OpenStack Operator's Manual

version 1

Nick West

April 22, 2021

Contents

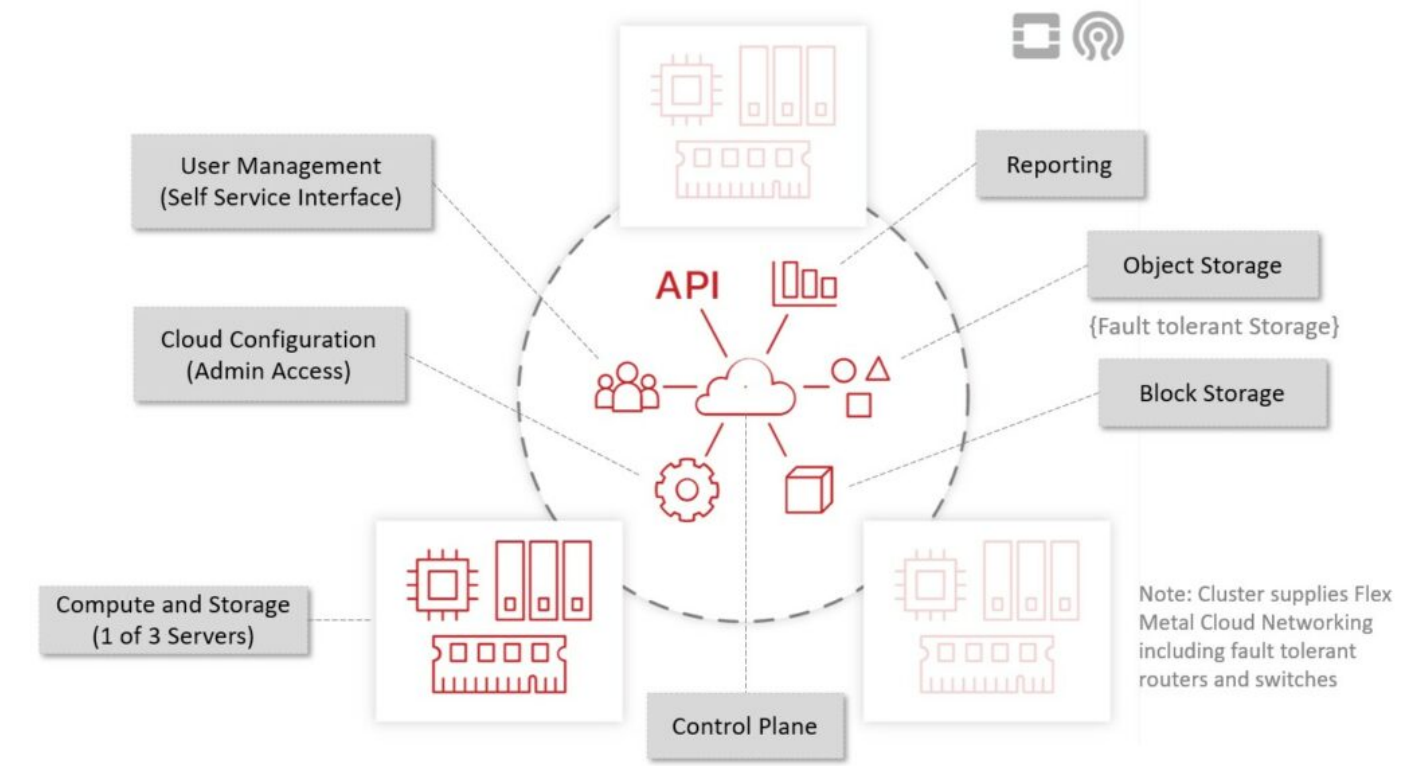
OpenStack Operators Manual - Production in 40 Hours 1
Private Cloud Core - OpenStack Operator Manual V1.2.1 1
Getting Started with OpenStack 1
Login to Horizon 1
Next Steps 3
Install OpenStackClient 3
Installing OpenStackClient 4
Next Steps 4
Manage OpenStack Users, Groups and Projects 5
Create and Manage Users 5
Create Users Using Horizon 5
Create Users Using the Command Line 6
Assign Role to a User 7
Create and Manage Groups 8
Create Groups Using Horizon 8
Add Users to a Group Using Horizon 8
Create Groups Using the Command Line 9
Add Users to a Group Using the Command Line 9
Create and Manage Projects 10
Create Projects Using Horizon 10
Create Projects Using the Command Line 11
Add Group to Project Using the Command Line 12
Next Steps 12
Networks, Routers, and Floating IPs 12
Exercises covered in this guide 12
Common terms 13
Create a network and router 13
Create a network 13
Create a router 14
Floating IPs 16
Allocate and Assign Floating IPs using Horizon 16
Troubleshooting 16
Check neutron docker containers 17
Next Steps 17
Create a Network using OpenStackClient 17
Exercises covered in this guide 17
Common terms 17
Create a network and router 18
Create a network 18
Create a router 19

Floating IPs 20
Allocate and Assign Floating IPs using OpenStackClient 20
Associate an instance with a private network 21
Troubleshooting 23
Check neutron docker containers 23
Next Steps 23
Create an OpenStack Instance using Horizon 23
Prerequisites for Creating an Instance 23
SSH key pairs 24
Create and upload an SSH key pair 24
Security groups 25
Create and manage security groups 25
Assign Storage 26
Create and attach a volume 26
Create an instance 26
Steps to create an instance 27
Troubleshooting Instance Error Status 29
Location of OpenStack service logs 29
Next Steps 29
Create an OpenStack Instance with OpenStackClient 29
OpenStackClient 30
Prerequisites for Creating an Instance 30
SSH key pairs 30
Create an SSH key pair 30
Security groups 31
Create and manage security groups 31
Assign Storage 32
Create and attach a volume 32
Create an instance 32
Before creating an instance 32
Procedure to create an instance 34
Troubleshooting 35
Troubleshooting Instance Error Status 35
Location of OpenStack service logs 35
Next Steps 35
SSH into an Instance 36
Access Instance associated with a Public IP 36
Using SSH 36
Using OpenStackClient 36
Access Instance associated with a Private IP 36
Next Steps 37
OpenStack Images 38

List and upload images 38
Using Horizon 38
List images 38
Upload and create images 38
Using the command line 39
List images 39
Upload an image 40
Create Image Snapshot 41
Snapshot using Horizon 41
Snapshot Using the command line 41
Create and verify an instance snapshot 41
Ceph, Cinder and Swift 42
Ceph 42
Ceph Data Durability 43
Cinder 43
More Information 43
What is block storage? 43
Swift 43
More Information 43
What is object storage? 43
How do Cinder and Swift relate to Ceph? 44
Using the Ceph CLI for Common Operational Tasks 44
Common tasks 44
More detailed ``ceph status`` output 44
Ceph Replication, Compression, and Erasure Coding 45
Ceph Replication 45
Setup replication across 2 OSDs 46
Setup replication across 3 OSDs 46
Erasure Coding 46
OpenStack CLI for Common Operational Tasks 46
Background 46
Get started 46
Output formatting 46
Common Tasks 47
Manage OpenStack users 47
Instance Management 48
Create an instance 48
Stop and start an instance 48
Create an instance snapshot 48
Perform live migration of instances 48
Troubleshoot instance issues 50
Upload images 50

Create a private network 51
Create security groups 51
Add SSH public key 52
Collect details about OpenStack environment 52
Backup OpenStack Service and Ceph Configurations 52
Background Information 52
Backup OpenStack services using kolla-ansible 53
Using kolla-ansible 53
Prepare and use kolla-ansible 53
Relevant files 53
Preparation 54
Backup kolla-ansible Files 54
Create OpenStack Database Backups 54
Restore OpenStack Database Backups 55
OpenStack Configuration Backups 56
Ceph Configuration Backups 56
Backing up Client Data and Disaster Recovery 56
Where is client data currently stored? 57
How to create backups of volumes 57
Using Horizon 57
Using OpenStackClient 58
Backing up ceph pool data 59
What to do in the event of a hard drive failure? 59
OpenStack Hardware Failures 59
What should be done should a hardware node fail? 60
Planned maintenance 60
Procedure for removing a compute node 60
Unplanned maintenance 61
Ceph failure scenarios and recovery 61
How do you know if a hard drive has failed? 61
Adding and Removing OpenStack Hardware Nodes 62
Types of Hardware Nodes 62
Adding a node 62
Removing a Storage and Compute Node 63
Requirements before removing a node 63
Remove a Storage and Compute node 63
Procedure 64
Automating OpenStack 69
Miscellaneous Guides 69
Associate an instance with a private network 70

OpenStack Operators Manual - Production in 40 Hours



Private Cloud Core - OpenStack Operator Manual V1.2.1

There are several phases of creating an OpenStack cloud and most of the public documentation focuses on the initial creation of a cloud. With Flex Metal, we provide a full private OpenStack so you will start as an “Operator”. This guide, with a few noted exceptions, applies to any OpenStack that has been provisioned to OpenStack.org’s RefStack standard.

This guide assumes you have followed the Flex Metal Central Process to provision your cloud already. Visit Flex Metal Central to signup or sign in.

For product details please visit the Flex Metal IaaS or Private Cloud pages.

- Start Here

Getting Started with OpenStack

To get started with OpenStack, you will first login to Horizon.

Horizon is a web-based interface used to administer OpenStack services, also known as OpenStack’s dashboard. This is where you will do everything in OpenStack from making an instance, creating networks, managing your users, and so on.

For a full list of the purpose and features of Horizon, see OpenStack’s Horizon documentation.

