the trove **show** command.)

5. **Clean up**

   At this point, you might want to delete the disabled `guest1` instance, by using the trove **delete** command.

```
$ trove delete INSTANCE_ID
```

# Use incremental backups

Incremental backups let you chain together a series of backups. You start with a regular backup. Then, when you want to create a subsequent incremental backup, you specify the parent backup.

Restoring a database instance from an incremental backup is the same as creating a database instance from a regular backup—the Database service handles the complexities of applying the chain of incremental backups.

This example shows you how to use incremental backups with a MySQL database.

**Assumptions.** Assume that you have created a regular backup for the following database instance:

- Instance name: `guest1`

- ID of the instance (`INSTANCE_ID`): `792a6a56-278f-4a01-9997-d997fa126370`

- ID of the regular backup artifact (`BACKUP_ID`): `6dc3a9b7-1f3e-4954-8582-3f2e4942cddd`

## To create and use incremental backups

1. **Create your first incremental backup**

   Use the trove **backup-create** command and specify:

- The *INSTANCE_ID* of the database instance you are doing the incremental backup for (in this example, `792a6a56-278f-4a01-9997-d997fa126370`)

- The name of the incremental backup you are creating: `backup1.1`

- The *BACKUP_ID* of the parent backup. In this case, the parent is the regular backup, with an ID of `6dc3a9b7-1f3e-4954-8582-3f2e4942cddd`

```
$ trove backup-create INSTANCE_ID backup1.1  --parent BACKUP_ID
+-------------+-------------------------------------+
|  Property   |                Value                |
+-------------+-------------------------------------+
|   created   |         2014-03-19T14:09:13         |
| description |                None                 |
|     id      | 1d474981-a006-4f62-b25f-43d7b8a7097e |
| instance_id | 792a6a56-278f-4a01-9997-d997fa126370 |
| locationRef |                None                 |
|    name     |              backup1.1              |
|  parent_id  | 6dc3a9b7-1f3e-4954-8582-3f2e4942cddd |
|    size     |                None                 |
|   status    |                NEW                  |
|   updated   |         2014-03-19T14:09:13         |
+-------------+-------------------------------------+
```

Note that this command returns both the ID of the database instance you are incrementally backing up (`instance_id`) and a new ID for the new incremental backup artifact you just created (`id`).

2. **Create your second incremental backup**

The name of your second incremental backup is `backup1.2`. This time, when you specify the parent, pass in the ID of the incremental backup you just created in the previous step (`backup1.1`). In this example, it is `1d474981-a006-4f62-b25f-43d7b8a7097e`.

```
$ trove backup-create INSTANCE_ID  backup1.2   --parent BACKUP_ID
+-------------+-------------------------------------+
|  Property   |                Value                |
+-------------+-------------------------------------+
|   created   |         2014-03-19T14:09:13         |
| description |                None                 |
|     id      | bb84a240-668e-49b5-861e-6a98b67e7a1f |
| instance_id | 792a6a56-278f-4a01-9997-d997fa126370 |
| locationRef |                None                 |
|    name     |              backup1.2              |
|  parent_id  | 1d474981-a006-4f62-b25f-43d7b8a7097e |
|    size     |                None                 |
|   status    |                NEW                  |
|   updated   |         2014-03-19T14:09:13         |
+-------------+-------------------------------------+
```

3. **Restore using incremental backups**

Now assume that your `guest1` database instance is damaged and you need to restore it from your incremental backups. In this example, you use the trove **create** command to create a new database instance called `guest2`.

To incorporate your incremental backups, you simply use the `--backup` parameter to pass in the *BACKUP_ID* of your most recent incremental backup. The Database service handles the complexities of applying the chain of all previous incremental backups.

```
$ trove create guest2 10 --size 1 --backup BACKUP_ID
+------------------
+--------------------------------------------------------------+
|      Property     |                      Value
      |
+------------------
+--------------------------------------------------------------+
|      created      |                 2014-03-19T14:10:56
      |
|     datastore     |          {u'version': u'mysql-5.5', u'type':
u'mysql'}      |
| datastore_version |                      mysql-5.5
      |
|      flavor       | {u'id': u'10', u'links':
      |
|                   | [{u'href': u'https://10.125.1.135:8779/v1.0/
      |
|                   |  626734041baa4254ae316de52a20b390/flavors/10',
u'rel':     |
|                   |  u'self'}, {u'href': u'https://10.125.1.135:8779/
      |
|                   |  flavors/10', u'rel': u'bookmark'}]}
      |
|        id         |          a3680953-eea9-4cf2-918b-5b8e49d7e1b3
      |
|       name        |                      guest2
      |
|      status       |                      BUILD
      |
|      updated      |                 2014-03-19T14:10:56
      |
|      volume       |                   {u'size': 1}
      |
+------------------
+--------------------------------------------------------------+
```

# Manage database configuration

You can manage database configuration tasks by using configuration groups. Configuration groups let you set configuration options, in bulk, on one or more databases.

This example assumes you have [created a MySQL database](#) and shows you how to use a configuration group to configure it. Although this example sets just one option on one database, you can use these same procedures to set multiple options on multiple database instances throughout your environment. This can provide significant time savings in managing your cloud.

## To bulk-configure a database or databases

1.  **List available options**

    First, determine which configuration options you can set. Different data store versions have different configuration options.

List the names and IDs of all available versions of the `mysql` data store:

```
$ trove datastore-version-list mysql
+--------------------------------------+-----------+
|                  id                  |   name    |
+--------------------------------------+-----------+
| eeb574ce-f49a-48b6-820d-b2959fcd38bb | mysql-5.5 |
+--------------------------------------+-----------+
```

Pass in the data store version ID with the trove **configuration-parameter-list** command to get the available options:

```
$ trove configuration-parameter-list DATASTORE_VERSION_ID
+-------------------------------+---------+---------+
---------------------+------------------+
|              name             |  type   |   min   |         max
   | restart_required |
+-------------------------------+---------+---------+
---------------------+------------------+
|    auto_increment_increment   | integer |    1    |        65535
   |      False       |
|     auto_increment_offset     | integer |    1    |        65535
   |      False       |
|           autocommit          | integer |    0    |          1
   |      False       |
|     bulk_insert_buffer_size   | integer |    0    |
 18446744073709547520 |      False       |
|      character_set_client     |  string |         |
   |      False       |
|   character_set_connection    |  string |         |
   |      False       |
|     character_set_database    |  string |         |
   |      False       |
|    character_set_filesystem   |  string |         |
   |      False       |
|     character_set_results     |  string |         |
   |      False       |
|      character_set_server     |  string |         |
   |      False       |
|      collation_connection     |  string |         |
   |      False       |
|      collation_database       |  string |         |
   |      False       |
|       collation_server        |  string |         |
   |      False       |
|        connect_timeout        | integer |    1    |        65535
   |      False       |
|        expire_logs_days       | integer |    1    |        65535
   |      False       |
|     innodb_buffer_pool_size   | integer |    0    |     68719476736
   |      True        |
|      innodb_file_per_table    | integer |    0    |          1
   |      True        |
| innodb_flush_log_at_trx_commit | integer |    0    |          2
   |      False       |
|      innodb_log_buffer_size   | integer | 1048576 |    4294967296
   |      True        |
|        innodb_open_files      | integer |   10    |    4294967296
   |      True        |
```

```
|    innodb_thread_concurrency    | integer |    0    |       1000
  |        False        |
|       interactive_timeout       | integer |    1    |       65535
  |        False        |
|         join_buffer_size        | integer |    0    |    4294967296
  |        False        |
|         key_buffer_size         | integer |    0    |    4294967296
  |        False        |
|          local_infile           | integer |    0    |         1
  |        False        |
|        max_allowed_packet       | integer |   1024  |    1073741824
  |        False        |
|        max_connect_errors       | integer |    1    |
18446744073709547520 |       False        |
|         max_connections         | integer |    1    |       65535
  |        False        |
|       max_user_connections      | integer |    1    |       100000
  |        False        |
|     myisam_sort_buffer_size     | integer |    4    |
18446744073709547520 |       False        |
|            server_id            | integer |    1    |       100000
  |        True         |
|         sort_buffer_size        | integer |  32768  |
18446744073709547520 |       False        |
|           sync_binlog           | integer |    0    |
18446744073709547520 |       False        |
|           wait_timeout          | integer |    1    |      31536000
  |        False        |
+-------------------------------+---------+---------
+--------------------+-----------------+
```

In this example, the **configuration-parameter-list** command returns a list of options that work with MySQL 5.5.

2.  **Create a configuration group**

    A configuration group contains a comma-separated list of key-value pairs. Each pair consists of a configuration option and its value.

    You can create a configuration group by using the trove **configuration-create** command. The general syntax for this command is:

    ```
    $ trove configuration-create NAME VALUES --datastore DATASTORE_NAME
    ```

    • *NAME*. The name you want to use for this group.

    • *VALUES*. The list of key-value pairs.

    • *DATASTORE_NAME*. The name of the associated data store.

    Set *VALUES* as a JSON dictionary, for example:

    ```
    {"myFirstKey" : "someString", "mySecondKey" : someInt}
    ```

    This example creates a configuration group called group1. group1 contains just one key and value pair, and this pair sets the sync_binlog option to 1.

    ```
    $ trove configuration-create group1 '{"sync_binlog" : 1}' --datastore
    mysql
    ```

```
+--------------------+------------------------------------+
|      Property      |               Value                |
+--------------------+------------------------------------+
| datastore_version_id | eeb574ce-f49a-48b6-820d-b2959fcd38bb |
|    description      |               None                 |
|         id         | 9a9ef3bc-079b-476a-9cbf-85aa64f898a5 |
|        name        |               group1               |
|       values       |          {"sync_binlog": 1}        |
+--------------------+------------------------------------+
```

3.  **Examine your existing configuration**

    Before you use the newly-created configuration group, look at how the `sync_binlog` option is configured on your database. Replace the following sample connection values with values that connect to your database:

    ```
    $ mysql -u user7 -ppassword -h 172.16.200.2 myDB7
    Welcome to the MySQL monitor. Commands end with ; or \g.
    ...
    mysql> show variables like 'sync_binlog';
    +---------------+-------+
    | Variable_name | Value |
    +---------------+-------+
    | sync_binlog   | 0     |
    +---------------+-------+
    ```

    As you can see, the `sync_binlog` option is currently set to `0` for the `myDB7` database.

4.  **Change the database configuration using a configuration group**

    You can change a database's configuration by attaching a configuration group to a database instance. You do this by using the trove **configuration-attach** command and passing in the ID of the database instance and the ID of the configuration group.

    Get the ID of the database instance:

    ```
    $ trove list
    +--------------------------------------+------------------+-----------
    ------------------+--------+-----------+------+
    |                  id                  |       name       | datastore |
     datastore_version | status | flavor_id | size |
    +--------------------------------------+------------------+-----------
    ------------------+--------+-----------+------+
    | 26a265dd-1c88-4333-b3ed-6b4e9e87ffbb | mysql_instance_7 |   mysql   |
       mysql-5.5       | ACTIVE |     6     |  5   |
    +--------------------------------------+------------------+-----------
    ------------------+--------+-----------+------+
    ```

    Get the ID of the configuration group:

    ```
    $ trove configuration-list
    +--------------------------------------+--------+-------------
    ------------------------------------+
    |                  id                  |  name  | description |
     datastore_version_id        |
    +--------------------------------------+--------+-------------
    ------------------------------------+
    | 9a9ef3bc-079b-476a-9cbf-85aa64f898a5 | group1 |     None    | eeb574ce-
    f49a-48b6-820d-b2959fcd38bb |
    ```

```
+--------------------------------------+--------+------------
+--------------------------------------+
```

Attach the configuration group to the database instance:

> **Note**
>
> This command syntax pertains only to python-troveclient version 1.0.6 and
> later. Earlier versions require you to pass in the configuration group ID as
> the first argument.

```
$ trove configuration-attach DB_INSTANCE_ID CONFIG_GROUP_ID
```

5. **Re-examine the database configuration**

   Display the sync_binlog setting again:

   ```
   mysql> show variables like 'sync_binlog';
   +---------------+-------+
   | Variable_name | Value |
   +---------------+-------+
   | sync_binlog   | 1     |
   +---------------+-------+
   ```

   As you can see, the sync_binlog option is now set to 1, as specified in the group1
   configuration group.

**Conclusion.** Using a configuration group to set a single option on a single database is
obviously a trivial example. However, configuration groups can provide major efficiencies
when you consider that:

• A configuration group can specify a large number of option values.

• You can apply a configuration group to hundreds or thousands of database instances in
your environment.

Used in this way, configuration groups let you modify your database cloud configuration,
on the fly, on a massive scale.

**Maintenance.** There are also a number of useful maintenance features for working with
configuration groups. You can:

• Disassociate a configuration group from a database instance, using the trove **configura-
tion-detach** command.

• Modify a configuration group on the fly, using the trove **configuration-patch** command.

• Find out what instances are using a configuration group, using the trove **configura-
tion-instances** command.

• Delete a configuration group, using the trove **configuration-delete** command. You might
want to do this if no instances use a group.

# Set up database replication

You can create a replica of an existing database instance. When you make subsequent changes to the original instance, the system automatically applies those changes to the replica.

- Replicas are read-only.

- When you create a replica, do not specify the `--users` or `--databases` options.

- You can choose a smaller volume or flavor for a replica than for the original, but the replica's volume must be big enough to hold the data snapshot from the original.

This example shows you how to replicate a MySQL database instance.

### To set up replication

1. **Get the instance ID**

   Get the ID of the original instance you want to replicate:

   ```
   $ trove list
   +-------------------------------------+-----------+-------------
   +------------------+--------+-----------+------+
   |                  id                 |   name    | datastore |
    datastore_version | status | flavor_id | size |
   +-------------------------------------+-----------+-----------
   +------------------+--------+-----------+------+
   | 97b4b853-80f6-414f-ba6f-c6f455a79ae6 |  base_1   |   mysql   |
    mysql-5.5     | ACTIVE |    10    | 2  |
   +-------------------------------------+---------+-------------
   +------------------+--------+-----------+------+
   ```

2. **Create the replica**

   Create a new instance that will be a replica of the original instance. You do this by passing in the `--replica_of` option with the trove **create** command. This example creates a replica called `replica_1`. `replica_1` is a replica of the original instance, `base_1`:

   ```
   $ trove create replica_1 6 --size=5 --datastore_version mysql-5.5 --
   datastore mysql --replica_of ID_OF_ORIGINAL_INSTANCE
   ```

3. **Verify replication status**

   Pass in `replica_1`'s instance ID with the trove **show** command to verify that the newly created `replica_1` instance is a replica of the original `base_1`. Note that the `replica_of` property is set to the ID of `base_1`.

   ```
   $ trove show INSTANCE_ID_OF_REPLICA_1
   +-------------------+-------------------------------------+
   | Property          | Value                               |
   +-------------------+-------------------------------------+
   | created           | 2014-09-16T11:16:49                 |
   | datastore         | mysql                               |
   | datastore_version | mysql-5.5                           |
   ```

```
| flavor           | 6                                    |
| id               | 49c6eff6-ef91-4eff-91c0-efbda7e83c38 |
| name             | replica_1                            |
| replica_of       | 97b4b853-80f6-414f-ba6f-c6f455a79ae6 |
| status           | BUILD                                |
| updated          | 2014-09-16T11:16:49                  |
| volume           | 5                                    |
+------------------+--------------------------------------+
```

Now pass in `base_1`'s instance ID with the trove **show** command to list the replica(s) associated with the original instance. Note that the `replicas` property is set to the ID of `replica_1`. If there are multiple replicas, they appear as a comma-separated list.

```
$ trove show INSTANCE_ID_OF_BASE_1
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| created          | 2014-09-16T11:04:56                  |
| datastore        | mysql                                |
| datastore_version| mysql-5.5                            |
| flavor           | 6                                    |
| id               | 97b4b853-80f6-414f-ba6f-c6f455a79ae6 |
| ip               | 172.16.200.2                         |
| name             | base_1                               |
| replicas         | 49c6eff6-ef91-4eff-91c0-efbda7e83c38 |
| status           | ACTIVE                               |
| updated          | 2014-09-16T11:05:06                  |
| volume           | 5                                    |
| volume_used      | 0.11                                 |
+------------------+--------------------------------------+
```

4.  **Detach the replica**

    If the original instance goes down, you can detach the replica. The replica becomes a standalone database instance. You can then take the new standalone instance and create a new replica of that instance.