Login to Horizon

Step 1 – Login to Flex Metal Central

To login to Horizon, you will first need to be logged into Flex Metal Central.

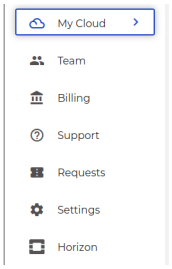
Step 2 – Navigate to the cloud

Once logged in, click on the cloud you are working with to navigate to that cloud's details.

Step 3 – Access Horizon

Next, find and click the **Horizon** link on the left sidebar. It will be the last item in the list.

Horizon link:



Clicking this link will load the Horizon login page.

Step 4 – How to obtain the Horizon password

Initially, you will login to Horizon with the administrator account. The administrator account's username is **admin**.

Next, you will need to obtain the password for the **admin** account. The password is stored in a text file called `/etc/kolla/passwords.yml` on each hardware node.

To acquire the password use SSH to access a hardware node, then within `/etc/kolla/passwords.yml`, find the entry labeled **keystone_admin_password**. The file stores information in a "key: value" format, so the password will be the second entry on that line.

Step 5 – SSH into a hardware node

During cloud setup, you uploaded your SSH public key. This key exists on each hardware node and allows you to login to each node using SSH.

To SSH in, ensure you have one node's IP address, and the private key on the system you will use to access your cloud.

Requirements to SSH into a node:

- Username, which is root
- Authentication is with SSH keys
- An IP address of a hardware node

Here's an example command used to SSH in assuming `~/.ssh/YOUR_KEY` is the path to your private key and a hardware node IP is **50.50.50.50**:

```
$ ssh -i ~/.ssh/YOUR_KEY root@50.50.50.50
```

Step 6 – Obtain Horizon password

Once logged in, search for `keystone_admin_password` inside of `/etc/kolla/passwords.yml`.

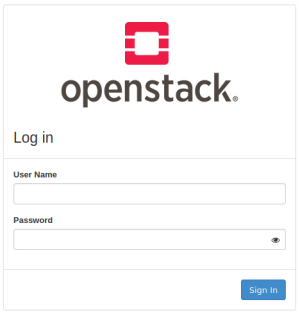
For example:

```
# grep keystone_admin_password /etc/kolla/passwords.yml
keystone_admin_password: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Step 7 – Access Horizon

You can now login to Horizon using the credentials obtained from the previous sections.

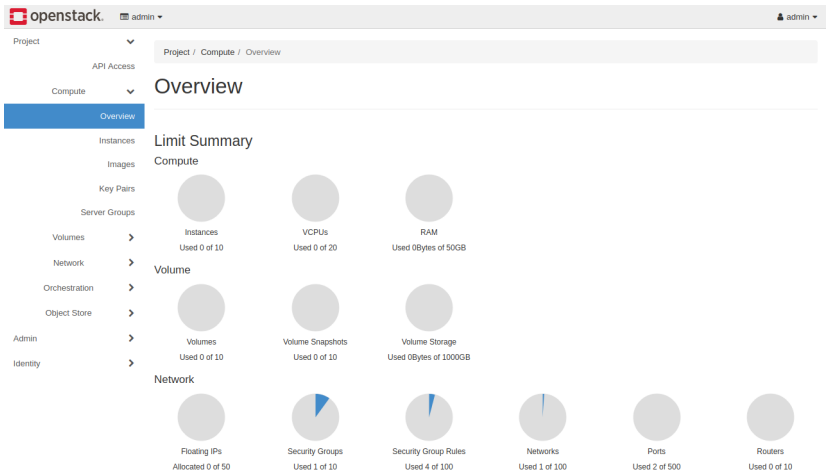
Horizon Login:



Enter in the credentials into the Horizon login page to login.

Once logged in, you should see something similar to:

Initial Horizon login:



Next Steps

The next guide in this series will cover installing OpenStack's command line client, OpenStackClient.

Should using the command line not be necessary for you, skip ahead to the guide that explains how to create Projects and Users.

Install OpenStackClient

OpenStack offers a way to administer your cloud over the command line using OpenStackClient.

If you would like to use OpenStackClient and are not familiar with how to install it, follow along with this installation guide.

Note! – These instructions are meant for a Linux environment.

Installing OpenStackClient

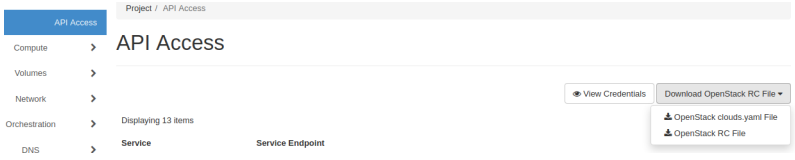
OpenStackClient can be installed to any shell using pip. It can be installed locally to your machine or to an instance in the OpenStack cloud. You could think of the instance that has OpenStackClient installed as a “jumpstation”.

For more information on installing using pip, navigate to OpenStackClient’s pip project page.

It is encouraged to install OpenStackClient using a Python virtual environment.

Once OpenStackClient is installed, you will need a copy of the **OpenStack RC file** as well as the OpenStack **clouds.yaml** file so API requests can be sent to your cloud. Both files can be found in Horizon under the `Project` tab, then `API Access`.

The section to download these files will appear like so:



Click the dropdown near the top right labeled **Download OpenStack RC File** and download the two files listed.

Place the OpenStack RC file in a safe place and `source` it using a shell:

```
$ source openrc.sh
```

Performing this step sets the needed environment variables in your shell so the API can be communicated with.

You will need to enter in your Horizon user’s password upon sourcing this file.

The **clouds.yaml** file should be placed in `~/.config/openstack/clouds.yaml`.

Here is important information that explains where to place this file pulled from the top of an example **clouds.yaml** file:

```
# This is a clouds.yaml file, which can be used by OpenStack tools as a source
# of configuration on how to connect to a cloud. If this is your only cloud,
# just put this file in ~/.config/openstack/clouds.yaml and tools like
# python-openstackclient will just work with no further config. (You will need
# to add your password to the auth section)
# If you have more than one cloud account, add the cloud entry to the clouds
# section of your existing file and you can refer to them by name with
# OS_CLOUD=openstack or --os-cloud=openstack
```

You should have a working OSC to use after these steps.

Here’s example usage of OSC where images are listed:

```
$ openstack image list
+--------------------------------------+---------------------------------+--------+
| ID                                   | Name                            | Status |
+--------------------------------------+---------------------------------+--------+
| 6986aaf9-e602-4307-9c4f-6377795ff890 | CentOS 7 (ce7-x86_64)           | active |
| d3f027f2-730f-4794-9288-44ed5c69050d | CentOS 8 (ce8-x86_64)           | active |
+--------------------------------------+---------------------------------+--------+
```

Next Steps

Navigate to the next guide in this series for a walkthrough of how to manage OpenStack projects and users.

Manage OpenStack Users, Groups and Projects

In OpenStack, managing your users, groups and projects is an important part of setting up the cloud. Projects are an organizational concept that allows a cloud to be subdivided. Users can be created and associated with a single or multiple projects. Groups are sets of users that can be assigned to projects.

Before getting into creating OpenStack components such as networks, instances, and the like, it is suggested you make use of projects so the cloud is organized as needed. In addition, you can create new user accounts and assign them to particular projects so your userbase has the appropriate access to the cloud.

This guide will explain how to create projects, additional user accounts, and groups.

Create and Manage Users

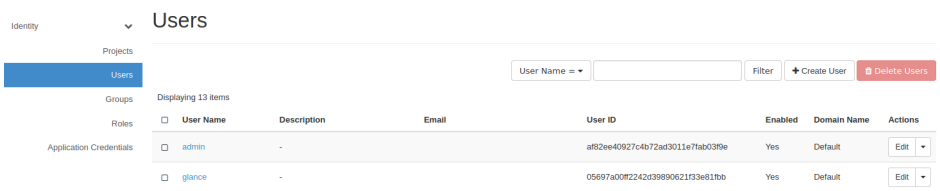
When you start using OpenStack there is only the administrator user. You can think of this as the user “root” in a Linux environment. It has full privileges to the system. Due to this user having full privileges and that there is a chance to cause harm to a system with this user, it is suggested additional users be created as needed. The administrator user should typically be used only for tasks where that level of access is needed.

Create Users Using Horizon

New OpenStack users can be created using Horizon. To create a new user, first ensure you are logged in as an administrator user. Typically, this is the user **admin** if you are just getting started.

Once logged in to Horizon, look for the section called **Identity**, then locate and click the **Users** link under that. This will bring up the page where users can be created.

OpenStack Users:



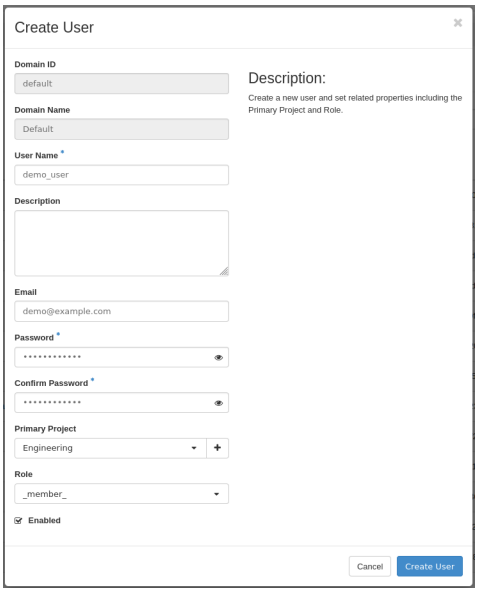
To make a new user, first click the **Create User** button near the top right to bring up the user creation form.

At minimum, you will need to enter a **User Name** and **Password** to create the user. In addition, an **Email** should be set as well.

All users have to be associated with a project, so choose the **Primary Project** this user will be assigned to.

Finally, select an appropriate **Role** for this user. Typically, the role of **member** is sufficient, however you can also assign the **admin** role. The **admin** role gives a user administrator access. In addition, custom roles can be created and assigned.

Create a User form:



With the user created, you should see it in the listing of users now.

User listing:

Create Users Using the Command Line

The base command to create a user using OpenStackClient is:

```
$ openstack user create
```

Generally, when making a user using OpenStackClient, you will need to know the username, email address, and project to assign the user to.

Use `$ openstack project list` to list the project IDs.

List projects:

```
$ openstack project list
+----------------------------------+-------------------------------------------------------------
| ID                               | Name
+----------------------------------+-------------------------------------------------------------
| 0d55c1cd820d4a5d9424456e1384ab73 | Engineering
| 6a654535b8f04445bbc4974b2e4802cd | service
| 80eb7814893a414296ec1464d4a753b1 | b9e8639372014c0b85cbfaffa6e1b5a8-a66df7d2-6e70-493f-922
| b9e8639372014c0b85cbfaffa6e1b5a8 | admin
| c4006f982a2c4f63a2fabeeed6bc9f16 | Project 1
+----------------------------------+-------------------------------------------------------------
```

This example will create a user called **demo_user_cli** and associate its default project to **Project 1**.

NOTE! – Entering passwords over the command line is generally considered insecure. You can pass the flag `--password-prompt` to interactively enter in the password.

Procedure:

The following demonstrates creating the **demo_user_cli** user:

```
$ openstack user create --project c4006f982a2c4f63a2fabeeed6bc9f16 \
--email demo@example.com --password-prompt demo_user_cli
User Password:
Repeat User Password:
```

```
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| default_project_id  | c4006f982a2c4f63a2fabeeed6bc9f16 |
| domain_id           | default                          |
| email               | demo@example.com                 |
| enabled             | True                             |
| id                  | d88a89208d344cb4930761dd55a194d1 |
| name                | demo_user_cli                    |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

List OpenStack users:

```
$ openstack user list
+----------------------------------+-------------------+
| ID                               | Name              |
+----------------------------------+-------------------+
| af82ee40927c4b72ad3011e7fab03f9e | admin             |
| 05697a00ff2242d39890621f33e81fbb | glance            |
| 30c2e20a7c1141dc9fda9d405f1d6db3 | cinder            |
| ec64a60b1c6a4b64a559597417dd3ae2 | placement         |
| 70b677b5fa4b4b8ca55699c8670f7993 | nova              |
| 508ef9606a3a4048a86ec48e542020b4 | neutron           |
| 5fce77bfae1a440d872d96982715af9e | heat              |
| 2da9eb178ee140c7aad2016f8d23ca9e | heat_domain_admin |
| e37f4e048f5e44c69305b6cec9ef2165 | watcher           |
| 012b00425e9e4db289f2d71f6441d835 | swift             |
| b7e2423b016b4defbe5f09ff1b23f468 | demo_user         |
| d88a89208d344cb4930761dd55a194d1 | demo_user_cli     |
+----------------------------------+-------------------+
```

From the above output, the **demo_user_cli** user is listed now.

Assign Role to a User

NOTE! – This section still needs to be filled out completely.

In OpenStack, there are roles that can be assigned to users and groups.

To view the current roles, use:

```
$ openstack role list
+----------------------------------+------------------+
| ID                               | Name             |
+----------------------------------+------------------+
| 3e5d1f2c2c014bf0bfa929b0e31eb2b1 | heat_stack_owner |
| 764cb7fcf8214515860c628fcfb855d2 | admin            |
| aa2689853f7b42038afcdb797b54ef11 | heat_stack_user  |
| be24b48c21554b75849c66b9b710df84 | reader           |
| ccc32facd900440bb01a046db5c4096d | member           |
| f5c0b887144d462bbd3bc35e9a0a9309 | _member_         |
+----------------------------------+------------------+
```

Create and Manage Groups

Groups in OpenStack are collections of Users. These can be assigned to projects and make it easier to assign a grouping of users.

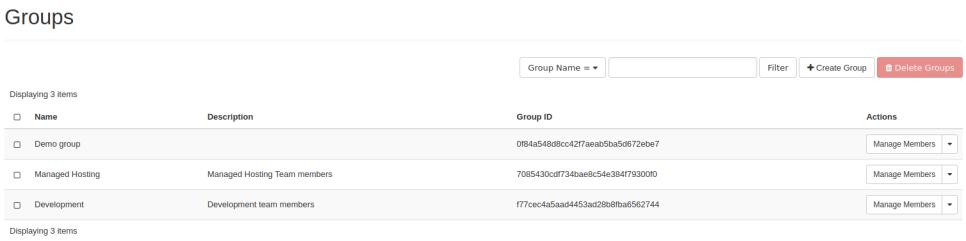
Create Groups Using Horizon

To make a group, you will need to start with logging into to Horizon with an administrator account.

From there, on the left, find the **Identity** tab, then find the **Groups** tab within that. Following this will take you to the section where groups in OpenStack can be managed.

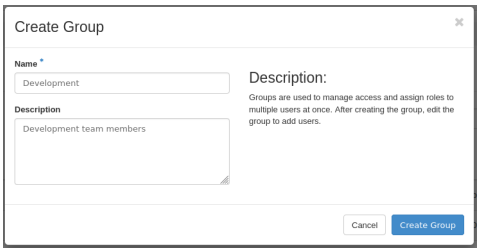
The following is a screenshot of how the **Groups** page will appear.

Group listing:



Next, find the **Create Group** button near the top right. This will load the form needed to create a group.

Create group:



Fill in the **Name** of the group and, optionally, a **Description**. Once the form has been submitted, the group will be created.

Add Users to a Group Using Horizon

Now that a group has been made, users can be added to it.

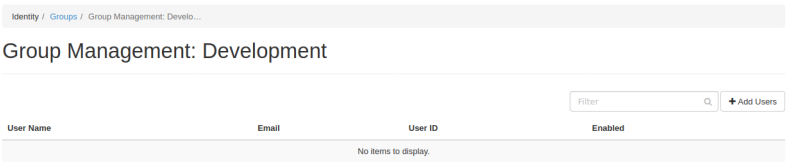
To add users, pull up the listing of Groups in OpenStack first.

From there, find the group you are working with, and click **Manage Members** in the far right **Actions** column.

Manage Members:

Clicking **Manage Members** will pull up another page.

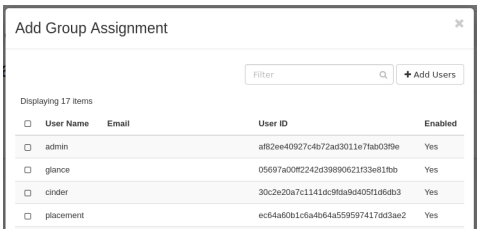
Manage Members, page 2:



To add users, click the **Add Users** button near the top right.

A new form will appear providing a list of users that can be added.

Manage Members -> Add Users Form:



Create Groups Using the Command Line

The base command to create a group using OpenStackClient is:

```
$ openstack group create
```

This section will demonstrate creating a group called **demo_group**.

Procedure

Use `$ openstack group create demo_group` to create the group:

```
$ openstack group create demo_group
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | default                          |
| id          | 5d1177ede33b4cddadab6579408da7d7 |
| name        | demo_group                       |
+-------------+----------------------------------+
```

List groups:

```
$ openstack group list
+----------------------------------+------------------+
| ID                               | Name             |
+----------------------------------+------------------+
| 5d1177ede33b4cddadab6579408da7d7 | demo_group       |
| 7085430cdf734bae8c54e384f79300f0 | Managed Hosting  |
| f77cec4a5aad4453ad28b8fba6562744 | Development      |
+----------------------------------+------------------+
```

Add Users to a Group Using the Command Line

Users can be added to groups using OpenStackClient. This will show an example where the user **demo_user_cli** is added to the group **demo_group**.

The base command to add a user to a group is:

```
$ openstack group add user
```

Procedure

Add the user **demo_user_cli** to the group **demo_group**:

```
$ openstack group add user demo_group demo_user
```

Confirm the user was added successfully using `openstack group contains user`:

```
$ openstack group contains user demo_group demo_user
demo_user in group demo_group
```

Create and Manage Projects

As an OpenStack administrator, it is typically advised that projects be created for specific uses. For example, you may want a project for development purposes, or need one for a specific department in your organization.

NOTE! – Projects can only be created by OpenStack accounts with **administrator** access.

This section will demonstrate how to create and manage projects using Horizon and the command line.

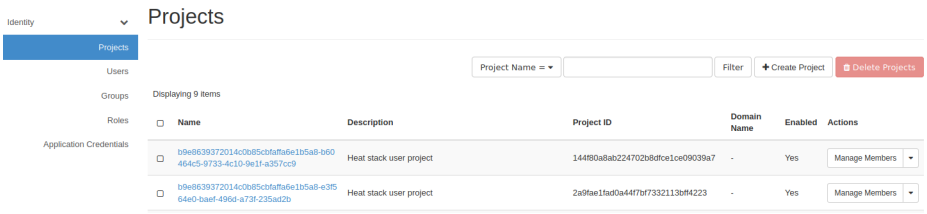
Create Projects Using Horizon

To create a project, you will want to start with being logged into Horizon as an administrator account.