    You detach a replica using the trove **detach-replica** command:

    ```
    $ trove detach-replica INSTANCE_ID_OF_REPLICA
    ```

# Set up database clustering

You can store data across multiple machines by setting up MongoDB sharded clusters.

Each cluster includes:

• One or more *shards*. Each shard consists of a three member replica set (three instances organized as a replica set).

• One or more *query routers*. A query router is the machine that your application actually connects to. This machine is responsible for communicating with the config server to figure out where the requested data is stored. It then accesses and returns the data from the appropriate shard(s).

• One or more *config servers*. Config servers store the metadata that links requested data with the shard that contains it.

This example shows you how to set up a MongoDB sharded cluster.

> **Note**
>
> **Before you begin.**    Make sure that:
>
> • The administrative user has registered a MongoDB datastore type and version.
>
> • The administrative user has created an appropriate flavor that meets the MongoDB minimum requirements.

### To set up clustering

1.  **Create a cluster**

    Create a cluster by using the trove **cluster-create** command. This command creates a one-shard cluster. Pass in:

    • The name of the cluster.

    • The name and version of the datastore you want to use.

    • The three instances you want to include in the replication set for the first shard. Specify each instance by using the `--instance` argument and the associated flavor ID and volume size. Use the same flavor ID and volume size for each instance. In this example, flavor `7` is a custom flavor that meets the MongoDB minimum requirements.

    ```
    $ trove cluster-create cluster1 mongodb "2.4" \
      --instance flavor_id=7,volume=2 --instance flavor_id=7,volume=2 \
      --instance flavor_id=7,volume=2
    +-------------------+-----------------------------------+
    | Property          | Value                             |
    +-------------------+-----------------------------------+
    | created           | 2014-08-16T01:46:51               |
    | datastore         | mongodb                           |
    | datastore_version | 2.4                               |
    | id                | aa6ef0f5-dbef-48cd-8952-573ad881e717 |
    | name              | cluster1                          |
    | task_description  | Building the initial cluster.     |
    | task_name         | BUILDING                          |
    | updated           | 2014-08-16T01:46:51               |
    +-------------------+-----------------------------------+
    ```

2.  **Display cluster information**

    Display information about a cluster by using the trove **cluster-show** command. Pass in the ID of the cluster.

    The cluster ID display when you first create a cluster. (If you need to find it later on, use the trove **cluster-list** command to list the names and IDs of all the clusters in your system.)

    ```
    $ trove cluster-show CLUSTER_ID
    +-------------------+-----------------------------------+
    ```

```
| Property          | Value                                |
+-------------------+--------------------------------------+
| created           | 2014-08-16T01:46:51                  |
| datastore         | mongodb                              |
| datastore_version | 2.4                                  |
| id                | aa6ef0f5-dbef-48cd-8952-573ad881e717 |
| ip                | 10.0.0.2                             |
| name              | cluster1                             |
| task_description  | No tasks for the cluster.            |
| task_name         | NONE                                 |
| updated           | 2014-08-16T01:59:33                  |
+-------------------+--------------------------------------+
```

### Note

**Your application connects to this IP address.**    The trove **cluster-show** command displays the IP address of the query router. This is the IP address your application uses to retrieve data from the database.

3. **List cluster instances**

   List the instances in a cluster by using the trove **cluster-instances** command.

```
$ trove cluster-instances CLUSTER_ID
+--------------------------------------+--------------+-----------
+------+
| ID                                   | Name         | Flavor ID | Size
 |
+--------------------------------------+--------------+-----------
+------+
| 45532fc4-661c-4030-8ca4-18f02aa2b337 | cluster1-rs1-1 | 7         |    2
 |
| 7458a98d-6f89-4dfd-bb61-5cf1dd65c121 | cluster1-rs1-2 | 7         |    2
 |
| b37634fb-e33c-4846-8fe8-cf2b2c95e731 | cluster1-rs1-3 | 7         |    2
 |
+--------------------------------------+--------------+-----------
+------+
```

   **Naming conventions for replication sets and instances.**    Note that the `Name` column displays an instance name that includes the replication set name. The replication set names and instance names are automatically generated, following these rules:

   • **Replication set name.**    This name consists of the cluster name, followed by the string -rs$n$, where $n$ is 1 for the first replication set you create, 2 for the second replication set, and so on. In this example, the cluster name is `cluster1`, and there is only one replication set, so the replication set name is `cluster1-rs1`.

   • **Instance name.**    This name consists of the replication set name followed by the string -$n$, where $n$ is 1 for the first instance in a replication set, 2 for the the second instance, and so on. In this example, the instance names are `cluster1-rs1-1`, `cluster1-rs1-2`, and `cluster1-rs1-3`.

4. **List clusters**

   List all the clusters in your system, using the trove **cluster-list** command.

```
$ trove cluster-list
+------------------------------------+----------+-----------
+------------------+-----------+
| ID                                 | Name     | Datastore | Datastore
 Version | Task Name |
+------------------------------------+----------+-----------
+------------------+-----------+
| aa6ef0f5-dbef-48cd-8952-573ad881e717 | cluster1 | mongodb   | 2.4
         | NONE      |
| b8829c2a-b03a-49d3-a5b1-21ec974223ee | cluster2 | mongodb   | 2.4
         | BUILDING  |
+------------------------------------+----------+-----------
+------------------+-----------+
```

5. **Delete a cluster**

   Delete a cluster, using the trove **cluster-delete** command.

```
$ trove cluster-delete CLUSTER_ID
```

# Query routers and config servers

Each cluster includes at least one query router and one config server. Query routers and config servers count against your quota. When you delete a cluster, the system deletes the associated query router(s) and config server(s).

# 3. OpenStack Python SDK

## Table of Contents

Use the OpenStack Python Software Development Kit (SDK) to write Python automation scripts that create and manage resources in your OpenStack cloud. The SDK implements Python bindings to the OpenStack API, which enables you to perform automation tasks in Python by making calls on Python objects rather than making REST calls directly. All Open-Stack command-line tools are implemented using the Python SDK.

You should also be familiar with:

• RESTful web services
• HTTP/1.1
• JSON and XML data serialization formats

## Install the OpenStack SDK

Each OpenStack project has its own Python library. These libraries are bundled with the command-line clients. For example, the Python bindings for the Compute API are bundled with the python-novaclient package.

For details about how to install the clients, see install the OpenStack command-line clients.

## Authenticate

When using the SDK, you must authenticate against an OpenStack endpoint before you can use OpenStack services. Each project uses a slightly different syntax for authentication.

You must typically authenticate against a specific version of a service. For example, a client might need to authenticate against Identity v2.0.

Python scripts that use the OpenStack SDK must have access to the credentials contained in the OpenStack RC file. Because credentials are sensitive information, do not include them in your scripts. This guide assumes that users source the `PROJECT-openrc.sh` file and access the credentials by using the environment variables in the Python scripts.

## Authenticate against an Identity endpoint

To authenticate against the Identity v2.0 endpoint, instantiate a keystoneclient.v_20.client.Client object:

```
from os import environ as env
import keystoneclient.v2_0.client as ksclient
keystone = ksclient.Client(auth_url=env['OS_AUTH_URL'],
                           username=env['OS_USERNAME'],
                           password=env['OS_PASSWORD'],
                           tenant_name=env['OS_TENANT_NAME'],
                           region_name=env['OS_REGION_NAME'])
```

After you instantiate a `Client` object, you can retrieve the token by accessing its `auth_token` attribute object:

```
import keystoneclient.v2_0.client as ksclient
keystone = ksclient.Client(...)
print keystone.auth_token
```

If the OpenStack cloud is configured to use public-key infrastructure (PKI) tokens, the Python script output looks something like this:

```
MIIQUQYJKoZIhvcNAQcCoIIQQjCCED4CAQExCTAHBgUrDgMCGjCCDqcGCSqGSIb3DQEHAaCCDpgE
gg6UeyJhY2Nlc3MiOiB7InRva2VuIjogeyJpc3N1ZWRfYXQiOiAiMjAxMy0wMC0yMFQxNjo1NjoyNi
4zNTg2MjUiLCAiZXhwaXJlcyI6ICIyMDEzLTEwLTIxVDE2OjU2OjI2WiIsICJpZCI6ICJwbGGFjZWhv
...
R3g14FJ0BxtTPbo6WarZ+sA3PZwdgIDyGNI-0Oqv-8ih4gJC9C6wBCel1dUXJ0Mn7BN-SfuxkooVk6
e090bcKjTWet3CC8IEj7a6LyLRVTdvmKGA5-pgp2mS5fb3G2mIad4Zeeb-zQn9V3Xf9WUGxuiVu1Hn
fhuUpJT-s9mU7+WEC3-8qkcBjEpqVCvMpmM4INI=
```

> **Note**
>
> This example shows a subset of a PKI token. A complete token is over 5000 characters long.

# Authenticate against an Image Service endpoint

To authenticate against an Image Service endpoint, instantiate a glanceclient.v2.client.Client object:

```
from os import environ as env
import glanceclient.v2.client as glclient
import keystoneclient.v2_0.client as ksclient

keystone = ksclient.Client(auth_url=env['OS_AUTH_URL'],
                           username=env['OS_USERNAME'],
                           password=env['OS_PASSWORD'],
                           tenant_name=env['OS_TENANT_NAME'],
                           region_name=env['OS_REGION_NAME'])
glance_endpoint = keystone.service_catalog.url_for(service_type='image')
glance = glclient.Client(glance_endpoint, token=keystone.auth_token)
```

# Authenticate against a Compute endpoint

To authenticate against a Compute endpoint, instantiate a novaclient.v_1_1.client.Client object:

```
from os import environ as env
import novaclient.v1_1.client as nvclient
nova = nvclient.Client(auth_url=env['OS_AUTH_URL'],
                       username=env['OS_USERNAME'],
                       api_key=env['OS_PASSWORD'],
                       project_id=env['OS_TENANT_NAME'],
                       region_name=env['OS_REGION_NAME'])
```

Alternatively, you can instantiate a `novaclient.client.Client` object and pass the version number:

```
from os import environ as env
import novaclient
nova = novaclient.client.Client("1.1", auth_url=env['OS_AUTH_URL'],
                                username=env['OS_USERNAME'],
                                api_key=env['OS_PASSWORD'],
                                project_id=env['OS_TENANT_NAME'],
                                region_name=env['OS_REGION_NAME'])
```

If you authenticate against an endpoint that uses a custom authentication back end, you must load the authentication plug-in and pass it to the constructor.

The Rackspace public cloud is an OpenStack deployment that uses a custom authentication back end. To authenticate against this cloud, you must install the rackspace-novaclient library that contains the Rackspace authentication plug-in, called `rackspace`. The following Python code shows the additional modifications required to instantiate a `Client` object that can authenticate against the Rackspace custom authentication back end.

```
import novaclient.auth_plugin
import novaclient.v1_1.client as nvclient
from os import environ as env
auth_system = 'rackspace'
auth_plugin = novaclient.auth_plugin.load_plugin('rackspace')
nova = nvclient.Client(auth_url=env['OS_AUTH_URL'],
                       username=env['OS_USERNAME'],
                       api_key=env['OS_PASSWORD'],
                       project_id=env['OS_TENANT_NAME'],
                       region_name=env['OS_REGION_NAME'],
                       auth_system='rackspace',
                       auth_plugin=auth_plugin)
```

If you set the `OS_AUTH_SYSTEM` environment variable, check for this variable in your Python script to determine whether you need to load a custom authentication back end:

```
import novaclient.auth_plugin
import novaclient.v1_1.client as nvclient
from os import environ as env
auth_system = env.get('OS_AUTH_SYSTEM', 'keystone')
if auth_system != "keystone":
  auth_plugin = novaclient.auth_plugin.load_plugin(auth_system)
else:
  auth_plugin = None
nova = nvclient.Client(auth_url=env['OS_AUTH_URL'],
                       username=env['OS_USERNAME'],
                       api_key=env['OS_PASSWORD'],
                       project_id=env['OS_TENANT_NAME'],
                       region_name=env['OS_REGION_NAME'],
                       auth_system=auth_system,
                       auth_plugin=auth_plugin)
```

## Authenticate against a Networking endpoint

To authenticate against a Networking endpoint, instantiate a
`neutronclient.v_2_0.client.Client` object:

```
from os import environ as env
from neutronclient.v2_0 import client as neutronclient
neutron = neutronclient.Client(auth_url=env['OS_AUTH_URL'],
                               username=env['OS_USERNAME'],
                               password=env['OS_PASSWORD'],
                               tenant_name=env['OS_TENANT_NAME'],
                               region_name=env['OS_REGION_NAME'])
```

You can also authenticate by explicitly specifying the endpoint and token:

```
from os import environ as env
import keystoneclient.v2_0.client as ksclient
from neutronclient.v2_0 import client as neutronclient
keystone = ksclient.Client(auth_url=env['OS_AUTH_URL'],
                           username=env['OS_USERNAME'],
                           password=env['OS_PASSWORD'],
                           tenant_name=env['OS_TENANT_NAME'],
                           region_name=env['OS_REGION_NAME'])
endpoint_url = keystone.service_catalog.url_for(service_type='network')
token = keystone.auth_token
neutron = neutronclient.Client(endpoint_url=endpoint_url, token=token)
```

# Manage images

When working with images in the SDK, you will call both `glance` and `nova` methods.

## List images

To list the available images, call the `glanceclient.v2.images.Controller.list`
method:

```
import glanceclient.v2.client as glclient
glance = glclient.Client(...)
images = glance.images.list()
```

The `images` method returns a Python generator, as shown in the following interaction
with the Python interpreter:

```
>>> images = glance.images.list()
>>> images
<generator object list at 0x105e9c2d0>
>>> list(images)
[{u'checksum': u'f8a2eeee2dc65b3d9b6e63678955bd83',
  u'container_format': u'ami',
  u'created_at': u'2013-10-20T14:28:10Z',
  u'disk_format': u'ami',
  u'file': u'/v2/images/dbc9b2db-51d7-403d-b680-3f576380b00c/file',
  u'id': u'dbc9b2db-51d7-403d-b680-3f576380b00c',
  u'kernel_id': u'c002c82e-2cfa-4952-8461-2095b69c18a6',
  u'min_disk': 0,
  u'min_ram': 0,
```

```
  u'name': u'cirros-0.3.2-x86_64-uec',
  u'protected': False,
  u'ramdisk_id': u'4c1c9b4f-3fe9-425a-a1ec-1d8fd90b4db3',
  u'schema': u'/v2/schemas/image',
  u'size': 25165824,
  u'status': u'active',
  u'tags': [],
  u'updated_at': u'2013-10-20T14:28:11Z',
  u'visibility': u'public'},
 {u'checksum': u'69c33642f44ca552ba4bb8b66ad97e85',
  u'container_format': u'ari',
  u'created_at': u'2013-10-20T14:28:09Z',
  u'disk_format': u'ari',
  u'file': u'/v2/images/4c1c9b4f-3fe9-425a-a1ec-1d8fd90b4db3/file',
  u'id': u'4c1c9b4f-3fe9-425a-a1ec-1d8fd90b4db3',
  u'min_disk': 0,
  u'min_ram': 0,
  u'name': u'cirros-0.3.2-x86_64-uec-ramdisk',
  u'protected': False,
  u'schema': u'/v2/schemas/image',
  u'size': 3714968,
  u'status': u'active',
  u'tags': [],
  u'updated_at': u'2013-10-20T14:28:10Z',
  u'visibility': u'public'},
 {u'checksum': u'c352f4e7121c6eae958bc1570324f17e',
  u'container_format': u'aki',
  u'created_at': u'2013-10-20T14:28:08Z',
  u'disk_format': u'aki',
  u'file': u'/v2/images/c002c82e-2cfa-4952-8461-2095b69c18a6/file',
  u'id': u'c002c82e-2cfa-4952-8461-2095b69c18a6',
  u'min_disk': 0,
  u'min_ram': 0,
  u'name': u'cirros-0.3.2-x86_64-uec-kernel',
  u'protected': False,
  u'schema': u'/v2/schemas/image',
  u'size': 4955792,
  u'status': u'active',
  u'tags': [],
  u'updated_at': u'2013-10-20T14:28:09Z',
  u'visibility': u'public'}]
```

## Get image by ID

To retrieve an image object from its ID, call the
`glanceclient.v2.images.Controller.get` method:

```
import glanceclient.v2.client as glclient
image_id = 'c002c82e-2cfa-4952-8461-2095b69c18a6'
glance = glclient.Client(...)
image = glance.images.get(image_id)
```

## Get image by name

The Image Service Python bindings do not support the retrieval of an im-
age object by name. However, the Compute Python bindings enable you
to get an image object by name. To get an image object by name, call the
`novaclient.v1_1.images.ImageManager.find` method:

```
import novaclient.v1_1.client as nvclient
name = "cirros"
nova = nvclient.Client(...)
image = nova.images.find(name=name)
```

## Upload an image

To upload an image, call the `glanceclient.v2.images.ImageManager.create` method:

```
import glanceclient.v2.client as glclient
imagefile = "/tmp/myimage.img"
glance = glclient.Client(...)
with open(imagefile) as fimage:
  glance.images.create(name="myimage", is_public=False, disk_format="qcow2",
                       container_format="bare", data=fimage)
```

# Configure access and security for instances

When working with images in the SDK, you will call `novaclient` methods.

# Add a keypair

To generate a keypair, call the `novaclient.v1_1.keypairs.KeypairManager.create` method:

```
import novaclient.v1_1.client as nvclient
nova = nvclient.Client(...)
keypair_name = "staging"
keypair = nova.keypairs.create(name=keypair_name)
print keypair.private_key
```

The Python script output looks something like this:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA8XkaMqInSPfy0hMfWO+OZRtIgrQAbQkNcaNHmv2GN2G6xZlb\nuBRux5Xk/6SZ
ABaNPm1nRWm/ZDHnxCsFTcAl2LYOQXx3Cl2qKNY4r2di4G48GAkd\n7k5lDP2RgQatUM8npO0CD9PU
...
mmrceYYK08/lQ7JKLmVkdzdQKt77+v1oBBuHiykLfI6h1m77NRDw9r8cV\nzczYeoALifpjTPMkKS8
ECfDCuDn/vc9K1He8CRaJHf8AMLQLM3MN
-----END RSA PRIVATE KEY-----
```

You typically write the private key to a file to use it later. The file must be readable and writeable by only the file owner; otherwise, the SSH client will refuse to read the private key file. It is safest to create the file with the appropriate permissions, as shown in the following example:

```
import novaclient.v1_1.client as nvclient
import os
nova = nvclient.Client(...)
keypair_name = "staging"
private_key_filename = "/home/alice/id-staging"
keypair = nova.keypairs.create(name=keypair_name)

# Create a file for writing that can only be read and written by owner
fp = os.open(private_key_filename, os.O_WRONLY | os.O_CREAT, 0o600)
with os.fdopen(fp, 'w') as f:
    f.write(keypair.private_key)
```

# Import a keypair

If you have already generated a keypair with the public key located at `~/.ssh/id_rsa.pub`, pass the contents of the file to the `novaclient.v1_1.keypairs.KeypairManager.create` method to import the public key to Compute:

```
import novaclient.v1_1.client as nvclient
import os.path
with open(os.path.expanduser('~/.ssh/id_rsa.pub')) as f:
    public_key = f.read()
nova = nvclient.Client(...)
nova.keypairs.create('mykey', public_key)
```

# List keypairs

To list keypairs, call the `novaclient.v1_1.keypairs.KeypairManager.list` method:

```
import novaclient.v1_1.client as nvclient
nova = nvclient.Client(...)
keypairs = nova.keypairs.list()
```

# Create and manage security groups

To list security groups for the current project, call the `novaclient.v_1.security_groups.SecurityGroupManager.list` method:

```
import novaclient.v1_1.client as nvclient
nova = nvclient.Client(...)
security_groups = nova.security_groups.list()
```

To create a security group with a specified name and description, call the `novaclient.v_1.security_groups.SecurityGroupManager.create` method:

```
import novaclient.v1_1.client as nvclient
nova = nvclient.Client(...)
nova.security_groups.create(name="web", description="Web servers")
```

To delete a security group, call the `novaclient.v_1.security_groups.SecurityGroupManager.delete` method, passing either a novaclient.v1_1.security_groups.SecurityGroup object or group ID as an argument:

```
import novaclient.v1_1.client as nvclient
nova = nvclient.Client(...)
group = nova.security_groups.find(name="web")
nova.security_groups.delete(group)
# The following lines would also delete the group:
# nova.security_groups.delete(group.id)
# group.delete()
```

# Create and manage security group rules

Access the security group rules from the `rules` attribute of a novaclient.v1_1.security_groups.SecurityGroup object:

```
import novaclient.v1_1.client as nvclient
nova = nvclient.Client(...)
group = nova.security_groups.find(name="web")
print group.rules
```

To add a rule, to a security group, call the
`novaclient.v1_1.security_group_rules.SecurityGroupRuleManager.create`
method:

```
import novaclient.v1_1.client as nvclient
nova = nvclient.Client(...)
group = nova.security_groups.find(name="web")
# Add rules for ICMP, tcp/80 and tcp/443
nova.security_group_rules.create(group.id, ip_protocol="icmp",
                                  from_port=-1, to_port=-1)
nova.security_group_rules.create(group.id, ip_protocol="tcp",
                                  from_port=80, to_port=80)
nova.security_group_rules.create(group.id, ip_protocol="tcp",
                                  from_port=443, to_port=443)
```

# Networking

To use the information in this section, you should have a general understanding of Open-
Stack Networking, OpenStack Compute, and the integration between the two. You should
also have access to a plug-in that implements the Networking API v2.0.

## Set environment variables

Make sure that you set the relevant environment variables.

As an example, see the sample shell file that sets these variables to get credentials:

```
export OS_USERNAME="admin"
export OS_PASSWORD="password"
export OS_TENANT_NAME="admin"
export OS_AUTH_URL="http://IPADDRESS/v2.0"
```

## Get credentials

The examples in this section use the `get_credentials` method:

```
def get_credentials():
    d = {}
    d['username'] = os.environ['OS_USERNAME']
    d['password'] = os.environ['OS_PASSWORD']
    d['auth_url'] = os.environ['OS_AUTH_URL']
    d['tenant_name'] = os.environ['OS_TENANT_NAME']
    return d
```

This code resides in the `credentials.py` file, which all samples import.

Use the `get_credentials()` method to populate and get a dictionary:

```
credentials = get_credentials()
```

## Get Nova credentials

Few examples in this section use the `get_nova_credentials` method:

```
def get_nova_credentials():
    d = {}
    d['username'] = os.environ['OS_USERNAME']
    d['api_key'] = os.environ['OS_PASSWORD']
    d['auth_url'] = os.environ['OS_AUTH_URL']
    d['project_id'] = os.environ['OS_TENANT_NAME']
    return d
```

This code resides in the `credentials.py` file, which all samples import.

Use the `get_nova_credentials()` method to populate and get a dictionary:

```
nova_credentials = get_nova_credentials()
```

## Print values

The examples in this section use the `print_values` and `print_values_server` methods:

```
def print_values(val, type):
    if type == 'ports':
        val_list = val['ports']
    if type == 'networks':
        val_list = val['networks']
    if type == 'routers':
        val_list = val['routers']
    for p in val_list:
        for k, v in p.items():
            print("%s : %s" % (k, v))
        print('\n')


def print_values_server(val, server_id, type):
    if type == 'ports':
        val_list = val['ports']

    if type == 'networks':
        val_list = val['networks']
    for p in val_list:
        bool = False
        for k, v in p.items():
            if k == 'device_id' and v == server_id:
                bool = True
        if bool:
            for k, v in p.items():
                print("%s : %s" % (k, v))
            print('\n')
```

This code resides in the `utils.py` file, which all samples import.

## Create network

The following program creates a network:

```
#!/usr/bin/env python
from neutronclient.v2_0 import client
from credentials import get_credentials

network_name = 'sample_network'
credentials = get_credentials()
neutron = client.Client(**credentials)
try:
    body_sample = {'network': {'name': network_name,
                   'admin_state_up': True}}

    netw = neutron.create_network(body=body_sample)
    net_dict = netw['network']
    network_id = net_dict['id']
    print('Network %s created' % network_id)

    body_create_subnet = {'subnets': [{'cidr': '192.168.199.0/24',
                          'ip_version': 4, 'network_id': network_id}]}

    subnet = neutron.create_subnet(body=body_create_subnet)
    print('Created subnet %s' % subnet)
finally:
    print("Execution completed")
```

# List networks

The following program lists networks:

```
#!/usr/bin/env python
from neutronclient.v2_0 import client
from credentials import get_credentials
from utils import print_values

credentials = get_credentials()
neutron = client.Client(**credentials)
netw = neutron.list_networks()

print_values(netw, 'networks')
```

For `print_values` see the section called "Print values" [145].

# Create ports

The following program creates a port:

```
#!/usr/bin/env python
from neutronclient.v2_0 import client
import novaclient.v1_1.client as nvclient
from credentials import get_credentials
from credentials import get_nova_credentials

credentials = get_nova_credentials()
nova_client = nvclient.Client(**credentials)

# Replace with server_id and network_id from your environment

server_id = '9a52795a-a70d-49a8-a5d0-5b38d78bd12d'
network_id = 'ce5d204a-93f5-43ef-bd89-3ab99ad09a9a'
```

```
server_detail = nova_client.servers.get(server_id)
print(server_detail.id)

if server_detail != None:
    credentials = get_credentials()
    neutron = client.Client(**credentials)

    body_value = {
                "port": {
                        "admin_state_up": True,
                        "device_id": server_id,
                        "name": "port1",
                        "network_id": network_id
                    }
                }
    response = neutron.create_port(body=body_value)
    print(response)
```

For `get_nova_credentials` see the section called "Get Nova credentials" [145].

For `get_credentials` see the section called "Get credentials" [144].

# List ports

The following program lists ports:

```
#!/usr/bin/env python
from neutronclient.v2_0 import client
from credentials import get_credentials
from utils import print_values

credentials = get_credentials()
neutron = client.Client(**credentials)
ports = neutron.list_ports()
print_values(ports, 'ports')
```

For `get_credentials` see the section called "Get credentials" [144].

For `print_values` see the section called "Print values" [145].

# List server ports

The following program lists the ports for a server:

```
#!/usr/bin/env python
from neutronclient.v2_0 import client
import novaclient.v1_1.client as nvclient
from credentials import get_credentials
from credentials import get_nova_credentials
from utils import print_values_server

credentials = get_nova_credentials()
nova_client = nvclient.Client(**credentials)

# change these values according to your environment

server_id = '9a52795a-a70d-49a8-a5d0-5b38d78bd12d'
```

```
network_id = 'ce5d204a-93f5-43ef-bd89-3ab99ad09a9a'
server_detail = nova_client.servers.get(server_id)
print(server_detail.id)

if server_detail is not None:
    credentials = get_credentials()
    neutron = client.Client(**credentials)
    ports = neutron.list_ports()

    print_values_server(ports, server_id, 'ports')
    body_value = {'port': {
        'admin_state_up': True,
        'device_id': server_id,
        'name': 'port1',
        'network_id': network_id,
        }}

    response = neutron.create_port(body=body_value)
    print(response)
```

# Create router and add port to subnet

This example queries OpenStack Networking to create a router and add a port to a subnet.

### To create a router and add a port to a subnet

1.  Import the following modules:

    ```
    from neutronclient.v2_0 import client
    import novaclient.v1_1.client as nvclient
    from credentials import get_credentials
    from credentials import get_nova_credentials
    from utils import print_values_server
    ```

2.  Get Nova Credentials. See .

3.  Instantiate the `nova_client` client object by using the `credentials` dictionary object:

    ```
    nova_client = nvclient.Client(**credentials)
    ```

4.  Create a router and add a port to the subnet:

    ```
    # Replace with network_id from your environment

    network_id = '81bf592a-9e3f-4f84-a839-ae87df188dc1'

    credentials = get_credentials()
    neutron = client.Client(**credentials)
    neutron.format = json
    request = {'router': {'name': 'router name',
                          'admin_state_up': True}}

    router = neutron.create_router(request)
    router_id = router['router']['id']
    # for example: '72cf1682-60a8-4890-b0ed-6bad7d9f5466'
    router = neutron.show_router(router_id)
    print(router)
    ```

```
body_value = {'port': {
    'admin_state_up': True,
    'device_id': router_id,
    'name': 'port1',
    'network_id': network_id,
    }}

response = neutron.create_port(body=body_value)
print(response)
print("Execution Completed")
```

### Example 3.1. Create router: complete code listing

```
#!/usr/bin/env python
from neutronclient.v2_0 import client
import novaclient.v1_1.client as nvclient
from credentials import get_credentials
from credentials import get_nova_credentials
from utils import print_values_server

credentials = get_nova_credentials()
nova_client = nvclient.Client(**credentials)

# Replace with network_id from your environment

network_id = '81bf592a-9e3f-4f84-a839-ae87df188dc1'
try:
    credentials = get_credentials()
    neutron = client.Client(**credentials)
    neutron.format = 'json'
    request = {'router': {'name': 'router name',
                          'admin_state_up': True}}
    router = neutron.create_router(request)
    router_id = router['router']['id']
    # for example: '72cf1682-60a8-4890-b0ed-6bad7d9f5466'
    router = neutron.show_router(router_id)
    print(router)
    body_value = {'port': {
        'admin_state_up': True,
        'device_id': router_id,
        'name': 'port1',
        'network_id': network_id,
        }}

    response = neutron.create_port(body=body_value)
    print(response)
finally:
    print("Execution completed")
```

# Delete a network

This example queries OpenStack Networking to delete a network.

### To delete a network

1.  Import the following modules:

    ```
    from neutronclient.v2_0 import client
    from credentials import get_credentials
    ```

2.   Get credentials. See the section called "Get credentials" [144].

3.   Instantiate the `neutron` client object by using the `credentials` dictionary object:

```
neutron = client.Client(**credentials)
```

4.   Delete the network:

```
body_sample = {'network': {'name': network_name,
                'admin_state_up': True}}

netw = neutron.create_network(body=body_sample)
net_dict = netw['network']
network_id = net_dict['id']
print('Network %s created' % network_id)

body_create_subnet = {'subnets': [{'cidr': '192.168.199.0/24',
                      'ip_version': 4, 'network_id': network_id}]}

subnet = neutron.create_subnet(body=body_create_subnet)
print('Created subnet %s' % subnet)

neutron.delete_network(network_id)
print('Deleted Network %s' % network_id)

print("Execution completed")
```

**Example 3.2. Delete network: complete code listing**

```
#!/usr/bin/env python
from neutronclient.v2_0 import client
from credentials import get_credentials

network_name = 'temp_network'
credentials = get_credentials()
neutron = client.Client(**credentials)
try:
    body_sample = {'network': {'name': network_name,
                    'admin_state_up': True}}

    netw = neutron.create_network(body=body_sample)
    net_dict = netw['network']
    network_id = net_dict['id']
    print('Network %s created' % network_id)

    body_create_subnet = {'subnets': [{'cidr': '192.168.199.0/24',
                          'ip_version': 4, 'network_id': network_id}]}

    subnet = neutron.create_subnet(body=body_create_subnet)
    print('Created subnet %s' % subnet)

    neutron.delete_network(network_id)
    print('Deleted Network %s' % network_id)
finally:
    print("Execution Completed")
```

# List routers

This example queries OpenStack Networking to list all routers.

### To list routers

1.  Import the following modules:

    ```
    from neutronclient.v2_0 import client
    from credentials import get_credentials
    from utils import print_values
    ```

2.  Get credentials. See the section called "Get credentials" [144].

3.  Instantiate the `neutron` client object by using the `credentials` dictionary object:

    ```
    neutron = client.Client(**credentials)
    ```

4.  List the routers

    ```
    routers_list = neutron.list_routers(retrieve_all=True)
    print_values(routers_list, 'routers')
    print("Execution completed")
    ```

    For `print_values` see the section called "Print values" [145].

### Example 3.3. List routers: complete code listing