On the left, look for the section called **Identity**, then click on the **Projects** link under that. This page is where projects in OpenStack are managed.

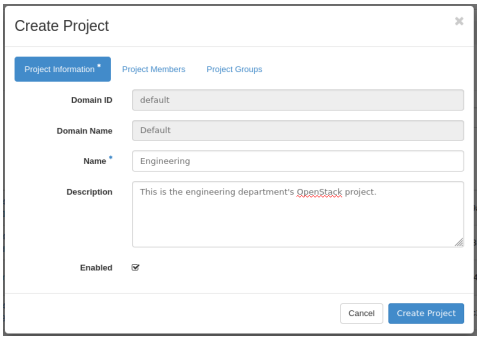
To make a new project, say for the Engineering team in this example, find and click the **Create Project** button near the top right.

Create Project:



A form will display where the project's details will be needed.

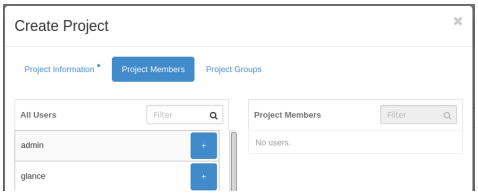
Create Project Form:



On the first tab, enter a **Name** and a **Description** for the project.

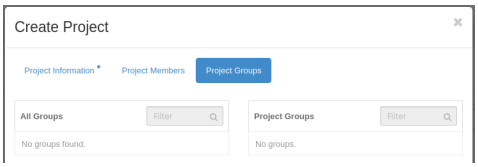
The second tab, **Project Members**, allows you to add and remove members to this project. Here is where you will add the needed users to this project.

Create Project Form, page 2:



Finally, the third tab has to do with adding groups to this project. If you have a group to add, this is where it will be done.

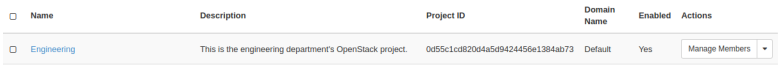
Create Project Form, page 3:



Once all the details are filled in, click the **Create Project** button to create the project.

You should see the newly created project listed under the **Projects** page.

Engineering project created:



Create Projects Using the Command Line

The base command to create a project using OpenStackClient is:

```
$ openstack project create
```

This section will details the steps needed to create a project called **demo_project**.

Procedure

Create a project called **demo_project**:

```
$ openstack project create demo_project
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | default                          |
| enabled     | True                             |
| id          | a7873c1cbbe14607b5c5e797ef8d56ba |
| is_domain   | False                            |
| name        | demo_project                     |
| options     | {}                               |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
```

Confirm the project was created successfully using `openstack project show demo_project`:

```
$ openstack project show demo_project
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | default                          |
| enabled     | True                             |
| id          | a7873c1cbbe14607b5c5e797ef8d56ba |
| is_domain   | False                            |
| name        | demo_project                     |
| options     | {}                               |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
```

Add Group to Project Using the Command Line

Now that a group and project have been made, the group can be added to the project.

This section will demonstrate adding the group **demo_group** to the project called **demo_project**.

The base command to add a group to a project is:

```
$ openstack role add
```

Procedure

Add group **demo_group** to the project **demo_project**:

```
$ openstack role add --project demo_project --group demo_group \
f5c0b887144d462bbd3bc35e9a0a9309
```

Verify the group was added to the project using `openstack role assignment list`:

```
$ openstack role assignment list --group demo_group --project demo_project --names
+----------+------+--------------------+----------------------+--------+--------+-----------
| Role     | User | Group              | Project              | Domain | System | Inherited
+----------+------+--------------------+----------------------+--------+--------+-----------
| _member_ |      | demo_group@Default | demo_project@Default |        |        | False
+----------+------+--------------------+----------------------+--------+--------+-----------
```

Next Steps

With this topic covered, the next guide will explain how to create networks using Horizon in OpenStack.

Alternatively, if prefer working over the command line, see the how to create networks using OpenStackClient guide.

Networks, Routers, and Floating IPs

This guide will explain basic networking functions in OpenStack including how to create a network, a router, and allocate and assign floating IPs.

The idea of this guide is it will explain how to create a private network with the intent to place virtual machines or instances on this network.

It is generally recommended that private networks be used where possible and to only expose the portions of your cloud to a public network when needed.

Neutron is the name of the service that handles networking in OpenStack. It provides “network connectivity as a service” between interfaces and uses the OpenStack Networking API.

Neutron allows networks, routers, floating IPs, and security groups to be created.

Exercises covered in this guide

This guide will explain how to:

- Create a private network
- Create a router
- Associate an instance with a private network
- Allocate floating IPs
- Assign a floating IP to an instance

Common terms

- `Provider Network`
    - a network that has been mapped to physical networking devices
    - this network comes already setup and is Internet-accessible
- `Floating IP`
    - public facing and allows external communication
    - attach to an instance on a private network to allow access to the Internet
    - allocated from the provider network
- `Port`
    - typically created as a result of another action (creating an instance)
    - associated with instances, routers, floating IPs, and essentially anything that can be connected to a network

Create a network and router

Networks and routers can be created in OpenStack. To make a private network accessible from the provider network, a router must be created.

This section will go over how to make a network and router in Horizon, then connect the public network to a private network using the router.

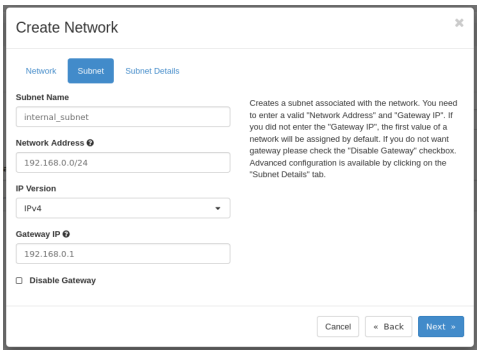
Create a network

To make a network in Horizon, find the **Network** tab on the left, then navigate to the **Networks** tab under that. Finally, locate the **Create Network** button near the top right. You'll be presented with the form to create a network.

Create a Network:

Specify a name for the network under **Network Name** on the first page. In this example, the network will be called **Internal**. Fill out any other needed details and navigate to the **Subnet** tab.

Create a Subnet:



- Under **Subnet Name** specify a name for the subnet
- For **Network Address** choose a network in CIDR notation. This example uses **192.168.0.0/24**.
- Finally under **Gateway IP** specify the gateway IP for this network. If the gateway IP is not filled out, one will be chosen by the neutron service.

  The final tab is called **Subnet Details**. This tab does not need to be filled out to create the network and subnet. This example will stop here, however on this page you can enable or disable DHCP, specify specific IPs to be allocated, set DNS name servers, and set Host Routes.

With the previous steps done, the network has been created. Loading the **Network -> Networks** tab will display the new network:

Internal network created:

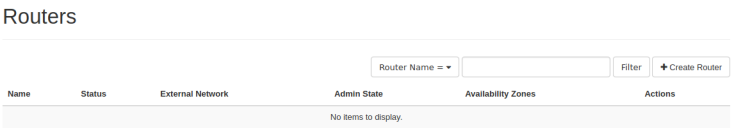
Create a router

With a network created, the next step is to creater a router which will bridge the connection from the **External** or **provider network** to the private network.

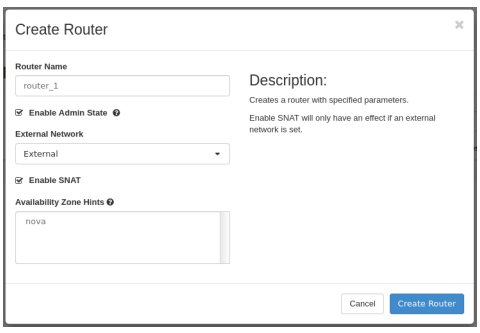
To make a router in Horizon, find the **Network** tab on the left, then locate the **Routers** tab under that. This page will list current routers and allows you to create a router.

To create a new router, click the **Create Router** button near the top right.

Create a router:



Create a router form:

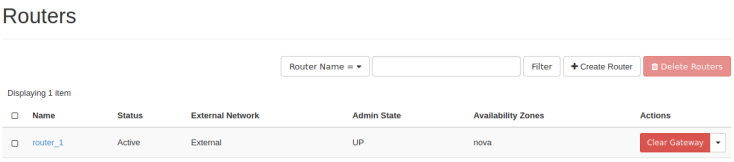


Under **Router Name** choose a name for the router. This example router will be called **router_1**.

The router will need to be connected to an external network, which will be the provider network. The network called **External** will be used.

Once created it will show in the list of routers.

Router listing:



This takes care of creating the router with the **External** network attached. To allow the router to communicate with the **External** and the **Internal** network, the **Internal** network will need to be attached. The remaining steps demonstrate how to do this.

Attach **Internal** network

First, pull up the newly created router from the **Network -> Routers** section of Horizon.

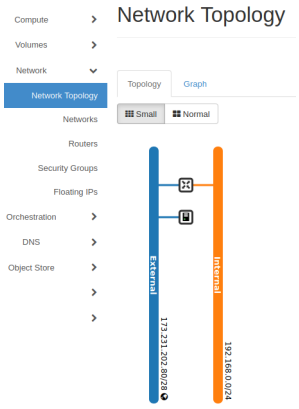
Next, to add an interface to this router, click the router's name in the listing of routers, then click the **Add Interface** button near the top right.

Add Interface to Router:

In the form that appears choose the subnet to connect the router to and optionally choose an IP address for the interface. If you don't choose an IP, the gateway IP of the subnet will be used.