```
#!/usr/bin/env python
from neutronclient.v2_0 import client
from credentials import get_credentials
from utils import print_values

try:
    credentials = get_credentials()
    neutron = client.Client(**credentials)
    routers_list = neutron.list_routers(retrieve_all=True)
    print_values(routers_list, 'routers')
finally:
    print("Execution completed")
```

# List security groups

This example queries OpenStack Networking to list security groups.

### To list security groups

1.  Import the following modules:

    ```
    from neutronclient.v2_0 import client
    from credentials import get_credentials
    from utils import print_values
    ```

2.  Get credentials. See the section called "Get credentials" [144].

3.  Instantiate the `neutron` client object by using the `credentials` dictionary object:

    ```
    neutron = client.Client(**credentials)
    ```

4.  List Security groups

    ```
    sg = neutron.list_security_groups()
    print(sg)
    ```

### Example 3.4. List security groups: complete code listing

```
#!/usr/bin/env python
from neutronclient.v2_0 import client
from credentials import get_credentials
from utils import print_values

credentials = get_credentials()
neutron = client.Client(**credentials)
sg = neutron.list_security_groups()
print(sg)
```

> **Note**
>
> OpenStack Networking security groups are case-sensitive while the `nova-net-work` security groups are case-insensitive.

# List subnets

This example queries OpenStack Networking to list subnets.

### To list subnets

1. Import the following modules:

   ```
   from neutronclient.v2_0 import client
   from credentials import get_credentials
   from utils import print_values
   ```

2. Get credentials. See the section called "Get credentials" [144].

3. Instantiate the `neutron` client object by using the `credentials` dictionary object:

   ```
   neutron = client.Client(**credentials)
   ```

4. List subnets:

   ```
   subnets = neutron.list_subnets()
   print(subnets)
   ```

### Example 3.5. List subnets: complete code listing

```
#!/usr/bin/env python
from neutronclient.v2_0 import client
from credentials import get_credentials
from utils import print_values

credentials = get_credentials()
neutron = client.Client(**credentials)
subnets = neutron.list_subnets()
print(subnets)
```

# Compute

To use the information in this section, you must be familiar with OpenStack Compute.

# Set environment variables

To set up environmental variables and authenticate against Compute API endpoints, see the section called "Authenticate" [137].

# Get OpenStack credentials (API v2)

These example use the `get_nova_credentials_v2` method:

```
def get_nova_credentials_v2():
    d = {}
    d['version'] = '2'
    d['username'] = os.environ['OS_USERNAME']
    d['api_key'] = os.environ['OS_PASSWORD']
    d['auth_url'] = os.environ['OS_AUTH_URL']
    d['project_id'] = os.environ['OS_TENANT_NAME']
    return d
```

This code resides in the `credentials.py` file, which all samples import.

Use the `get_nova_credentials_v2()` method to populate and get a dictionary:

```
credentials = get_nova_credentials_v2()
```

# List servers (API v2)

The following program lists servers by using the Compute API v2.

## To list servers

1.  Import the following modules:

    ```
    from credentials import get_nova_credentials_v2
    from novaclient.client import Client
    ```

2.  Get Nova credentials. See the section called "Get OpenStack credentials (API v2)" [153].

3.  Instantiate the `nova_client` client object by using the `credentials` dictionary object:

    ```
    nova_client = Client(**credentials)
    ```

4.  List servers by calling `servers.list` on `nova_client` object:

    ```
    print(nova_client.servers.list())
    ```

## Example 3.6. List servers code listing

```
#!/usr/bin/env python
from credentials import get_nova_credentials_v2
from novaclient.client import Client

credentials = get_nova_credentials_v2()
nova_client = Client(**credentials)

print(nova_client.servers.list())
```

# Create server (API v2)

The following program creates a server (VM) by using the Compute API v2.

### To create a server

1.  Import the following modules:

    ```
    import time
    from credentials import get_nova_credentials_v2
    from novaclient.client import Client
    ```

2.  Get OpenStack credentials. See the section called "Get OpenStack credentials (API v2)" [153].

3.  Instantiate the `nova_client` client object by using the `credentials` dictionary object:

    ```
    nova_client = Client(**credentials)
    ```

4.  Get the flavor and image to use to create a server. This code uses the `cirros` image, the `m1.tiny` flavor, and the `private` network:

    ```
    image = nova_client.images.find(name="cirros")
    flavor = nova_client.flavors.find(name="m1.tiny")
    net = nova_client.networks.find(label="private")
    ```

5.  To create the server, use the network, image, and flavor:

    ```
    nics = [{'net-id': net.id}]
    instance = nova_client.servers.create(name="vm2", image=image,
                                          flavor=flavor, key_name="keypair-1",
     nics=nics)
    ```

6.  Sleep for five seconds and determine whether the server/vm was created by calling `nova_client.servers.list()`:

    ```
    print("Sleeping for 5s after create command")
    time.sleep(5)
    print("List of VMs")
    print(nova_client.servers.list())
    ```

### Example 3.7. Create server code listing

```
#!/usr/bin/env python
import time
from credentials import get_nova_credentials_v2
from novaclient.client import Client

try:
    credentials = get_nova_credentials_v2()
    nova_client = Client(**credentials)

    image = nova_client.images.find(name="cirros")
    flavor = nova_client.flavors.find(name="m1.tiny")
    net = nova_client.networks.find(label="private")
    nics = [{'net-id': net.id}]
    instance = nova_client.servers.create(name="vm2", image=image,
```

```
                                          flavor=flavor, key_name="keypair-1",
 nics=nics)
    print("Sleeping for 5s after create command")
    time.sleep(5)
    print("List of VMs")
    print(nova_client.servers.list())
finally:
    print("Execution Completed")
```

# Delete server (API v2)

The following program deletes a server (VM) by using the Compute API v2.

## To delete a server

1.  Import the following modules:

    ```
    import time
    from credentials import get_nova_credentials_v2
    from novaclient.client import Client
    ```

2.  Get Nova credentials. See the section called "Get OpenStack credentials (API v2)" [153].

3.  Instantiate the `nova_client` client object by using the `credentials` dictionary object:

    ```
    nova_client = Client(**credentials)
    ```

4.  Determine whether the `vm1` server exists:

    a.  List servers: `servers_list`.

    b.  Iterate over `servers_list` and compare name with `vm1`.

    c.  If true, set the variable name `server_exists` to `True` and break from the for loop:

    ```
    servers_list = nova_client.servers.list()
    server_del = "vm1"
    server_exists = False

    for s in servers_list:
        if s.name == server_del:
            print("This server %s exists" % server_del)
            server_exists = True
            break
    ```

5.  If the server exists, run the `delete` method of the `nova_client.servers` object:

    ```
    nova_client.servers.delete(s)
    ```

## Example 3.8. Delete server code listing

```
#!/usr/bin/env python
from credentials import get_nova_credentials_v2
from novaclient.client import Client
```

```
credentials = get_nova_credentials_v2()
nova_client = Client(**credentials)

servers_list = nova_client.servers.list()
server_del = "vm1"
server_exists = False

for s in servers_list:
    if s.name == server_del:
        print("This server %s exists" % server_del)
        server_exists = True
        break
if not server_exists:
    print("server %s does not exist" % server_del)
else:
    print("deleting server..........")
    nova_client.servers.delete(s)
    print("server %s deleted" % server_del)
```

# Update server (API v2)

The following program updates the name of a server (VM) by using the Compute API v2.

## To update a server

1. Import the following modules:

   ```
   from credentials import get_nova_credentials_v2
   from novaclient.client import Client
   from utils import print_server
   ```

   `print_server` is a method defined in `utils.py` and prints the server details as shown in the code listing below:

   ```
   def print_server(server):
       print("-"*35)
       print("server id: %s" % server.id)
       print("server name: %s" % server.name)
       print("server image: %s" % server.image)
       print("server flavour: %s" % server.flavor)
       print("server key name: %s" % server.key_name)
       print("user_id: %s" % server.user_id)
       print("-"*35)
   ```

2. Get OpenStack Credentials. See the section called "Get OpenStack credentials (API v2)" [153].

3. Instantiate the `nova_client` client object by using the `credentials` dictionary object:

   ```
   nova_client = Client(**credentials)
   ```

4. Get the server instance using `server_id` and print the details by calling `print_server` method:

   ```
   server_id = '99889c8d-113f-4a7e-970c-77f1916bfe14'
   server = nova_client.servers.get(server_id)
   n = server.name
   print_server(server)
   ```

5. Call `server.update` on the server object with the new value for `name` variable:

```
server.update(name = n + '1')
```

6. Get the updated instance of the server:

```
server_updated = nova_client.servers.get(server_id)
```

7. Call `print_server` again to check the update server details:

```
print_server(server_updated)
```

### Example 3.9. Update server code listing

```python
#!/usr/bin/env python

from credentials import get_nova_credentials_v2
from novaclient.client import Client
from utils import print_server

credentials = get_nova_credentials_v2()
nova_client = Client(**credentials)

# Change the server_id specific to your environment

server_id = '99889c8d-113f-4a7e-970c-77f1916bfe14'
server = nova_client.servers.get(server_id)
n = server.name
print_server(server)

server.update(name=n +'1')
server_updated = nova_client.servers.get(server_id)
print_server(server_updated)
```

# List flavors (API v2)

The following program lists flavors and their details by using the Compute API v2.

### To list flavors

1. Import the following modules:

```python
from credentials import get_nova_credentials_v2
from novaclient.client import Client
from utils import print_flavors
```

The `print_flavors` method is defined in `utils.py` and prints the flavor details:

```python
def print_flavors(flavor_list):
    for flavor in flavor_list:
        print("-"*35)
        print("flavor id : %s" % flavor.id)
        print("flavor name : %s" % flavor.name)
    print("-"*35)
```

2. Get OpenStack credentials. See the section called "Get OpenStack credentials (API v2)" [153].

3.  Instantiate the `nova_client` client object by using the `credentials` dictionary object:

```
nova_client = Client(**credentials)
```

4.  List flavors by calling `list()` on `nova_client.flavors` object:

```
flavors_list =  nova_client.flavors.list()
```

5.  Print the flavor details, id and name by calling `print_flavors`:

```
print_flavors(flavors_list)
```

### Example 3.10. List flavors code listing

```
#!/usr/bin/env python

from credentials import get_nova_credentials_v2
from novaclient.client import Client
from utils import print_flavors

credentials = get_nova_credentials_v2()
nova_client = Client(**credentials)

flavors_list = nova_client.flavors.list()
print_flavors(flavors_list)
```

# List floating IPs (API v2)

The following program lists the floating IPs and their details by using the Compute API v2.

### To list floating IPs

1.  Import the following modules:

```
from credentials import get_nova_credentials_v2
from novaclient.client import Client
from utils import print_values_ip
```

The `print_values_ip` method is defined in `utils.py` and prints the floating_ip object details:

```
def print_values_ip(ip_list):
    ip_dict_lisl = []
    for ip in ip_list:
        print("-"*35)
        print("fixed_ip : %s" % ip.fixed_ip)
        print("id : %s" % ip.id)
        print("instance_id : %s" % ip.instance_id)
        print("ip : %s" % ip.ip)
        print("pool : %s" % ip.pool)
```

2.  Get OpenStack credentials. See the section called "Get OpenStack credentials (API v2)" [153].

3.  Instantiate the `nova_client` client object by using the `credentials` dictionary object:

```
nova_client = Client(**credentials)
```

4.  List floating IPs by calling `list()` on `nova_client.floating_ips` object:

```
ip_list = nova_client.floating_ips.list()
```

5.  Print the floating IP object details by calling `print_values_ip`:

```
print_values_ip(ip_list)
```

### Example 3.11. List floating IPs code listing

```python
#!/usr/bin/env python

from credentials import get_nova_credentials_v2
from novaclient.client import Client
from utils import print_values_ip

credentials = get_nova_credentials_v2()
nova_client = Client(**credentials)
ip_list = nova_client.floating_ips.list()
print_values_ip(ip_list)
```

# List hosts (API v2)

The following program lists the hosts by using the Compute API v2.

### To list hosts

1.  Import the following modules:

```
from credentials import get_nova_credentials_v2
from novaclient.client import Client
from utils import print_hosts
```

The `print_hosts` method is defined in `utils.py` and prints the host object details:

```python
def print_hosts(host_list):
    for host in host_list:
        print("-"*35)
        print("host_name : %s" % host.host_name)
        print("service : %s" % host.service)
        print("zone : %s" % host.zone)
    print("-"*35)
```

2.  Get OpenStack credentials. See the section called "Get OpenStack credentials (API v2)" [153].

3.  Instantiate the `nova_client` client object by using the `credentials` dictionary object:

```
nova_client = Client(**credentials)
```

4.  List hosts by calling `list()` on `nova_client.hosts` object:

```
host_list = nova_client.hosts.list()
```

5.  Print the host object details by calling `print_hosts(host_list)`:

```
        print_hosts(host_list)
```

**Example 3.12. List hosts code listing**

```
#!/usr/bin/env python

from credentials import get_nova_credentials_v2
from novaclient.client import Client
from utils import print_hosts

credentials = get_nova_credentials_v2()
nova_client = Client(**credentials)
host_list = nova_client.hosts.list()