With the interface added to the router, the **External** and **Internal** networks are now connected. This can visually be seen by navigating to the **Network** tab on the left, then the **Network Topology** tab under that.

Network Topology:



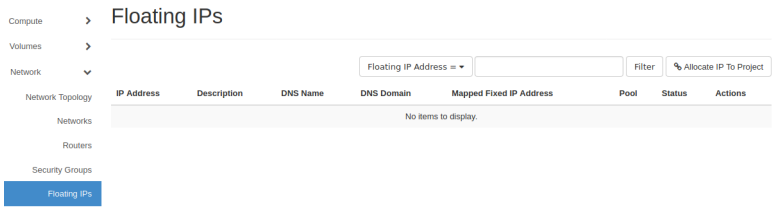
Floating IPs

Floating IPs in OpenStack are publicy routable IP addresses that can be attached and detached to instances. For example if there's an instance associated with a private network but needs to be accessed from the Internet, a floating IP can be associated with the instance, allowing communication from the Internet.

Allocate and Assign Floating IPs using Horizon

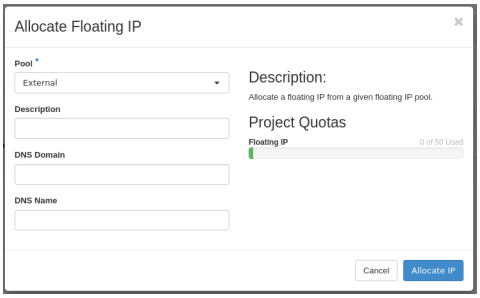
To use Floating IPs they will first need to be allocated from the provider network's pool of IPs.

To allocate floating IPs in Horizon, navigate to the **Network** tab on the left, and look for **Floating IPs**.



Click the **Allocate IP To Project** to allocate a new IP.

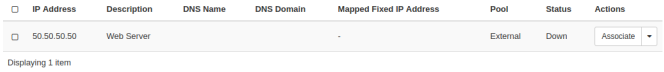
Allocate IP To Project:



The IP will be obtained from the provider network. In this example, the name of that network is **External**.

Once the IP is added, it will appear in the floating IP list.

Floating IP list:



Now that an IP has been allocated, it can be assigned to an instance. Use the **Associate** button from the drop down on the right.

Associate floating IP:

Choose the floating IP and the port to associate it to.

Troubleshooting

NOTE! – This section will be need to be updated and is incomplete.

Check neutron docker containers

In Private Cloud Core, OpenStack has been deployed using `kolla-ansible`. This means each service is running in a Docker container on the hardware nodes.

You can check the status of the neutron Docker containers to see if they are running:

```
# docker ps | grep -i neutron
66c57a96ecf5        kolla/centos-binary-neutron-metering-agent:train-centos8      "dumb-init
b407dda02505        kolla/centos-binary-neutron-metadata-agent:train-centos8      "dumb-init
93ce576891db        kolla/centos-binary-neutron-l3-agent:train-centos8            "dumb-init
5531d47d6405        kolla/centos-binary-neutron-dhcp-agent:train-centos8          "dumb-init
d3e20fd98aef        kolla/centos-binary-neutron-openvswitch-agent:train-centos8   "dumb-init
2802a09606e5        kolla/centos-binary-neutron-server:train-centos8              "dumb-init
```

Next Steps

With this guide complete, move on to the next guide, which explains how to create an Instance in Horizon.

Alternatively, if you prefer working with the command line, see the guide that explains how to create an Instance using OpenStackClient.

Create a Network using OpenStackClient

This guide will explain basic networking functions in OpenStack including how to create a private network, a router, and allocate and assign floating IPs.

Neutron is the name of the service that handles networking in OpenStack. It provides “network connectivity as a service” between interfaces and uses the OpenStack Networking API.

Neutron allows networks, routers, floating IPs, and security groups to be created.

Exercises covered in this guide

This guide will explain how to:

- Create a private network
- Create a router
- Associate an instance with a private network network
- Allocate floating IPs
- Assign a floating IP to an instance

Common terms

- `Provider Network`
    - a network that has been mapped to physical networking devices
    - this network comes already setup and is Internet-accessible

- `Floating IP`
    - public facing and allows external communication
    - attach to an instance on a private network to allow access to the Internet
    - allocated from the provider network
- `Port`
    - typically created as a result of another action (creating an instance)
    - associated with instances, routers, floating IPs, and essentially anything that can be connected to a network

---

Create a network and router

Networks and routers can be created in OpenStack. To make a private network accessible from the provider network, a router must be created.

This will go over how to make a network and router using the command line with OpenStackClient.

Create a network

Listed are the steps needed to create a private network. Variables are presented in all capital and should be replaced accordingly. Note the output of most of the commands has been truncated.

Use this command to create a network, replacing NETWORK_NAME with the name of the network:

```
$ openstack network create NETWORK_NAME
```

Create a network called **private_network**:

```
$ openstack network create private_network --fit-width
+---------------------------+--------------------------------------------------------------------
| Field                     | Value
+---------------------------+--------------------------------------------------------------------
| admin_state_up            | UP
| availability_zone_hints   |
| availability_zones        |
| created_at                | 2020-10-30T20:57:16Z
| description               |
| dns_domain                |
| id                        | 890e5ab6-a5d7-44c2-bf84-351342006cdd
```

Next, a subnet will need to be created.

Use this command to create a subnet, replacing NETWORK_NAME and SUBNET_NAME with the respective names of the network and subnet and replace SUBNET_RANGE with the subnet to use. An example subnet range could be **10.0.0.0/24**:

```
$ openstack subnet create --subnet-range SUBNET_RANGE --network NETWORK_NAME \
SUBNET_NAME
```

Create a subnet called **private** with subnet range of **10.0.0.0/24**:

```
$ openstack subnet create --subnet-range 10.0.0.0/24 --network private_network private --fit
+----------------------+--------------------------------------------------------------------
| Field                | Value
+----------------------+--------------------------------------------------------------------
| allocation_pools     | 10.0.0.2-10.0.0.254
```

```
| cidr                 | 10.0.0.0/24
| created_at           | 2020-10-30T21:03:50Z
| description          |
| dns_nameservers      |
| dns_publish_fixed_ip | None
| enable_dhcp          | True
| gateway_ip           | 10.0.0.1
| host_routes          |
| id                   | 46d0b88c-f1cf-4e40-a395-9281a7dd59d9
```

Create a router

The following are the commands required to create a router.

To make a router, use this base command, replacing ROUTER_NAME with the name of the router:

```
$ openstack router create ROUTER_NAME
```

Create a router called **router_1**:

```
$ openstack router create router_1 --fit-width
+-------------------------+-------------------------------------------------------------------------
| Field                   | Value
+-------------------------+-------------------------------------------------------------------------
| admin_state_up          | UP
| availability_zone_hints |
| availability_zones      |
| created_at              | 2020-10-30T21:08:38Z
| description             |
| distributed             | False
| external_gateway_info   | null
| flavor_id               | None
| ha                      | True
| id                      | a700e1b1-36ac-4e4f-b4c7-fa80be513656
```

With the router created, a subnet needs to be attached to it along with the external or, provider network.

To add a subnet, use this command, replacing ROUTER_NAME and SUBNET_NAME with the names of the respective router and subnet:

```
$ openstack router add subnet ROUTER_NAME SUBNET_NAME
```

Add subnet **private** to the router called **router_1**:

```
$ openstack router add subnet router_1 private
```

The command to add the subnet to the router returns no output if successful.

Finally the router also needs the external network connected to it.

Use this command to add an external network, replacing EXTERNAL_NETWORK_UUID with the UUID of the network:

```
$ openstack router set --external-gateway EXTERNAL_NETWORK_UUID \
ROUTER_NAME
```

You can get the UUID of the network to use by running `openstack network list`. The UUID will be listed in the first column.

Add the external network to the router called **router_1**:

```
$ openstack router set --external-gateway \
55d31bd5-77ba-4ed0-ab6e-99554b33aa90 router_1
```

With these steps completed, you have a router that connects the external network to the private network.

You can see the details of the router by running this, replacing ROUTER_NAME with the name of the router:

```
$ openstack router show ROUTER_NAME
```

Show the details for the router called **router_1**, including the interfaces that were previously attached:

```
$ openstack router show router_1 --fit-width
+-------------------------+-------------------------------------------------------------------------
| Field                   | Value
+-------------------------+-------------------------------------------------------------------------
| admin_state_up          | UP
| availability_zone_hints |
| availability_zones      | nova
| created_at              | 2020-10-30T21:08:38Z
| description             |
| distributed             | False
| external_gateway_info   | {"network_id": "55d31bd5-77ba-4ed0-ab6e-99554b33aa90", "external
|                         | [{"subnet_id": "ca55b96b-0d70-4cbc-92fc-7bf5ce74cfa4", "ip_addre
|                         | "173.231.202.88"}], "enable_snat": true}
| flavor_id               | None
| ha                      | True
| id                      | a700e1b1-36ac-4e4f-b4c7-fa80be513656
| interfaces_info         | [{"port_id": "8c45b709-9f74-486d-bb13-0d8b459066dd", "ip_address
|                         | "subnet_id": "52df9b51-8a5a-45f0-bf22-693eb9712c32"}, {"port_id"
|                         | "d0233252-3756-44b2-8bad-e3e82781f96d", "ip_address": "169.254.1
|                         | "52df9b51-8a5a-45f0-bf22-693eb9712c32"}, {"port_id":
|                         | "e4bd63c8-8272-4285-85b2-f98e9f028471", "ip_address": "169.254.1
|                         | "52df9b51-8a5a-45f0-bf22-693eb9712c32"}, {"port_id":
|                         | "e8d5407c-1574-45e4-8f04-77dc1a848591", "ip_address": "10.0.0.1"
|                         | "46d0b88c-f1cf-4e40-a395-9281a7dd59d9"}]
```

Floating IPs

Floating IPs in OpenStack are publicy routable IP addresses that can be attached and detached to instances. For example if there's an instance associated with a private network but needs to be accessed from the Internet, a floating IP can be associated with the instance, allowing communication from the Internet.

Allocate and Assign Floating IPs using OpenStackClient

OpenStackClient can be used to manage Floating IPs. The following is a list of commands used to manage floating IPs.

Allocate additional floating IPs where `NETWORK` is the UUID of the network to allocate IPs from:

```
$ openstack floating ip create NETWORK

$ openstack floating ip create 55d31bd5-77ba-4ed0-ab6e-99554b33aa90 --fit-width
+---------------------+-------------------------------------------------------------------------
| Field               | Value
+---------------------+-------------------------------------------------------------------------
| created_at          | 2020-10-29T20:05:53Z
| description         |
| dns_domain          |
```

```
| dns_name            |
| fixed_ip_address    | None
| floating_ip_address | 50.50.50.50
| floating_network_id | 55d31bd5-77ba-4ed0-ab6e-99554b33aa90
| id                  | e1818df0-ce58-4f36-94ff-170a5a0c1f9f
```

Some of the output is truncated.

View floating IPs:

```
$ openstack floating ip list --fit-width
+----------------------+---------------------+------------------+------+----------------------
| ID                   | Floating IP Address | Fixed IP Address | Port | Floating Network
+----------------------+---------------------+------------------+------+----------------------
| e1818df0-ce58-4f36-  | 50.50.50.50         | None             | None | 55d31bd5-77ba-4ed0-
| 94ff-170a5a0c1f9f    |                     |                  |      | ab6e-99554b33aa90
+----------------------+---------------------+------------------+------+----------------------
```

Assign floating IPs to an instance:

```
$ openstack floating ip set FLOATING_IP
```

Associate an instance with a private network

This section will explain how to take an instance created on the provider network and associate that with a private network. This may not be a very common task assuming an instance was created on the appropriate network to begin with.

In the event an instance was created but on the wrong network, you can have it associated with another network.

The commands needed to add and remove networks an instance is associated with are:

- `openstack server add network SERVER NETWORK`
- `openstack server remove network SERVER NETWORK`

SERVER and NETWORK are placeholders and should be replaced with the UUIDs of the server and network needed.

To change the network for instance, some information is needed first:

- Instance UUID
- UUID of network to which the instance is moving
- UUID of network from which the instance is moving

To obtain the UUID of the instance use:

```
$ openstack server list
```

Obtain UUID of instance called **Server 1**:

```
$ openstack server list --fit-width
+--------------------------------------+----------+--------+-------------------------+------
| ID                                   | Name     | Status | Networks                | Image
+--------------------------------------+----------+--------+-------------------------+------
| 4208022c-3afa-4233-84ed-8df04bb2c4ce | Server 1 | ACTIVE | External=173.231.202.87 |
+--------------------------------------+----------+--------+-------------------------+------
```

This instance is shown to be on the **External** network and instead it should be associated with another network, called **private_network**.

To move this instance to the **private_network**, the UUID of both networks is needed.

Use the following to list network UUIDs:

```
$ openstack network list
```

List the UUID of each network:

```
$ openstack network list --fit-width
+--------------------------------------+--------------------------------------+------------
| ID                                   | Name                                 | Subnets
+--------------------------------------+--------------------------------------+------------
| 3eb74273-45aa-45e7-977e-3c706c0499fb | Internal                             | c6bb7ad7-79
| 55d31bd5-77ba-4ed0-ab6e-99554b33aa90 | External                             | ca55b96b-0d
| 8346bc1d-2b69-421d-ab7f-008b9af53c5d | HA network tenant                    | 52df9b51-8a
| 890e5ab6-a5d7-44c2-bf84-351342006cdd | private_network                      | 46d0b88c-f1
+--------------------------------------+--------------------------------------+------------
```

From this output, the UUIDs can be obtained.

Next the instance will need to be associated with the network called **private_network**.

Associate **Server 1** with **private_network** using the UUIDs from above:

```
$ openstack server add network 4208022c-3afa-4233-84ed-8df04bb2c4ce \
890e5ab6-a5d7-44c2-bf84-351342006cdd
```

Disassociate **Server 1** from the **External** network:

```
$ openstack server remove network 4208022c-3afa-4233-84ed-8df04bb2c4ce \
55d31bd5-77ba-4ed0-ab6e-99554b33aa90
```

Both the add and remove network commands do not return any output if they are successful.

To be sure the network has been associated correctly, use `openstack server show SERVER_UUID` to confirm, replacing SERVER_UUID with the UUID of the server you are working with.

Confirm new network for **Server 1**:

```
$ openstack server show 4208022c-3afa-4233-84ed-8df04bb2c4ce --fit-width
+-------------------------------------+------------------------------------------------------
| Field                               | Value
+-------------------------------------+------------------------------------------------------
| OS-DCF:diskConfig                   | AUTO
| OS-EXT-AZ:availability_zone         | nova
| OS-EXT-SRV-ATTR:host                | hc1.example.com
| OS-EXT-SRV-ATTR:hypervisor_hostname | hc1.example.com
| OS-EXT-SRV-ATTR:instance_name       | instance-00000096
| OS-EXT-STS:power_state              | Running
| OS-EXT-STS:task_state               | None
| OS-EXT-STS:vm_state                 | active
| OS-SRV-USG:launched_at              | 2020-10-23T20:52:15.000000
| OS-SRV-USG:terminated_at            | None
| accessIPv4                          |
| accessIPv6                          |
| addresses                           | private_network=10.0.0.192
| config_drive                        |
| created                             | 2020-10-23T20:51:27Z
| flavor                              | hc1.small (hc1.small)
| hostId                              | fc32f234767ef0316a30779802b178074140cb60635ff81fa1cd
| id                                  | 4208022c-3afa-4233-84ed-8df04bb2c4ce
```

The **addresses** field confirms this instance is now associated with the network called **private_network**.

Troubleshooting

NOTE! – This section will be need to be updated and is incomplete.

Check neutron docker containers

In Private Cloud Core, OpenStack has been deployed using `kolla-ansible`. This means each service is running in a Docker container on the hardware nodes.

You can check the status of the neutron Docker containers to see if they are running:

```
# docker ps | grep -i neutron
66c57a96ecf5        kolla/centos-binary-neutron-metering-agent:train-centos8      "dumb-init
b407dda02505        kolla/centos-binary-neutron-metadata-agent:train-centos8      "dumb-init
93ce576891db        kolla/centos-binary-neutron-l3-agent:train-centos8            "dumb-init
5531d47d6405        kolla/centos-binary-neutron-dhcp-agent:train-centos8          "dumb-init
d3e20fd98aef        kolla/centos-binary-neutron-openvswitch-agent:train-centos8   "dumb-init
2802a09606e5        kolla/centos-binary-neutron-server:train-centos8              "dumb-init
```

Next Steps

The next guide will explain how to make an instance.

With this guide complete, you should have all that is needed to create an instance.

Navigate to the create an instance with the command line guide to continue this series.

Should you want to learn to the make the instance using Horizon, see the create an instance using Horizon guide.

Create an OpenStack Instance using Horizon

The purpose of this guide is to explain how to create an instance using Horizon. An example will be demonstrated where an instance is created on a private network. The intent of the instance will be to act as a “jumpstation” where OpenStackClient will be installed. Finally, this jumpstation can be used to administer the OpenStack cloud using OpenStackClient.

You’ll also learn how to upload or create an SSH key pair, assign storage using a volume, and create a security group. Each of these components will be added to the instance.

Make sure you are logged into Horizon when following this guide. If you are not familiar with accessing Horizon, see the Getting Started guide.

Prerequisites for Creating an Instance

Before making an instance, the following should exist:

- A flavor
- An image
- A network
- A security group
- An SSH public key

At minimum, a flavor, an image, and a network should exist. You may also want to attach a volume, specify security groups to use, and attach an SSH public key.

SSH key pairs

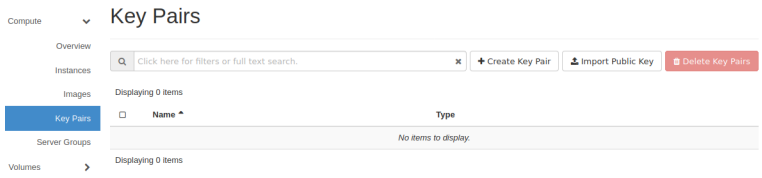
An SSH key pair will be required to access any instances over SSH. Password authentication is by default disabled in the operating system images.

You can either have an SSH key created or you can upload the public key of your SSH key pair.

Create and upload an SSH key pair

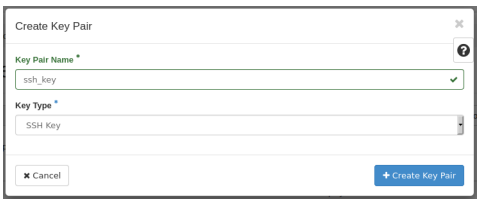
To make an SSH key pair, start in the main dashboard of Horizon. On the left, click the **Compute** heading. Following that, find the **Key Pairs** link under **Compute**. Clicking this link takes you to the SSH Key Pairs section.