print_hosts(host_list)
```

# Appendix A. OpenStack command-line interface cheat sheet

The following tables give a quick reference of the most used command-line commands.

## Table A.1. Identity (keystone)

| Description | Command |
|---|---|
| List all users | `$ keystone user-list` |
| List Identity service catalog | `$ keystone catalog` |
| List all services in service catalog | `$ keystone service-list` |
| Create new user | `$ keystone user-create --name NAME --tenant-id TENANT \`<br>`  --pass PASSWORD --email EMAIL --enabled BOOL` |
| Create new tenant | `$ keystone tenant-create --name NAME --description "DESCRIPTION" \`<br>`  --enabled BOOL` |

## Table A.2. Image Service (glance)

| Description | Command |
|---|---|
| List images you can access | `$ glance image-list` |
| Delete specified image | `$ glance image-delete IMAGE` |
| Describe a specific image | `$ glance image-show IMAGE` |
| Update image | `$ glance image-update IMAGE` |
| Manage images | |
| Kernel image | `$ glance image-create --name "cirros-threepart-kernel" \`<br>`  --disk-format aki --container-format aki --is-public False \`<br>`  --file ~/images/cirros-0.3.1~pre4-x86_64-vmlinuz` |
| RAM image | `$ glance image-create --name "cirros-threepart-ramdisk" \`<br>`  --disk-format ari --container-format ari --is-public False \`<br>`  --file ~/images/cirros-0.3.1~pre4-x86_64-initrd` |
| Three-part image | `$ glance image-create --name "cirros-threepart" --disk-format ami \`<br>`  --container-format ami --is-public False \`<br>`  --property kernel_id=$KID-property ramdisk_id=$RID \`<br>`  --file ~/images/cirros-0.3.1~pre4-x86_64-blank.img` |
| Register raw image | `$ glance image-create --name "cirros-qcow2" --disk-format qcow2 \`<br>`  --container-format bare --is-public False \`<br>`  --file ~/images/cirros-0.3.1~pre4-x86_64-disk.img` |

## Table A.3. Compute (nova)

| Description | Command |
|---|---|
| List instances, notice status of instance | `$ nova list` |
| List images | `$ nova image-list` |
| List flavors | `$ nova flavor-list` |
| Boot an instance using flavor and image names (if names are unique) | `$ nova boot --image IMAGE --flavor FLAVOR INSTANCE_NAME`<br>`$ nova boot --image cirros-0.3.1-x86_64-uec --flavor m1.tiny \`<br>`  MyFirstInstance` |
| Login to instance | `# ip netns`<br>`# ip netns exec NETNS_NAME ssh USER@SERVER`<br>`# ip netns exec qdhcp-6021a3b4-8587-4f9c-8064-0103885dfba2 \`<br>`  ssh cirros@10.0.0.2`<br><br>Note, in CirrOS the password for user `cirros` is "cubswin:)" without the quotes. |
| Show details of instance | `$ nova show NAME` |

| Description | Command |
|---|---|
| | `$ nova show MyFirstInstance` |
| View console log of instance | `$ nova console-log MyFirstInstance` |
| Set metadata on an instance | `$ nova meta volumeTwoImage set newmeta='my meta data'` |
| Create an instance snapshot | `$ nova image-create volumeTwoImage snapshotOfVolumeImage`<br>`$ nova image-show snapshotOfVolumeImage` |
| Pause, suspend, stop, rescue, resize, rebuild, reboot an instance | |
| Pause | `$ nova pause NAME`<br>`$ nova pause volumeTwoImage` |
| Unpause | `$ nova unpause NAME` |
| Suspend | `$ nova suspend NAME` |
| Unsuspend | `$ nova resume NAME` |
| Stop | `$ nova stop NAME` |
| Start | `$ nova start NAME` |
| Rescue | `$ nova rescue NAME`<br>`$ nova rescue NAME --rescue_image_ref RESCUE_IMAGE` |
| Resize | `$ nova resize NAME FLAVOR`<br>`$ nova resize my-pem-server m1.small`<br>`$ nova resize-confirm my-pem-server1` |
| Rebuild | `$ nova rebuild NAME IMAGE`<br>`$ nova rebuild newtinny cirros-qcow2` |
| Reboot | `$ nova reboot NAME`<br>`$ nova reboot newtinny` |
| Inject user data and files into an instance | `$ nova boot --user-data FILE INSTANCE`<br>`$ nova boot --user-data userdata.txt --image cirros-qcow2 \`<br>`  --flavor m1.tiny MyUserdataInstance2`<br><br>To validate that the file is there, ssh into the instance, and look in `/var/lib/cloud` for the file. |
| Inject a keypair into an instance and access the instance with that keypair | |
| Create keypair | `$ nova keypair-add test > test.pem`<br>`$ chmod 600 test.pem` |
| Boot | `$ nova boot --image cirros-0.3.0-x86_64 --flavor m1.small \`<br>`  --key_name test MyFirstServer` |
| Use ssh to connect to the instance | `# ip netns exec qdhcp-98f09f1e-64c4-4301-a897-5067ee6d544f \`<br>`  ssh -i test.pem cirros@10.0.0.4` |
| Manage security groups | |
| Add rules to default security group allowing ping and SSH between instances in the default security group | `$ nova secgroup-add-group-rule default default icmp -1 -1`<br>`$ nova secgroup-add-group-rule default default tcp 22 22` |

## Table A.4. Networking (neutron)

| Description | Command |
|---|---|
| Create network | `$ neutron net-create NAME` |
| Create a subnet | `$ neutron subnet-create NETWORK_NAME CIDR`<br>`$ neutron subnet-create my-network 10.0.0.0/29` |
| List network and subnet | `$ neutron net-list`<br>`$ neutron subnet-list` |
| Examine details of network and subnet | `$ neutron net-show ID_OR_NAME_OF_NETWORK`<br>`$ neutron subnet-show ID_OR_NAME_OF_NETWORK` |

## Table A.5. Block Storage (cinder)

| Description | Command |
|---|---|
| Manage volumes and volume snapshots | |

| Description | Command |
|---|---|
| Create a new volume | `$ cinder create SIZE_IN_GB --display-name NAME`<br>`$ cinder create 1 --display-name MyFirstVolume` |
| Boot an instance and attach to volume | `$ nova boot --image cirros-qcow2 --flavor m1.tiny MyVolumeInstance` |
| List volumes, notice status of volume | `$ cinder list` |
| Attach volume to instance after instance is active, and volume is available | `$ nova volume-attach INSTANCE_ID VOLUME_ID auto`<br>`$ nova volume-attach MyVolumeInstance /dev/vdb auto` |
| Manage volumes after login into the instance | |
| List storage devices | `# fdisk -l` |
| Make filesystem on volume | `# mkfs.ext3 /dev/vdb` |
| Create a mountpoint | `# mkdir /myspace` |
| Mount the volume at the mountpoint | `# mount /dev/vdb /myspace` |
| Create a file on the volume | `# touch /myspace/helloworld.txt`<br>`# ls /myspace` |
| Unmount the volume | `# umount /myspace` |

## Table A.6. Object Storage (swift)

| Description | Command |
|---|---|
| Display information for the account, container, or object | `$ swift stat`<br>`$ swift stat ACCOUNT`<br>`$ swift stat CONTAINER`<br>`$ swift stat OBJECT` |
| List containers | `$ swift list` |
| Create a container | `$ swift post CONTAINER_NAME` |
| Upload file to a container | `$ swift upload CONTAINER_NAME FILE_NAME`<br>`$ swift upload mycontainer myfile.txt` |
| List objects in container | `$ swift list container` |
| Download object from container | `$ swift download CONTAINER_NAME FILE_NAME` |
| Upload with chunks, for large file | `$ swift upload -S SIZE CONTAINER_NAME FILE_NAME`<br>`$ swift upload -S 64 container largeFile` |

# 4. HOT guide

## Table of Contents

# Writing a hello world HOT template

HOT is a new template format meant to replace the CloudFormation-compatible format (CFN) as the native format supported by the Orchestration module over time. This guide is targeted towards template authors and explains how to write HOT templates based on examples. A detailed specification of HOT can be found at the section called "Heat Orchestration Template (HOT) specification" [167].

This section gives an introduction on how to write HOT templates, starting from very basic steps and then going into more and more detail by means of examples.

## A most basic template

The most basic template you can think of contains only a single resource definition using only predefined properties. For example, the template below could be used to deploy a single compute instance:

```
heat_template_version: 2013-05-23

description: Simple template to deploy a single compute instance

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      key_name: my_key
      image: ubuntu-trusty-x86_64
      flavor: m1.small
```

Each HOT template has to include the `heat_template_version` key with value `2013-05-23`, the current HOT version. While the `description` key is optional, it is good practice to include some useful text that describes what users can do with the template. In case you want to provide a longer description that does not fit on a single line, you can provide multi-line text in YAML, for example:

```
description: >
  This is how you can provide a longer description
  of your template that goes over several lines.
```

The `resources` section is required and must contain at least one resource definition. In the above example, a compute instance is defined with fixed values for the `key_name`, `image` and `flavor` properties.

> **Note**
>
> All the defined elements (key pair, image, flavor) have to exist in the OpenStack environment where the template is used.

# Input parameters

Input parameters defined in the `parameters` section of a template allow users to customize a template during deployment. For example, this allows for providing custom key pair names or image IDs to be used for a deployment. From a template author's perspective, this helps to make a template more easily reusable by avoiding hardcoded assumptions.

The following example extends the previous template to provide parameters for the key pair, image and flavor properties of the resource:

```
heat_template_version: 2013-05-23

description: Simple template to deploy a single compute instance

parameters:
  key_name:
    type: string
    label: Key Name
    description: Name of key-pair to be used for compute instance
  image_id:
    type: string
    label: Image ID
    description: Image to be used for compute instance
  flavor:
    type: string
    label: Instance Type
    description: Type of instance (flavor) to be used

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      key_name: { get_param: key_name }
      image: { get_param: image_id }
      flavor: { get_param: flavor }
```

Values for the three parameters must be defined by the template user during the deployment of a stack. The `get_param` intrinsic function retrieves a user-specified value for a given parameter and uses this value for the associated resource property.

For more information about intrinsic functions, see the section called "Intrinsic functions" [175].

## Providing default values

You can provide default values for parameters. If a user doesn't define a value for a parameter, the default value is used during the stack deployment. The following example defines a default value `m1.small` for the `flavor` property:

```
parameters:
  flavor:
    type: string
    label: Instance Type
    description: Flavor to be used
    default: m1.small
```

### Note

If a template doesn't define a default value for a parameter, then the user must define the value, otherwise the stack creation will fail.

## Hidding parameters values

The values that a user provides when deploying a stack are available in the stack details and can be accessed by any user in the same tenant. To hide the value of a parameter, use the `hidden` boolean attribute of the parameter:

```
parameters:
  database_password:
    type: string
    label: Database Password
    description: Password to be used for database
    hidden: true
```

## Restricting user input

You can restrict the values of an input parameter to make sure that the user defines valid data for this parameter. The `constraints` property of an input parameter defines a list of constraints to apply for the parameter. The following example restricts the `flavor` parameter to a list of three possible values:

```
parameters:
  flavor:
    type: string
    label: Instance Type
    description: Type of instance (flavor) to be used
    constraints:
      - allowed_values: [ m1.medium, m1.large, m1.xlarge ]
        description: Value must be one of m1.medium, m1.large or m1.xlarge.
```

The following example defines multiple contraints for a password definition:

```
parameters:
  database_password:
    type: string
    label: Database Password
    description: Password to be used for database
    hidden: true
    constraints:
      - length: { min: 6, max: 8 }
        description: Password length must be between 6 and 8 characters.
      - allowed_pattern: "[a-zA-Z0-9]+"
        description: Password must consist of characters and numbers only.
      - allowed_pattern: "[A-Z]+[a-zA-Z0-9]*"
        description: Password must start with an uppercase character.
```

The list of supported constraints is available in the the section called "Parameter constraints" [170] section.

> **Note**
>
> You can define multiple constraints of the same type. Especially in the case of allowed patterns this not only allows for keeping regular expressions simple and maintainable, but also for keeping error messages to be presented to users precise.

## Template outputs

In addition to template customization through input parameters, you can provide information about the resources created during the stack deployment to the users in the `outputs` section of a template. In the following example the output section provides the IP address of the `my_instance` resource:

```
outputs:
  instance_ip:
    description: The IP address of the deployed instance
    value: { get_attr: [my_instance, first_address] }
```

> **Note**
>
> Output values are typically resolved using intrinsic function such as the `get_attr`. See the section called "Intrinsic functions" [175] for more information about intrinsic functions..

See the section called "Outputs section" [175] for more information about the `outputs` section.

# Heat Orchestration Template (HOT) specification

## Template structure

HOT templates are defined in YAML and use the following structure:

```
heat_template_version: 2013-05-23

description:
  # description of the template
```

```
parameter_groups:
  # declaration of input parameter groups and order

parameters:
  # declaration of input parameters

resources:
  # declaration of template resources

outputs:
  # declaration of output parameters
```

| | |
|---|---|
| **heat_template_version** | This key with value `2013-05-23` (or a later date) indicates that the YAML document is a HOT template of the specified version. |
| **description** | This optional key gives a description of the template, or the workload that can be deployed using the template. |
| **parameter_groups** | This section specifies how the input parameters should be grouped and the order to provide the parameters in.<br><br>This section is optional. |
| **parameters** | This section specifies input parameters that have to be provided when instantiating the template.<br><br>This section is optional. |
| **resources** | This section contains the declaration of the resources of the template. This section with at least one resource must be defined in any HOT template, or the template would not really do anything when being instantiated. |
| **outputs** | This section specifies output parameters available to users once the template has been instantiated.<br><br>This section is optional. |

# Parameter groups section

The `parameter_groups` section specifies how the input parameters should be grouped and the order to provide the parameters in. These groups are typically used to describe expected behavior for downstream user interfaces.

These groups are specified in a list with each group containing a list of associated parameters. The lists are used to denote the expected order of the parameters. A parameter can only be included in one group. Use the name of the parameter to add it to a group. The parameters details are defined in the `parameters` section.

```
parameter_groups:
- label: <human-readable label of parameter group>
  description: <description of the parameter group>
  parameters:
  - <param name>
  - <param name>
```

| **label** | A human-readable label that defines the associated group of parameters. |
| **description** | A human-readable description of the parameter group. |
| **parameters** | A list of parameters that belong with this parameter group. |
| **param name** | The name of a parameter defined in the `parameters` section. |

# Parameters section

The `parameters` section defines input parameters that have to be provided when instantiating the template. Such parameters are typically used to customize each deployment, or for binding to environment specifics like certain images.

Each parameter is specified in a separated nested block with the name of the parameter defined in the first line and additional attributes such as a type or a default value defined as nested elements:

```
parameters:
  <param name>:
    type: <string | number | json | comma_delimited_list | boolean>
    label: <human-readable name of the parameter>
    description: <description of the parameter>
    default: <default value for parameter>
    hidden: <true | false>
    constraints:
      <parameter constraints>
```

| **param name** | The name of the parameter. |
| **type** | The type of the parameter. Supported types are `string`, `number`, `comma_delimited_list`, `json` and `boolean`.<br><br>This attribute is required. |
| **label** | A human readable name for the parameter.<br><br>This attribute is optional. |
| **description** | A human readable description for the parameter.<br><br>This attribute is optional. |
| **default** | A default value for the parameter. This value is used if the user doesn't specify his own value during deployment.<br><br>This attribute is optional. |
| **hidden** | Defines whether the parameters should be hidden when a user requests information about a stack created from the template. This attribute can be used to hide passwords specified as parameters.<br><br>This attribute is optional and defaults to `false`. |
| **constraints** | A list of constraints to apply. The constraints are validated by the Orchestration engine when a user deploys a stack. The stack creation fails if the parameter value doesn't comply to the constraints. |

This attribute is optional.

The following example shows a minimalistic definition of two parameters:

```
parameters:
  user_name:
    type: string
    label: User Name
    description: User name to be configured for the application
  port_number:
    type: number
    label: Port Number
    description: Port number to be configured for the web server
```

## Note

The description and the label are optional, but defining these attributes is good practice to provide useful information about the role of the parameter to the user.

## Parameter constraints

The `constraints` block of a parameter definition defines additional validation constraints that apply to the value of the parameter. The parameter values provided by a user are validated against the contrainsts at instantiation time. The contrainsts are defined as a list with the following syntax:

```
constraints:
  - <constraint type>: <constraint definition>
    description: <constraint description>
```

| | |
|---|---|
| **constraint type** | Type of constraint to apply. The set of currently supported constraints is given below. |
| **constraint definition** | The actual constraint, depending on the constraint type. The concrete syntax for each constraint type is given below. |
| **description** | A description of the constraint. The text is presented to the user when the value he defines violates the constraint. If ommitted, a default validation message is presented to the user.<br><br>This attribute is optional. |

The following example shows the definition of a string parameter with two constraints. Note that while the descriptions for each constraint are optional, it is good practice to provide concrete descriptions to present useful messages to the user at deployment time.

```
parameters:
  user_name:
    type: string
    label: User Name
    description: User name to be configured for the application
    constraints:
      - length: { min: 6, max: 8 }
        description: User name must be between 6 and 8 characters
      - allowed_pattern: "[A-Z]+[a-zA-Z0-9]*"
        description: User name must start with an uppercase character
```

### Note

While the descriptions for each constraint are optional, it is good practice to provide concrete descriptions so useful messages can be presented to the user at deployment time.

The following sections list the supported types of parameter constraints, along with the syntax for each type.

## length

The `length` constraint applies to parameters of type `string`. It defines a lower and upper limit for the length of the string value.

The syntax of the `length` constraint is:

```
length: { min: <lower limit>, max: <upper limit> }
```

It is possible to define a length constraint with only a lower limit or an upper limit. However, at least one of `min` or `max` must be specified.

## range

The `range` constraint applies to parameters of type `number`. It defines a lower and upper limit for the numeric value of the parameter.

The syntax of the `range` constraint is:

```
range: { min: <lower limit>, max: <upper limit> }
```

It is possible to define a range constraint with only a lower limit or an upper limit. However, at least one of `min` or `max` must be specified.

The minimum and maximum boundaries are included in the range. For example, the following range constraint would allow for all numeric values between 0 and 10:

```
range: { min: 0, max: 10 }
```

## allowed_values

The `allowed_values` constraint applies to parameters of type `string` or `number`. It specifies a set of possible values for a parameter. At deployment time, the user-provided value for the respective parameter must match one of the elements of the list.

The syntax of the `allowed_values` constraint is:

```
allowed_values: [ <value>, <value>, ... ]
```

Alternatively, the following YAML list notation can be used:

```
allowed_values:
  - <value>
  - <value>
  - ...
```

For example:

```
parameters:
  instance_type:
    type: string
    label: Instance Type
    description: Instance type for compute instances
    constraints:
      - allowed_values:
        - m1.small
        - m1.medium
        - m1.large
```

## allowed_pattern

The `allowed_pattern` constraint applies to parameters of type `string`. It specifies a regular expression against which a user-provided parameter value must evaluate at deployment.

The syntax of the `allowed_pattern` constraint is:

```
allowed_pattern: <regular expression>
```

For example:

```
parameters:
  user_name:
    type: string
    label: User Name
    description: User name to be configured for the application
    constraints:
      - allowed_pattern: "[A-Z]+[a-zA-Z0-9]*"
        description: User name must start with an uppercase character
```

## custom_constraint

The `custom_constraint` constraint adds an extra step of validation, generally to check that the specified resource exists in the backend. Custom constraints get implemented by plug-ins and can provide any kind of advanced constraint validation logic.

The syntax of the `custom_constraint` constraint is:

```
custom_constraint: <name>
```

The `name` attribute specifies the concrete type of custom constraint. It corresponds to the name under which the respective validation plugin has been registered in the Orchestration engine.

For example:

```
parameters:
  key_name
    type: string
    description: SSH key pair
    constraints:
      - custom_constraint: nova.keypair
```

## Pseudo Parameters

In addition to parameters defined by a template author, the Orchestration module also creates two parameters for every stack that allow referential access to the stack's name and identifier. These parameters are named `OS::stack_name` for the stack name and `OS::stack_id` for the stack identifier. These values are accessible via the `get_param` intrinsic function, just like user-defined parameters.

# Resources section

The `resources` section defines actual resources that make up a stack deployed from the HOT template (for instance compute instances, networks, storage volumes).

Each resource is defined as a separate block in the `resources` section with the following syntax:

```
resources:
  <resource ID>:
    type: <resource type>
    properties:
      <property name>: <property value>
    metadata:
      <resource specific metadata>
    depends_on: <resource ID or list of ID>
    update_policy: <update policy>
    deletion_policy: <deletion policy>
```

| | |
|---|---|
| **resource ID** | A resource ID which must be unique within the `resources` section of the template. |
| **type** | The resource type, such as `OS::Nova::Server` or `OS::Neutron::Port`.<br><br>This attribute is required. |
| **properties** | A list of resource-specific properties. The property value can be provided in place, or via a function (see *Intrinsic functions*).<br><br>This section is optional. |
| **metadata** | Resource-specific metadata.<br><br>This section is optional. |
| **depends_on** | Dependencies of the resource on one or more resources of the template.<br><br>See *Resource dependencies* for details.<br><br>This attribute is optional. |

**update_policy**          Update policy for the resource, in the form of a nested dictionary. Whether update policies are supported and what the exact semantics are depends on the type of the current resource.

This attribute is optional.

**deletion_policy**        Deletion policy for the resource. Which type of deletion policy is supported depends on the type of the current resource.

This attribute is optional.

Depending on the type of resource, the resource block might include more resource specific data.

All resource types that can be used in CFN templates can also be used in HOT templates, adapted to the YAML structure as outlined above.

The following example demonstrates the definition of a simple compute resource with some fixed property values:

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: F18-x86_64-cfntools
```

# Resource dependencies

The `depends_on` attribute of a resource defines a dependency between this resource and one or more other resources.