SSH key pair section:



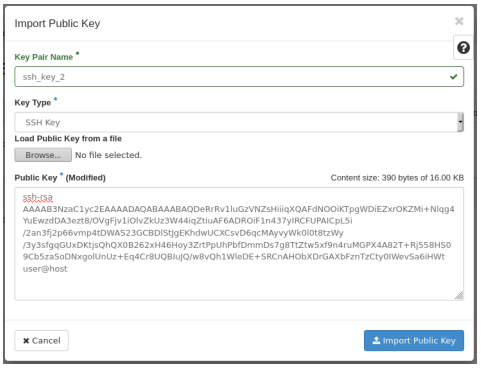
To make an SSH key pair, click **Create Key Pair** near the top and to the right. You'll need to fill out the key pair name and the type of key it will be. Once done, you'll be prompted to download the private key which needs to be stored in a safe place. Only you should have access to your private SSH key pair! The public key portion of the key pair now exists in Horizon.

Create SSH key pair form:

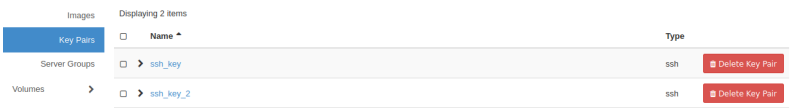


Should you already have an SSH key pair you want to use, you can instead upload your public key. This can be done in the same **Key Pairs** section. Locate the **Import Key Pair** button near the top right of the screen. You'll need to fill out the name of the key pair, specify the type of key it is, and finally choose the public key from your computer or paste the public key in.

Upload SSH key pair form:



After an SSH key has been created or uploaded, they should appear in the list like so:



Security groups

A security group in OpenStack controls network access. If no security group is assigned, a default one will be used. The default security group disables all inbound traffic and only allows outbound traffic, which may not be useful.

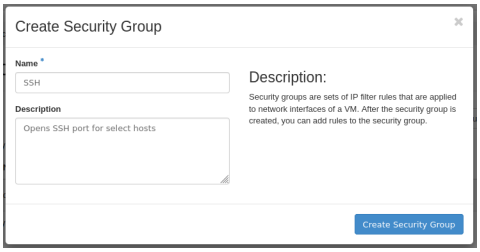
In Horizon, security groups can be found on the left, under **Network**, then **Security Groups**. You can create and modify security groups using this section.

A security group should be made that allows the type of incoming and outgoing traffic required. OpenStack's default security group does not allow incoming traffic so if you wanted to SSH into an instance, port 22 will be closed. An example of a security group would be one that allows web traffic by opening ports 80 and 443 and you may want another security group that opens port 22 for SSH traffic.

Create and manage security groups

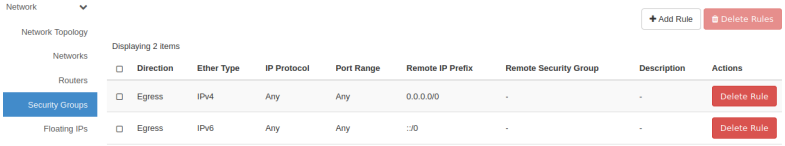
In Horizon, on the main landing page, look for the **Network** tab, then find **Security Groups**. Click + Create Security Group to load the form to create a Security Group.

Create Security Group form:



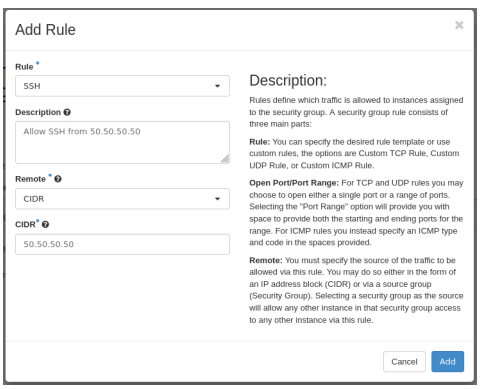
Fill out the name and description of the group. Once done, you'll be taken to a page where you can define egress (outbound) and ingress (inbound) rules for the group.

Manage security group rules:



Let's say you want to allow incoming SSH traffic from `50.50.50.50`. To do so, click the **Add Rule** button near the top right.

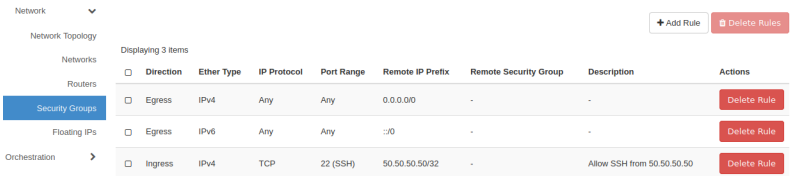
Add Rule:



From here you can specify a custom rule using the various options, or you can use the first drop down to specify common protocols, such as SSH.

To add the rule for SSH, select from the first drop down the **SSH** option. You can describe the rule using the next box. The third field called **Remote** should be left as **CIDR**. Finally the last box needs to be filled in with the IP, `50.50.50.50`. You can specify a range of IPs using CIDR notation as well.

Newly added SSH rule:



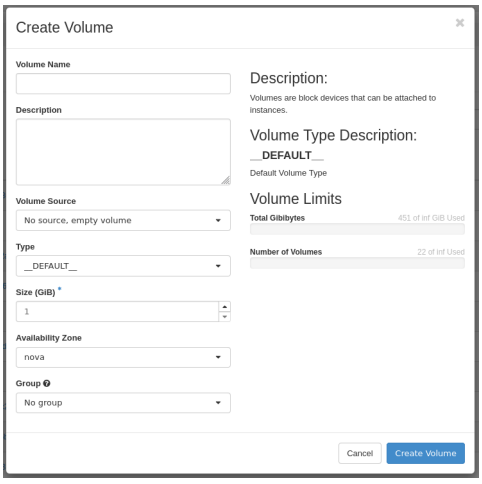
Assign Storage

Storage can be assigned through volumes. A volume in OpenStack is like a removable USB drive that can be attached to instances as seen fit. Cinder is the OpenStack service that allows volumes to be created. A volume can also be used to boot an instance.

Create and attach a volume

On the main page of Horizon, on the left, under the **Volumes** tab, look for the **Volumes** sub tab. From here you may see a list of volumes and can also create one. To create a volume, click the + Create Volume icon.

Create volume form:



Minimally, the size of the volume needs to be specified. All other options are optional.

There are several sources that can be used for volumes, including other volumes, images, and snapshots.

To create a new empty volume, specify “No source, empty volume” under the **Volume Source** option.

Create an instance

An instance is another name for a virtual machine in OpenStack. Instances are created by the Nova service and contribute to the processing power of the cloud.

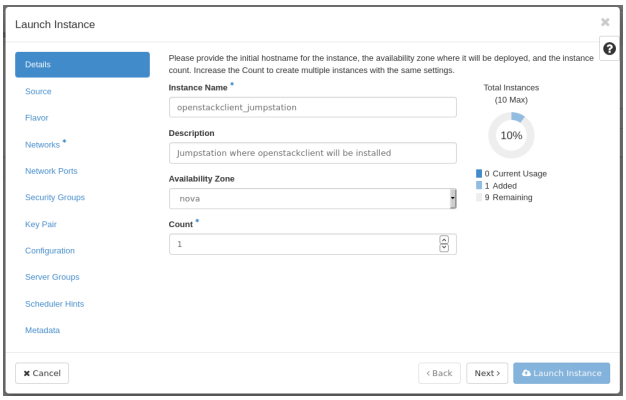
With the previous steps followed, you should have everything needed to make an instance.

This section will demonstrate creating a “jumpstation” instance on a private network.

Steps to create an instance

In Horizon, look for the **Compute** heading on the left. From there, choose **Instances**. This displays the Instances page. Click the Launch Instance button to create one.

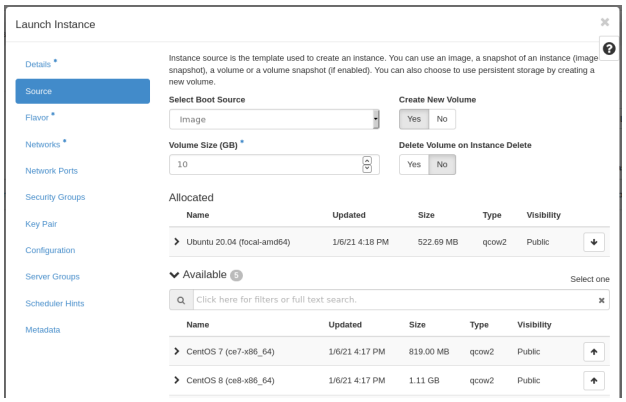
Launch Instance



The minimal requirements needed to launch an instance are marked by an asterisk.

After the **Details** section is filled out, move on to the **Source** tab on the left. Here you will specify a source to boot from, which is typically going to be an image.

Choose Source



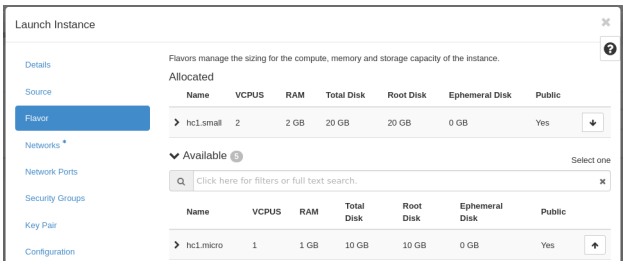
You need to choose a **Boot Source**, specify whether or not to create a volume, and finally the image source.