If a resource depends on just one other resource, the ID of the other resource is specified as string of the `depends_on` attribute, as shown in the following example:

```
resources:
  server1:
    type: OS::Nova::Server
    depends_on: server2

  server2:
    type: OS::Nova::Server
```

If a resource depends on more than one other resources, the value of the `depends_on` attribute is specified as a list of resource IDs, as shown in the following example:

```
resources:
  server1:
    type: OS::Nova::Server
    depends_on: [ server2, server3 ]

  server2:
    type: OS::Nova::Server

  server3:
    type: OS::Nova::Server
```

# Outputs section

The `outputs` section defines output parameters that should be available to the user once a stack has been created. This would be, for example, parameters such as IP addresses of deployed instances, or URLs of web applications deployed as part of a stack.

Each output parameter is defined as a separate block within the outputs section according to the following syntax:

```
outputs:
  <parameter name>:
    description: <description>
    value: <parameter value>
```

| | |
|---|---|
| **parameter name** | The output parameter name, which must be unique within the `outputs` section of a template. |
| **description** | A short description of the output parameter. |
| | This attribute is optional. |
| **parameter value** | The value of the output parameter. This value is usually resolved by means of a function. See *Intrinsic functions* for details about the functions. |
| | This attribute is required. |

The example below shows how the IP address of a compute resource can be defined as an output parameter:

```
outputs:
  instance_ip:
    description: IP address of the deployed compute instance
    value: { get_attr: [my_instance, first_address] }
```

# Intrinsic functions

HOT provides a set of intrinsic functions that can be used inside templates to perform specific tasks, such as getting the value of a resource attribute at runtime. The following section describes the role and syntax of the intrinsic functions.

## get_attr

The `get_attr` function references an attribute of a resource. The attribute value is resolved at runtime using the resource instance created from the respective resource definition.

The syntax of the `get_attr` function is:

```
get_attr:
  - <resource ID>
  - <attribute name>
  - <key/index 1> (optional)
  - <key/index 2> (optional)
  - ...
```

**resource ID**          The resource ID for which the attribute needs to be resolved.

                         The resource ID must exist in the `resources` section of the template.

**attribute name**       The attribute name to be resolved. If the attribute returns a complex
                         data structure such as a list or a map, then subsequent keys or index-
                         es can be specified. These additional parameters are used to navigate
                         the data structure to return the desired value.

The following example demonstrates how to use the `get_param` function:

```
resources:
  my_instance:
    type: OS::Nova::Server
    # ...

outputs:
  instance_ip:
    description: IP address of the deployed compute instance
    value: { get_attr: [my_instance, first_address] }
  instance_private_ip:
    description: Private IP address of the deployed compute instance
    value: { get_attr: [my_instance, networks, private, 0] }
```

In this example, if the networks attribute contained the following data:

```
{"public": ["2001:0db8:0000:0000:0000:ff00:0042:8329", "1.2.3.4"],
 "private": ["10.0.0.1"]}
```

then the value of the `get_attr` function would resolve to `10.0.0.1` (first item of the
`private` entry in the `networks` map).

# get_file

The `get_file` function returns the content of a file into the template. It is generally used
as a file inclusion mechanism for files containing scripts or configuration files.

The syntax of the `get_file` function is:

```
get_file: <content key>
```

The `content key` is used to look up the `files` dictionary that is provided in the REST API
call. The Orchestration client command (**heat**) is `get_file` aware and will populate the
`files` dictionnary with the actual content of fetched paths and URLs. The Orchestration
client command supports relative paths and will transform these to the absolute URLs re-
quired by the Orcestration API.

> ## Note
>
> The `get_file` argument must be a static path or URL and not rely on intrin-
> sic functions like `get_param`. the Orchestration client does not process intrinsic
> functions (they are only processed by the Orchestration engine).

The example below demonstrates the `get_file` function usage with both relative and ab-
solute URLs:

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      # general properties ...
      user_data:
        get_file: my_instance_user_data.sh

  my_other_instance:
    type: OS::Nova::Server
    properties:
      # general properties ...
      user_data:
        get_file: http://example.com/my_other_instance_user_data.sh
```

The `files` dictionary generated by the Orchestration client during instantiation of the stack would contain the following keys: * `file:///path/to/my_instance_user_data.sh` * `http://example.com/my_other_instance_user_data.sh`*

## get_param

The `get_param` function references an input parameter of a template. It resolves to the value provided for this input parameter at runtime.

The syntax of the `get_param` function is:

```
get_param:
  - <parameter name>
  - <key/index 1> (optional)
  - <key/index 2> (optional)
  - ...
```

**parameter name**     The parameter name to be resolved. If the parameters returns a complex data structure such as a list or a map, then subsequent keys or indexes can be specified. These additional parameters are used to navigate the data structure to return the desired value.

The following example demonstrates the use of the `get_param` function:

```
parameters:
  instance_type:
    type: string
    label: Instance Type
    description: Instance type to be used.
  server_data:
    type: json

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      flavor: { get_param: instance_type}
      metadata: { get_param: [ server_data, metadata ] }
      key_name: { get_param: [ server_data, keys, 0 ] }
```

In this example, if the `instance_type` and `server_data` parameters contained the following data:

```
{"instance_type": "m1.tiny",
{"server_data": {"metadata": {"foo": "bar"},
                 "keys": ["a_key","other_key"]}}}
```

then the value of the property `flavor` would resolve to `m1.tiny`, `metadata` would re-
solve to `{"foo": "bar"}` and `key_name` would resolve to `a_key`.

## get_resource

The `get_resource` function references another resource within the same template. At
runtime, it is resolved to reference the ID of the referenced resource, which is resource type
specific. For example, a reference to a floating IP resource returns the respective IP address
at runtime. The syntax of the `get_resource` function is:

```
get_resource: <resource ID>
```

The resource ID of the referenced resource is given as single parameter to the get_resource
function.

For exemple:

```
resources:
  instance_port:
    type: OS::Neutron::Port
    properties: ...

  instance:
    type: OS::Nova::Server
    properties:
      ...
      networks:
        port: { get_resource: instance_port }
```

## list_join

The `list_join` function joins a list of strings with the given delimiter.

The syntax of the `list_join` function is:

```
list_join:
- <delimiter>
- <list to join>
```

For example:

```
list_join: [', ', ['one', 'two', 'and three']]
```

This resolve to the string `one, two, and three`.

## resource_facade

The `resource_facade` function retrieves data in a parent provider template.

A provider template provdes a custom definition of a resource, called its facade. For
more information about custom templates, see the section called "Template composi-
tion" [200]. The syntax of the `resource_facade` function is:

```
resource_facade: <data type>
```

`data type` can be one of `metadata`, `deletion_policy` or `update_policy`.

## str_replace

The `str_replace` function dynamically constructs strings by providing a template string with placeholders and a list of mappings to assign values to those placeholders at runtime. The placeholders are replaced with mapping values wherever a mapping key exactly matches a placeholder.

The syntax of the `str_replace` function is:

```
str_replace:
  template: <template string>
  params: <parameter mappings>
```

**template**     Defines the template string that contains placeholders which will be substituted at runtime.

**params**       Provides parameter mappings in the form of dictionary. Each key refers to a placeholder used in the `template` attribute.

The following example shows a simple use of the `str_replace` function in the outputs section of a template to build a URL for logging into a deployed application:

```
resources:
  my_instance:
    type: OS::Nova::Server
    # general metadata and properties ...

outputs:
  Login_URL:
    description: The URL to log into the deployed application
    value:
      str_replace:
        template: http://host/MyApplication
        params:
          host: { get_attr: [ my_instance, first_address ] }
```

The following examples show the use of the `str_replace` function to build an instance initialization script:

```
parameters:
  DBRootPassword:
    type: string
    label: Database Password
    description: Root password for MySQL
    hidden: true

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      # general properties ...
      user_data:
        str_replace:
          template: |
```

```
              #!/bin/bash
              echo "Hello world"
              echo "Setting MySQL root password"
              mysqladmin -u root password $db_rootpassword
              # do more things ...
           params:
             $db_rootpassword: { get_param: DBRootPassword }
```

# Instances

## Manage instances

### Create an instance

Use the `OS::Nova::Server` resource to create a Compute instance. The `flavor` property is the only mandatory one, but you need to define a boot source using one of the `image` or `block_device_mapping` properties.

You also need to define the `networks` property to indicate to which networks your instance must connect if multiple networks are available in your tenant.

The following example creates a simple instance, booted from an image, and connecting to the `private` network:

```
resources:
  instance:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64
      networks:
        - network: private
```

### Connect an instance to a network

Use the `networks` property of an `OS::Nova::Server` resource to define which networks an instance should connect to. Define each network as a YAML map, containing one of the following keys:

**port**
: The ID of an existing Networking port. You usually create this port in the same template using an `OS::Neutron::Port` resource. You will be able to associate a floating IP to this port, and the port to your Compute instance.

**network**
: The name or ID of an existing network. You don't need to create an `OS::Neutron::Port` resource if you use this property, but you will not be able to associate a floating IP with the instance interface in the template.

The following example demonstrates the use of the `port` and `network` properties:

```
resources:
```

```
instance_port:
  type: OS::Neutron::Port
  properties:
    network: private
    fixed_ips:
      - subnet_id: "private-subnet"

instance1:
  type: OS::Nova::Server
  properties:
    flavor: m1.small
    image: ubuntu-trusty-x86_64
    networks:
      - port: { get_resource: instance_port }

instance2:
  type: OS::Nova::Server
  properties:
    flavor: m1.small
    image: ubuntu-trusty-x86_64
    networks:
      - network: private
```

## Create and associate security groups to an instance

Use the OS::Neutron::SecurityGroup resource to create security groups.

Define the security_groups property of the OS::Neutron::Port resource to asso-
ciate security groups to a port, then associate the port to an instance.

The following example creates a security group allowing inbound connections on ports 80
and 443 (web server) and associates this security group to an instance port:

```
resources:
  web_secgroup:
    type: OS::Neutron::SecurityGroup
    properties:
      rules:
        - protocol: tcp
          remote_ip_prefix: 0.0.0.0/0
          port_range_min: 80
          port_range_max: 80
        - protocol: tcp
          remote_ip_prefix: 0.0.0.0/0
          port_range_min: 443
          port_range_max: 443

  instance_port:
    type: OS::Neutron::Port
    properties:
      network: private
      security_groups:
        - default
        - { get_resource: web_secgroup }
      fixed_ips:
        - subnet_id: private-subnet

  instance:
    type: OS::Nova::Server
```

```
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64
      networks:
        - port: { get_resource: instance_port }
```

# Create and associate a floating IP to an instance

You can use two sets of resources to create and associate floating IPs to instances.

## OS::Nova resources

Use the `OS::Nova::FloatingIP` resource to create a floating IP, and the `OS::Nova::FloatingIPAssociation` resource to associate the floating IP to an instance.

The following example creates an instance and a floating IP, and associate the floating IP to the instance:

```
resources:
  floating_ip:
    type: OS::Nova::FloatingIP
    properties:
      pool: public

  inst1:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64
      networks:
        - network: private

  association:
    type: OS::Nova::FloatingIPAssociation
    properties:
      - floating_ip: { get_resource: floating_ip }
      - server_id: { get_resource: instance }
```

## OS::Neutron resources

### Note

The Networking service (neutron) must be enabled on your OpenStack deployment to use these resources.

Use the `OS::Neutron::FloatingIP` resource to create a floating IP, and the `OS::Neutron::FloatingIPAssociation` resource to associate the floating IP to a port:

```
resources:
  instance_port:
    type: OS::Neutron::Port
    properties:
      network: private
      fixed_ips:
```

```
          - subnet_id: "private-subnet"

  floating_ip:
    type: OS::Neutron::FloatingIP
    properties:
      floating_network: public
      port_id: { get_resource: instance_port }

  association:
    type: OS::Neutron::FloatingIPAssociation
    properties:
      floatingip_id: { get_resource: floating_ip }
      port_id: { get_resource: instance_port }
```

## Enable remote access to an instance

The `key_name` attribute of the `OS::Nova::Server` resource defines the key pair to use to enable SSH remote access:

```
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64
      key_name: my_key
```

### Note

For more information about key pairs, see the Configure access and security for instances section of the OpenStack user guide.

## Create a key pair

You can create new key pairs with the `OS::Nova::KeyPair` resource. Key pairs can be imported or created during the stack creation.

If the `public_key` property is not specified, the Orchestration module creates a new key pair. If the `save_private_key` property is set to `true`, the `private_key` attribute of the resource holds the private key.

The following example creates a new key pair and uses it as authentication key for an instance:

```
resources:
  my_key:
    type: OS::Nova::KeyPair
    properties:
      save_private_key: true

  my_instance:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64
      key_name: { get_resource: my_key }
```

```
outputs:
  private_key:
    description: Private key
    value: { get_attr: [ my_key, private_key ] }
```

# Manage networks

## Create a network and a subnet

> **Note**
>
> The Networking service (neutron) must be enabled on your OpenStack deployment to create and manage networks and subnets. Networks and subnets cannot be created if your deployment uses legacy networking (nova-network).

Use the `OS::Neutron::Net` resource to create a network, and the `OS::Neutron::Subnet` resource to provide a subnet for this network:

```
resources:
  new_net:
    type: OS::Neutron::Net

  new_subnet:
    type: OS::Neutron::Subnet
    properties:
      network_id: { get_resource: new_net }
      cidr: "10.8.1.0/24"
      dns_nameservers: [ "8.8.8.8", "8.8.4.4" ]
      ip_version: 4
```

## Create and manage a router

Use the `OS::Neutron::Router` resource to create a router. You can define its gateway with the `external_gateway_info` property:

```
resources:
  router1:
    type: OS::Neutron::Router
    properties:
      external_gateway_info: { network: public }
```

You can connect subnets to routers with the `OS::Neutron::RouterInterface` resource:

```
resources:
  subnet1_interface:
    type: OS::Neutron::RouterInterface
    properties:
      router_id: { get_resource: router1 }
      subnet: private-subnet
```

## Complete network example

The following example creates a network stack:

- A network and an associated subnet.

- A router with an external gateway.

- An interface to the new subnet for the new router.

In this example, the `public` network is an existing shared network:

```
resources:
  internal_net:
    type: OS::Neutron::Net

  internal_subnet:
    type: OS::Neutron::Subnet
    properties:
      network_id: { get_resource: internal_net }
      cidr: "10.8.1.0/24"
      dns_nameservers: [ "8.8.8.8", "8.8.4.4" ]
      ip_version: 4

  internal_router:
    type: OS::Neutron::Router
    properties:
      external_gateway_info: { network: public }

  internal_interface:
    type: OS::Neutron::RouterInterface
    properties:
      router_id: { get_resource: internal_router }
      subnet: { get_resource: internal_subnet }
```

# Manage volumes

## Create a volume

Use the `OS::Cinder::Volume` resource to create a new Block Storage volume.

For example:

```
resources:
  my_new_volume:
    type: OS::Cinder::Volume
    properties:
      size: 10
```

The volumes that you create are empty by default. Use the `image` property to create a bootable volume from an existing image:

```
resources:
  my_new_bootable_volume:
    type: OS::Cinder::Volume
    properties:
      size: 10
      image: ubuntu-trusty-x86_64
```

You can also create new volumes from another volume, a volume snapshot, or a volume backup. Use the `source_volid`, `snapshot_id` or `backup_id` properties to create a new volume from an existing source.

For example, to create a new volume from a backup:

```
resources:
  another_volume:
    type: OS::Cinder::Volume
    properties:
      backup_id: 2fff50ab-1a9c-4d45-ae60-1d054d6bc868
```

In this example the `size` property is not defined because the Block Storage service uses the size of the backup to define the size of the new volume.

## Attach a volume to an instance

Use the `OS::Cinder::VolumeAttachment` resource to attach a volume to an instance.

The following example creates a volume and an instance, and attaches the volume to the instance:

```
resources:
  new_volume:
    type: OS::Cinder::Volume
    properties:
      size: 1

  new_instance:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      image: ubuntu-trusty-x86_64

  volume_attachment:
    type: OS::Cinder::VolumeAttachment
    properties:
      volume_id: { get_resource: new_volume }
      instance_uuid: { get_resource: new_instance }
```

## Boot an instance from a volume

Use the `block_device_mapping` property of the `OS::Nova::Server` resource to define a volume used to boot the instance. This property is a list of volumes to attach to the instance before its boot.

The following example creates a bootable volume from an image, and uses it to boot an instance:

```
resources:
  bootable_volume:
    type: OS::Cinder::Volume
    properties:
      size: 10
      image: ubuntu-trusty-x86_64

  instance:
    type: OS::Nova::Server
    properties:
      flavor: m1.small
      networks:
```

```
        - network: private
    block_device_mapping:
      - device_name: vda
        volume_id: { get_resource: bootable_volume }
        delete_on_termination: false
```

# Software configuration

There are a variety of options to configure the software which runs on the servers in your stack. These can be broadly divided into the following:

• Custom image building

• User-data boot scripts and cloud-init

• Software deployment resources

This section will describe each of these options and provide examples for using them together in your stacks.

# Image building

The first opportunity to influence what software is configured on your servers is by booting them with a custom-built image. There are a number of reasons you might want to do this, including:

• - since the required software is already on the image there is no need to download and install anything at boot time.

• - software downloads can fail for a number of reasons including transient network failures and inconsistent software repositories.

• - custom built images can be verified in test environments before being promoted to production.

• - post-boot configuration may depend on agents already being installed and enabled

A number of tools are available for building custom images, including:

• diskimage-builder image building tools for OpenStack

• imagefactory builds images for a variety of operating system/cloud combinations

Examples in this guide which require custom images will use diskimage-builder.

# User-data boot scripts and cloud-init

When booting a server it is possible to specify the contents of the user-data to be passed to that server. This user-data is made available either from configured config-drive or from the Metadata service.

How this user-data is consumed depends on the image being booted, but the most commonly used tool for default cloud images is Cloud-init.

Whether the image is using Cloud-init or not, it should be possible to specify a shell script in the user_data property and have it be executed by the server during boot:

```
resources:

  the_server:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data: |
        #!/bin/bash
        echo "Running boot script"
        # ...
```

: debugging these scripts it is often useful to view the boot log using `nova console-log <server-id>` to view the progress of boot script execution.

Often there is a need to set variable values based on parameters or resources in the stack. This can be done with the `str_replace` intrinsic function:

```
parameters:
  foo:
    default: bar

resources:

  the_server:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data:
        str_replace:
          template: |
            #!/bin/bash
            echo "Running boot script with $FOO"
            # ...
          params:
            $FOO: {get_param: foo}
```

: If a stack-update is performed and there are any changes at all to the content of user_data then the server will be replaced (deleted and recreated) so that the modified boot configuration can be run on a new server.

When these scripts grow it can become difficult to maintain them inside the template, so the `get_file` intrinsic function can be used to maintain the script in a separate file:

```
parameters:
  foo:
    default: bar

resources:

  the_server:
    type: OS::Nova::Server
    properties:
```

```
        # flavor, image etc
      user_data:
        str_replace:
          template: {get_file: the_server_boot.sh}
          params:
            $FOO: {get_param: foo}
```

:`str_replace` can replace any strings, not just strings starting with $. However doing this for the above example is useful because the script file can be executed for testing by passing in environment variables.

## Choosing the user_data_format

The `OS::Nova::Server` user_data_format property determines how the user_data should be formatted for the server. For the default value `HEAT_CFNTOOLS`, the user_data is bundled as part of the heat-cfntools cloud-init boot configuration data. While `HEAT_CFNTOOLS` is the default for `user_data_format`, it is considered legacy and `RAW` or `SOFTWARE_CONFIG` will generally be more appropriate.

For `RAW` the user_data is passed to Nova unmodified. For a Cloud-init enabled image, the following are both valid `RAW` user-data:

```
resources:

  server_with_boot_script:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data_format: RAW
      user_data: |
        #!/bin/bash
        echo "Running boot script"
        # ...

  server_with_cloud_config:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data_format: RAW
      user_data: |
        #cloud-config
        final_message: "The system is finally up, after $UPTIME seconds"
```

For `SOFTWARE_CONFIG` user_data is bundled as part of the software config data, and metadata is derived from any associated Software deployment resources.

## Signals and wait conditions

Often it is necessary to pause further creation of stack resources until the boot configuration script has notified that it has reached a certain state. This is usually either to notify that a service is now active, or to pass out some generated data which is needed by another resource. The resources `OS::Heat::WaitCondition` and `OS::Heat::SwiftSignal` both perform this function using different techniques and tradeoffs.

`OS::Heat::WaitCondition` is implemented as a call to the Orchestration API resource signal. The token is created using credentials for a user account which is scoped only to the

wait condition handle resource. This user is created when the handle is created, and is associated to a project which belongs to the stack, in an identity domain which is dedicated to the orchestration service.

Sending the signal is a simple HTTP request, as with this example using curl:

```
curl -i -X POST -H 'X-Auth-Token: <token>' \
     -H 'Content-Type: application/json' -H 'Accept: application/json' \
     '<wait condition URL>' --data-binary '<json containing signal data>'
```

The JSON containing the signal data is expected to be of the following format:

```
{
  "status": "SUCCESS",
  "reason": "The reason which will appear in the 'heat event-list' output",
  "data": "Data to be used elsewhere in the template via get_attr",
  "id": "Optional unique ID of signal"
}
```

All of these values are optional, and if not specified will be set to the following defaults:

```
{
  "status": "SUCCESS",
  "reason": "Signal <id> received",
  "data": null,
  "id": "<sequential number starting from 1 for each signal received>"
}
```

If status is set to FAILURE then the resource (and the stack) will go into a FAILED state using the reason as failure reason.

The following template example uses the convenience attribute curl_cli which builds a curl command with a valid token:

```
resources:
  wait_condition:
    type: OS::Heat::WaitCondition
    properties:
      handle: {get_resource: wait_handle}
      # Note, count of 5 vs 6 is due to duplicate signal ID 5 sent below
      count: 5
      timeout: 300

  wait_handle:
    type: OS::Heat::WaitConditionHandle

  the_server:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data_format: RAW
      user_data:
        str_replace:
          template: |
            #!/bin/sh
            # Below are some examples of the various ways signals
            # can be sent to the Handle resource

            # Simple success signal
```

```
                wc_notify --data-binary '{"status": "SUCCESS"}'

                # Or you optionally can specify any of the additional fields
                wc_notify --data-binary '{"status": "SUCCESS", "reason":
 "signal2"}'
                wc_notify --data-binary '{"status": "SUCCESS", "reason":
 "signal3", "data": "data3"}'
                wc_notify --data-binary '{"status": "SUCCESS", "reason":
 "signal4", "data": "data4"}'

                # If you require control of the ID, you can pass it.
                # The ID should be unique, unless you intend for duplicate
                # signals to overrite each other.  The following two calls
                # do the exact same thing, and will be treated as one signal
                # (You can prove this by changing count above to 7)
                wc_notify --data-binary '{"status": "SUCCESS", "id": "5"}'
                wc_notify --data-binary '{"status": "SUCCESS", "id": "5"}'

                # Example of sending a failure signal, optionally
                # reason, id, and data can be specified as above
                # wc_notify --data-binary '{"status": "FAILURE"}'
            params:
              wc_notify: { get_attr: [wait_handle, curl_cli] }

outputs:
  wc_data:
    value: { get_attr: [wait_condition, data] }
    # this would return the following json
    # {"1": null, "2": null, "3": "data3", "4": "data4", "5": null}

  wc_data_4:
    value: { get_attr: [wait_condition, data, '4'] }
    # this would return "data4"
```

OS::Heat::SwiftSignal is implemented by creating an Object Storage API temporary URL which is populated with signal data with an HTTP PUT. The orchestration service will poll this object until the signal data is available. Object versioning is used to store multiple signals.

Sending the signal is a simple HTTP request, as with this example using curl:

```
curl -i -X PUT '<object URL>' --data-binary '<json containing signal data>'
```

The above template example only needs to have the type changed to the swift signal resources:

```
resources:
  signal:
    type: OS::Heat::SwiftSignal
    properties:
      handle: {get_resource: wait_handle}
      timeout: 300

  signal_handle:
    type: OS::Heat::SwiftSignalHandle
  # ...
```

The decision to use OS::Heat::WaitCondition or OS::Heat::SwiftSignal will depend on a few factors:

- `OS::Heat::SwiftSignal` depends on the availability of an Object Storage API

- `OS::Heat::WaitCondition` depends on whether the orchestration service has been configured with a dedicated stack domain (which may depend on the availability of an Identity V3 API).

- The preference to protect signal URLs with token authentication or a secret webhook URL.

## Software config resources

Boot configuration scripts can also be managed as their own resources. This allows configuration to be defined once and run on multiple server resources. These software-config resources are stored and retrieved via dedicated calls to the Orchestration API. It is not possible to modify the contents of an existing software-config resource, so a stack-update which changes any existing software-config resource will result in API calls to create a new config and delete the old one.

The resource `OS::Heat::SoftwareConfig` is used for storing configs represented by text scripts, for example:

```
resources:
  boot_script:
    type: OS::Heat::SoftwareConfig
    properties:
      group: ungrouped
      config: |
        #!/bin/bash
        echo "Running boot script"
        # ...

  server_with_boot_script:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data_format: RAW
      user_data: {get_resource: boot_script}
```

The resource `OS::Heat::CloudConfig` allows Cloud-init cloud-config to be represented as template YAML rather than a block string. This allows intrinsic functions to be included when building the cloud-config. This also ensures that the cloud-config is valid YAML, although no further checks for valid cloud-config are done.

```
parameters:
  file_content:
    type: string
    description: The contents of the file /tmp/file

resources:
  boot_config:
    type: OS::Heat::CloudConfig
    properties:
      cloud_config:
        write_files:
        - path: /tmp/file
          content: {get_param: file_content}
```

```
server_with_cloud_config:
  type: OS::Nova::Server
  properties:
    # flavor, image etc
    user_data_format: RAW
    user_data: {get_resource: boot_config}
```

The resource `OS::Heat::MultipartMime` allows multiple
`OS::Heat::SoftwareConfig` and `OS::Heat::CloudConfig` resources to be com-
bined into a single Cloud-init multi-part message:

```
parameters:
  file_content:
    type: string
    description: The contents of the file /tmp/file

  other_config:
    type: string
    description: The ID of a software-config resource created elsewhere

resources:
  boot_config:
    type: OS::Heat::CloudConfig
    properties:
      cloud_config:
        write_files:
        - path: /tmp/file
          content: {get_param: file_content}

  boot_script:
    type: OS::Heat::SoftwareConfig
    properties:
      group: ungrouped
      config: |
        #!/bin/bash
        echo "Running boot script"
        # ...

  server_init:
    type: OS::Heat::MultipartMime
    properties:
      parts:
      - config: {get_resource: boot_config}
      - config: {get_resource: boot_script}
      - config: {get_resource: other_config}

  server:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data_format: RAW
      user_data: {get_resource: server_init}
```

# Software deployment resources

There are many situations where it is not desirable to replace the server whenever there is a
configuration change. The `OS::Heat::SoftwareDeployment` resource allows any num-
ber of software configurations to be added or removed from a server throughout its life-cy-
cle.

## Building custom image for software deployments

`OS::Heat::SoftwareConfig` resources are used to store software configuration, and a `OS::Heat::SoftwareDeployment` resource is used to associate a config resource with one server. The `group` attribute on `OS::Heat::SoftwareConfig` specifies what tool will consume the config content.

`OS::Heat::SoftwareConfig` has the ability to define a schema of `inputs` and which the configuration script supports. Inputs are mapped to whatever concept the configuration tool has for assigning variables/parameters.

Likewise, `outputs` are mapped to the tool's capability to export structured data after configuration execution. For tools which do not support this, outputs can always be written to a known file path for the hook to read.

The `OS::Heat::SoftwareDeployment` resource allows values to be assigned to the config inputs, and the resource remains in an `IN_PROGRESS` state until the server signals to heat what (if any) output values were generated by the config script.

## Custom image script

Each of the following examples requires that the servers be booted with a custom image. The following script uses diskimage-builder to create an image required in later examples:

```
# Clone the required repositories. Some of these are also available
# via pypi or as distro packages.
git clone https://git.openstack.org/openstack/diskimage-builder.git
git clone https://git.openstack.org/openstack/tripleo-image-elements.git
git clone https://git.openstack.org/openstack/heat-templates.git

# Required by diskimage-builder to discover element collections
export ELEMENTS_PATH=tripleo-image-elements/elements:heat-templates/hot/
software-config/elements

# The base operating system element(s) provided by the diskimage-builder
# elements collection. Other values which may work include:
# centos7, debian, opensuse, rhel, rhel7, or ubuntu
export BASE_ELEMENTS="fedora selinux-permissive"
# Install and configure the os-collect-config agent to poll the heat service
# for configuration changes to execute
export AGENT_ELEMENTS="os-collect-config os-refresh-config os-apply-config"


# heat-config installs an os-refresh-config script which will invoke the
# appropriate hook to perform configuration. The element heat-config-script
# installs a hook to perform configuration with shell scripts
export DEPLOYMENT_BASE_ELEMENTS="heat-config heat-config-script"

# Install a hook for any other chosen configuration tool(s).
# Elements which install hooks include:
# heat-config-cfn-init, heat-config-puppet, or heat-config-salt
export DEPLOYMENT_TOOL=""

# The name of the qcow2 image to create, and the name of the image
# uploaded to the OpenStack image registry.
export IMAGE_NAME=fedora-software-config
```

```
# Create the image
diskimage-builder/bin/disk-image-create vm $BASE_ELEMENTS $AGENT_ELEMENTS \
      $DEPLOYMENT_BASE_ELEMENTS $DEPLOYMENT_TOOL -o $IMAGE_NAME.qcow2

# Upload the image, assuming valid credentials are already sourced
glance image-create --disk-format qcow2 --container-format bare \
      --name $IMAGE_NAME < $IMAGE_NAME.qcow2
```

# Configuring with scripts

The Custom image script already includes the `heat-config-script` element so the built image will already have the ability to configure using shell scripts.

Config inputs are mapped to shell environment variables. The script can communicate outputs to heat by writing to the file `$heat_outputs_path.<output name>`. See the following example for a script which expects inputs `foo`, `bar` and generates an output `result`.

```
resources:
  config:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      inputs:
      - name: foo
      - name: bar
      outputs:
      - name: result
      config: |
        #!/bin/sh -x
        echo "Writing to /tmp/$bar"
        echo $foo > /tmp/$bar
        echo -n "The file /tmp/$bar contains `cat /tmp/$bar` for server
 $deploy_server_id during $deploy_action" > $heat_outputs_path.result
        echo "Written to /tmp/$bar"
        echo "Output to stderr" 1>&2

  deployment:
    type: OS::Heat::SoftwareDeployment
    properties:
      config:
        get_resource: config
      server:
        get_resource: server
      input_values:
        foo: fooooo
        bar: baaaaa

  server:
    type: OS::Nova::Server
    properties:
      # flavor, image etc
      user_data_format: SOFTWARE_CONFIG

outputs:
  result:
    value:
      get_attr: [deployment, result]
  stdout:
```

```
    value:
      get_attr: [deployment, deploy_stdout]
  stderr:
    value:
      get_attr: [deployment, deploy_stderr]
  status_code:
    value:
      get_attr: [deployment, deploy_status_code]
```

> : A config resource can be associated with multiple deployment resources, and each deployment can specify the same or different values for the `server` and `input_values` properties.

As can be seen in the `outputs` section of the above template, the `result` config output value is available as an attribute on the `deployment` resource. Likewise the captured stdout, stderr and status_code are also available as attributes.

## Configuring with os-apply-config

The agent toolchain of `os-collect-config`, `os-refresh-config` and `os-apply-config` can actually be used on their own to inject heat stack configuration data into a server running a custom image.

The custom image needs to have the following to use this approach:

• All software dependencies installed

• os-refresh-config scripts to be executed on configuration changes

• os-apply-config templates to transform the heat-provided config data into service configuration files

The projects tripleo-image-elements and tripleo-heat-templates demonstrate this approach.

## Configuring with cfn-init

Likely the only reason to use the `cfn-init` hook is to migrate templates which contain AWS::CloudFormation::Init metadata without needing a complete rewrite of the config metadata. It is included here as it introduces a number of new concepts.

To use the `cfn-init` tool the `heat-config-cfn-init` element is required to be on the built image, so Custom image script needs to be modified with the following:

```
export DEPLOYMENT_TOOL="heat-config-cfn-init"
```

Configuration data which used to be included in the `AWS::CloudFormation::Init` section of resource metadata is instead moved to the `config` property of the config resource, as in the following example:

```
resources:

  config:
    type: OS::Heat::StructuredConfig
    properties:
      group: cfn-init
```

```
      inputs:
      - name: bar
      config:
        config:
          files:
            /tmp/foo:
              content:
                get_input: bar
              mode: '000644'

deployment:
  type: OS::Heat::StructuredDeployment
  properties:
    name: 10_deployment
    signal_transport: NO_SIGNAL
    config:
      get_resource: config
    server:
      get_resource: server
    input_values:
      bar: baaaaa

other_deployment:
  type: OS::Heat::StructuredDeployment
  properties:
    name: 20_other_deployment
    signal_transport: NO_SIGNAL
    config:
      get_resource: config
    server:
      get_resource: server
    input_values:
      bar: barmy

server:
  type: OS::Nova::Server
  properties:
    image: {get_param: image}
    flavor: {get_param: flavor}
    key_name: {get_param: key_name}
    user_data_format: SOFTWARE_CONFIG
```

There are a number of things to note about this template example:

- `OS::Heat::StructuredConfig` is like `OS::Heat::SoftwareConfig` except that
  the `config` property contains structured YAML instead of text script. This is useful for a
  number of other configuration tools including ansible, salt and os-apply-config.

- `cfn-init` has no concept of inputs, so `{get_input: bar}` acts as a placeholder
  which gets replaced with the `OS::Heat::StructuredDeployment` `input_values`
  value when the deployment resource is created.

- `cfn-init` has no concept of outputs, so specifying `signal_transport: NO_SIGNAL`
  will mean that the deployment resource will immediately go into the `CREATED` state in-
  stead of waiting for a completed signal from the server.

- The template has 2 deployment resources deploying the same config with differ-
  ent `input_values`. The order these are deployed in on the server is determined

by sorting the values of the `name` property for each resource (10_deployment, 20_other_deployment)

# Configuring with puppet

The puppet hook makes it possible to write configuration as puppet manifests which are deployed and run in a masterless environment.

To specify configuration as puppet manifests the `heat-config-puppet` element is required to be on the built image, so Custom image script needs to be modified with the following:

```
export DEPLOYMENT_TOOL="heat-config-puppet"
```

```
resources:

  config:
    type: OS::Heat::SoftwareConfig
    properties:
      group: puppet
      inputs:
      - name: foo
      - name: bar
      outputs:
      - name: result
      config:
        get_file: example-puppet-manifest.pp

  deployment:
    type: OS::Heat::SoftwareDeployment
    properties:
      config:
        get_resource: config
      server:
        get_resource: server
      input_values:
        foo: fooooo
        bar: baaaaa

  server:
    type: OS::Nova::Server
    properties:
      image: {get_param: image}
      flavor: {get_param: flavor}
      key_name: {get_param: key_name}
      user_data_format: SOFTWARE_CONFIG

outputs:
  result:
    value:
      get_attr: [deployment, result]
  stdout:
    value:
get_attr: [deployment, deploy_stdout]
```

This demonstrates the use of the `get_file` function, which will attach the contents of the file `example-puppet-manifest.pp`, containing:

```
file { 'barfile':
    ensure  => file,
    mode    => '0644',
    path    => '/tmp/$::bar',
    content => '$::foo',
}

file { 'output_result':
    ensure  => file,
    path    => '$::heat_outputs_path.result',
    mode    => '0644',
    content => 'The file /tmp/$::bar contains $::foo',
}
```

# Environments

The environment affects the runtime behaviour of a template. It provides a way to override the resource implementations and a mechanism to place parameters that the service needs.

To fully understand the runtime behavior you have to consider what plug-ins are installed on the cloud you're using.

# Environment file format

The environment is a yaml text file that contains two main sections:

**parameters**                    A map of key/pair values.

**resource_registry**             Definition of custom resources.

Use the -e option of the **heat stack-create** command to create a stack using with the environment defined in such a file.

You can also provide environment parameters as a list of key/value pairs using the -P option of the **heat stack-create** command.

In the following example the environment is read from the `my_env.yaml` file and an extra parameter is provided using the -P option:

```
$ heat stack-create my_stack -e my_env.yaml -P "param1=val1;param2=val2" -f
 my_tmpl.yaml
```

# Global and effective environments

The environment used for a stack is the combination of the environment you use with the template for the stack, and a global environment that is determined by your cloud operator. An entry in the user environment takes precedence over the global environment. OpenStack includes a default global environment, but you cloud operator can add additional environment entries.

# Define values for a template arguments

You can define values for the template arguments in the `parameters` section of an environment file:

```
parameters:
  KeyName: my_keypair
  InstanceType: m1.tiny
  ImageId: F18-x86_64-cfntools
```

# Create and override resources

You can create or override resources in the `resource_registry` section of an environment file. The resource you provide in this manner must have an identifier, and references either other resources IDs or the URL of an existing template file.

The following example maps the new `OS::Networking::FloatingIP` resource to the existing `OS::Nova::FloatingIP` resource:

```
resource_registry:
  "OS::Networking::FloatingIP": "OS::Nova::FloatingIP"
```

You can use wildcards to map multiple resources:

```
resource_registry:
  "OS::Network*": "OS::Neutron*"
```

To create or override a resource with a custom resource, create a template file to define this resource, and provide the URL to the template file in the environment file:

```
resource_registry:
  "AWS::EC2::Instance": file:///path/to/my_instance.yaml
```

The supported URL scheme are `file`, `http` and `https`.

> **Note**
>
> The template file extension must be `.yaml` or `.template`, or it will not be treated as a custom template resource.

You can limit the usage of a custom resource to a specific resource of the template:

```
resource_registry:
  resources:
    my_db_server:
      "OS::DBInstance": file:///home/mine/all_my_cool_templates/db.yaml
```

# Template composition

When writing complex templates you are encouraged to break up your template into separate smaller templates. These can then be brought together using template resources. This is a mechanism to define a resource using a template, thus composing one logical stack with multiple templates.

Template resources provide a feature similar to the `AWS::CloudFormation::Stack` resource, but also provide a way to:

- Define new resource types and build your own resource library.

- Override the default behaviour of existing resource types.

To achieve this:

- The Orchestration client gets the associated template files and passes them along in the `files` section of the `POST stacks/` API request.

- The environment in the Orchestration engine manages the mapping of resource type to template creation.

- The Orchestration engine translates template parameters into resource properties.

The following examples illustrate how you can use a custom template to define new types of resources. These examples use a custom template stored in a `my_nova.yml` file:

```
heat_template_version: 2013-05-23

parameters:
  key_name:
    type: string
    description: Name of a KeyPair

resources:
  server:
    type: OS::Nova::Server
    properties:
      key_name: {get_param: key_name}
      flavor: m1.small
      image: ubuntu-trusty-x86_64
```

# Use the template filename as type

The following template defines the `my_nova.yaml` file as value for the `type` property of a resource:

```
heat_template_version: 2013-05-23
resources:
  my_server:
    type: my_nova.yaml
    properties:
      key_name: my_key
```

The `key_name` argument of the `my_nova.yaml` template gets its value from the `key_name` property of the new template.

## Note

The above reference to `my_nova.yaml` assumes it is in the same directory. You can use any of the following forms:

- Relative path (`my_nova.yaml`)

- Absolute path (`file:///home/user/templates/my_nova.yaml`)

- Http URL (`http://example.com/templates/my_nova.yaml`)

- Https URL (`https://example.com/templates/my_nova.yaml`)

To create the stack run:

```
$ heat stack-create -f main.yaml stack1
```

# Define a new resource type

You can associate a name to the `my_noya.yaml` template in an environment file. If the name is already known by the Orchestration module then your new resource will override the default one.

In the following example a new `OS::Nova::Server` resource overrides the default resource of the same name.

An `env.yaml` environment file holds the definition of the new resource:

```
resource_registry:
  "OS::Nova::Server": my_nova.yaml
```

> **Note**
>
> See the section called "Environments" [199] for more detail about environment files.

You can now use the new `OS::Nova::Server` in your new template:

```
resources:
  my_server:
    type: OS::Nova::Server
    properties:
      key_name: my_key
```

To create the stack run:

```
$ heat stack-create -f main.yaml -e env.yaml example-two
```

# Appendix B. Community support

## Table of Contents

The following resources are available to help you run and use OpenStack. The OpenStack community constantly improves and adds to the main features of OpenStack, but if you have any questions, do not hesitate to ask. Use the following resources to get OpenStack support, and troubleshoot your installations.

# Documentation

For the available OpenStack documentation, see docs.openstack.org.

To provide feedback on documentation, join and use the `<openstack-docs@lists.openstack.org>` mailing list at OpenStack Documentation Mailing List, or report a bug.

The following books explain how to install an OpenStack cloud and its associated components:

- *Installation Guide for Debian 7*

- *Installation Guide for openSUSE 13.1 and SUSE Linux Enterprise Server 11 SP3*

- *Installation Guide for Red Hat Enterprise Linux 7, CentOS 7, and Fedora 20*

- *Installation Guide for Ubuntu 14.04*

The following books explain how to configure and run an OpenStack cloud:

- *Architecture Design Guide*

- *Cloud Administrator Guide*

- *Configuration Reference*

- *Operations Guide*

- *High Availability Guide*

- *Security Guide*

- *Virtual Machine Image Guide*

The following books explain how to use the OpenStack dashboard and command-line clients:

- *API Quick Start*

- *End User Guide*

- *Admin User Guide*

- *Command-Line Interface Reference*

The following documentation provides reference and guidance information for the OpenStack APIs:

- OpenStack API Complete Reference (HTML)

- API Complete Reference (PDF)

- *OpenStack Block Storage Service API v2 Reference*

- *OpenStack Compute API v2 and Extensions Reference*

- *OpenStack Identity Service API v2.0 Reference*

- *OpenStack Image Service API v2 Reference*

- *OpenStack Networking API v2.0 Reference*

- *OpenStack Object Storage API v1 Reference*

The *Training Guides* offer software training for cloud administration and management.

# ask.openstack.org

During the set up or testing of OpenStack, you might have questions about how a specific task is completed or be in a situation where a feature does not work correctly. Use the ask.openstack.org site to ask questions and get answers. When you visit the http://ask.openstack.org site, scan the recently asked questions to see whether your question has already been answered. If not, ask a new question. Be sure to give a clear, concise summary in the title and provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.

# OpenStack mailing lists

A great way to get answers and insights is to post your question or problematic scenario to the OpenStack mailing list. You can learn from and help others who might have similar issues. To subscribe or view the archives, go to http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack. You might be interested in the other mailing lists for specific projects or development, which you can find on the wiki. A description of all mailing lists is available at http://wiki.openstack.org/MailingLists.

# The OpenStack wiki

The OpenStack wiki contains a broad range of topics but some of the information can be difficult to find or is a few pages deep. Fortunately, the wiki search feature enables you to search by title or content. If you search for specific information, such as about networking or nova, you can find a large amount of relevant material. More is being added all the time, so be sure to check back often. You can find the search box in the upper-right corner of any OpenStack wiki page.

# The Launchpad Bugs area

The OpenStack community values your set up and testing efforts and wants your feedback. To log a bug, you must sign up for a Launchpad account at https://launchpad.net/+login. You can view existing bugs and report bugs in the Launchpad Bugs area. Use the search feature to determine whether the bug has already been reported or already been fixed. If it still seems like your bug is unreported, fill out a bug report.

Some tips:

• Give a clear, concise summary.

• Provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.

• Be sure to include the software and package versions that you are using, especially if you are using a development branch, such as, `"Juno release" vs git commit bc79c3ecc55929bac585d04a03475b72e06a3208`.

• Any deployment-specific information is helpful, such as whether you are using Ubuntu 14.04 or are performing a multi-node installation.

The following Launchpad Bugs areas are available:

• Bugs: OpenStack Block Storage (cinder)

• Bugs: OpenStack Compute (nova)

• Bugs: OpenStack Dashboard (horizon)

• Bugs: OpenStack Identity (keystone)

• Bugs: OpenStack Image Service (glance)

• Bugs: OpenStack Networking (neutron)

• Bugs: OpenStack Object Storage (swift)

• Bugs: Bare Metal (ironic)

• Bugs: Data Processing Service (sahara)

• Bugs: Database Service (trove)

- Bugs: Orchestration (heat)

- Bugs: Telemetry (ceilometer)

- Bugs: Queue Service (marconi)

- Bugs: OpenStack API Documentation (developer.openstack.org)

- Bugs: OpenStack Documentation (docs.openstack.org)

# The OpenStack IRC channel

The OpenStack community lives in the #openstack IRC channel on the Freenode network. You can hang out, ask questions, or get immediate feedback for urgent and pressing issues. To install an IRC client or use a browser-based client, go to http://webchat.freenode.net/. You can also use Colloquy (Mac OS X, http://colloquy.info/), mIRC (Windows, http://www.mirc.com/), or XChat (Linux). When you are in the IRC channel and want to share code or command output, the generally accepted method is to use a Paste Bin. The OpenStack project has one at http://paste.openstack.org. Just paste your longer amounts of text or logs in the web form and you get a URL that you can paste into the channel. The OpenStack IRC channel is `#openstack` on `irc.freenode.net`. You can find a list of all OpenStack IRC channels at https://wiki.openstack.org/wiki/IRC.

# Documentation feedback

To provide feedback on documentation, join and use the `<openstack-docs@lists.openstack.org>` mailing list at OpenStack Documentation Mailing List, or report a bug.

# OpenStack distribution packages

The following Linux distributions provide community-supported packages for OpenStack:

- **Debian:** http://wiki.debian.org/OpenStack

- **CentOS, Fedora, and Red Hat Enterprise Linux:** http://openstack.redhat.com/

- **openSUSE and SUSE Linux Enterprise Server:** http://en.opensuse.org/Portal:OpenStack

- **Ubuntu:** https://wiki.ubuntu.com/ServerTeam/CloudArchive