Here, the **Boot Source** is selected as **Image** and a new Volume of size 10GB will be created. The images are listed on this same form and you can use the up arrow on the right to choose the image.

Next you need to specify a flavor. The flavor is a way to define resource allocation to an instance. The number of vCPUs, RAM, and disk space are defined using flavors.

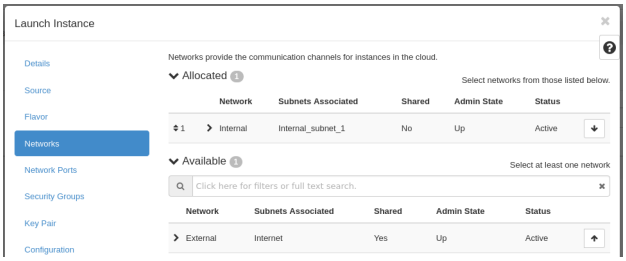
This example has selected the **hc1.small** flavor.

Choose Flavor



Next, for the **Network** select the private network created in the previous guide or choose your own private network.

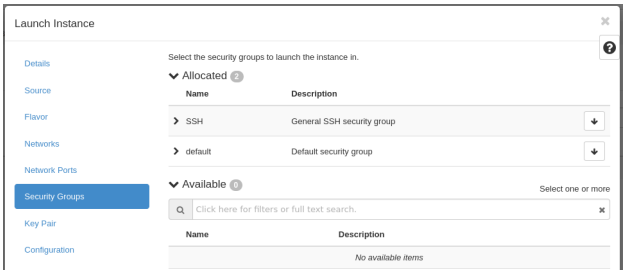
Choose Network



Here, the **Internal** Network is the network this instance will be associated with. Neutron will obtain an IP from that network and assign it to this instance.

After the **Network** section, you'll want to be sure you have a **Security Group** that matches your needs. This example will use the **default** security group and in addition will use one made previously, called **SSH**.

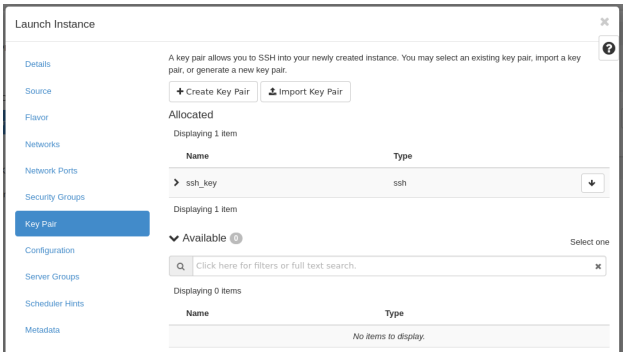
Choose Security Group



By default, only the **default** security group will be selected. To set the **SSH** security group, choose the up arrow on the right of the **SSH** security group listing.

Finally, an SSH key pair will be added to the instance.

Choose SSH Key

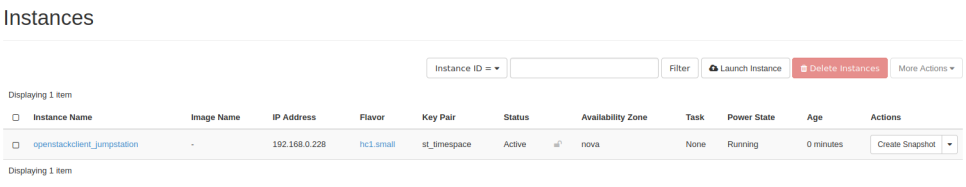


Since **ssh_key** is the only SSH public key associated with this cloud it is the default option and selected for you already. If there were other keys you could add them from the list below. You can create a key pair or upload a public key using this form.

With these steps done, you are ready to create the instance. Click **Launch Instance** to do so.

When the instance has finished being setup and active it will appear this way:

Instance listing:



Troubleshooting Instance Error Status

Sometimes, instance creation will not succeed which can be for a variety of reasons. Running `openstack server list` will show the **Status** of an instance.

Here's an example of an instance with Status, "ERROR":

```
$ openstack server list
+--------------------------------------+-----------------------------+---------+------------
| ID                                   | Name                        | Status  | Networks
+--------------------------------------+-----------------------------+---------+------------
| 35d624fe-785d-4915-aa7e-4bb580b29325 | centos_instance_2           | ERROR   |
--------------------------------------+-----------------------------+---------+------------
```

Get more information on the error status by running:

```
$ openstack server show $INSTANCE_UUID --fit-width
```

Look for the **fault** row for the reason as to why the instance failed to create.

It is suggested to look over https://docs.openstack.org/nova/train/admin/support-compute.html for common issues that may arise and how to troubleshoot them.

Location of OpenStack service logs

You may need to look at service's logs to get a better idea of an issue. The default log location for an OpenStack deployment is `/var/log/$service_name`, however OpenStack in this case has been deployed using `kolla-ansible` and logs are stored in `/var/log/kolla/$service_name` on each hardware node.

Next Steps

The next guide in this series will explain how to SSH into this instance.

If you wish to skip the SSH guide, see the managing images in OpenStack guide.

Create an OpenStack Instance with OpenStackClient

The purpose of this guide is to explain how to create an instance using the command line through OpenStackClient.

An example will be demonstrated where an instance is created on a private network. The intent of the instance will be to act as a "jumpstation" where OpenStackClient will be installed. Finally, this jumpstation can be used to administer the OpenStack cloud using OpenStackClient.

An instance is another name for a virtual machine in OpenStack. Instances are created by the Nova service and contribute to the processing power of the cloud.

You'll also learn how to upload or create an SSH key pair, assign storage using a volume, and create a security group. Each of these components will be added to the instance.

NOTE! – You will need to have OpenStackClient installed already to follow this guide.

OpenStackClient

OpenStackClient is how you would manage an OpenStack cloud using the command line.

For general information on OpenStackClient and how to install it, see the Day 1 guide.

Throughout these guides OpenStackClient will be also be known as OSC.

To get a full list of the available commands, run:

```
openstack --help
```

NOTE! – In addition to OpenStackClient, there are other command line clients for various OpenStack services that can be used. For example, Nova and Cinder each have their own command line client, as well as other services.

In later releases of OpenStack use of service-specific command line interfaces will be deprecated. When using the command line to administer OpenStack it is recommended to use OpenStackClient where possible as opposed to individual service's command line interfaces such as `nova`'s CLI.

Prerequisites for Creating an Instance

Before making an instance, ensure these items exist:

- A flavor
- An image
- A network
- A security group
- An SSH public key

At minimum, a flavor, an image, and a network should exist. You may also want to attach a volume, specify security groups to use, and attach an SSH public key.

SSH key pairs

An SSH key pair will be required to access any instances over SSH. Password authentication is by default disabled in the operating system images.

You can either have an SSH key created or you can upload the public key of your SSH key pair.

Create an SSH key pair

SSH keys can be managed, created, and uploaded through the command line using OpenStackClient.

To make a key pair, use:

```
$ openstack keypair create KEY_NAME
```

KEY_NAME is the name of the SSH key pair.

This will generate a key pair for you and return the private key. The private key should be kept somewhere safe and be inaccessible to others.

To upload your public key, use:

```
$ openstack keypair create --public-key PATH_TO_PUBLIC_KEY KEY_NAME
```

KEY_NAME is the name of the SSH key pair and **PATH_TO_PUBLIC_KEY** is the path on the filesystem to the public key.

Here's an example of uploading an SSH public key:

```
$ openstack keypair create --public-key ~/.ssh/ssh_key.pub ssh_key_2
+-------------+-------------------------------------------------+
| Field       | Value                                           |
+-------------+-------------------------------------------------+
| fingerprint | ff:a4:81:c7:59:07:aa:54:43:39:52:cd:b2:12:aa:fb |
| name        | ssh_key_2                                       |
| user_id     | 43317575cccc440fbcb38a1f23b45125                |
+-------------+-------------------------------------------------+
```

Security groups

A security group in OpenStack controls network access. If no security group is assigned, a default one will be used. The default security group disables all inbound traffic and only allows outbound traffic, which may not be useful.

A security group should be made that allows the type of incoming and outgoing traffic required. OpenStack's default security group does not allow incoming traffic so if you wanted to SSH into an instance, port 22 will be closed. An example of a security group would be one that allows web traffic by opening ports 80 and 443 and you may want another security group that opens port 22 for SSH traffic.

Create and manage security groups

This section will demonstrate creating a security group that allows SSH traffic from a specific host.

Basics

> The command to create a security group appears like so, where

SECURITY_GROUP is the name of the security group:

```
$ openstack security group create SECURITY_GROUP
```

> To list security groups, run:

```
$ openstack security group list
+--------------------------------------+----------------------+-----------------------------
| ID                                   | Name                 | Description
+--------------------------------------+----------------------+-----------------------------
| 8639e3c5-47ce-4072-a1f5-1c1e931a8f75 | default              | Default security group
| ebffcf78-52d9-436c-81db-5ea788a0c33d | devstack             |
| ec8a02ba-4bc2-4b78-a555-902caead87fe | basic_webserver_group | This will open standard por
+--------------------------------------+----------------------+-----------------------------
```

> To list the details of a specify security group, where UUID is the security

group's UUID, run:

```
$ openstack security group show UUID
```

Procedure

The following example demonstrates opening port 22 for incoming traffic coming from **50.50.50.50**.

There are two steps to the example: The first step shows creating a security group called **ssh_demo** and the next step shows adding the SSH rule to that group.

Open port 22 for incoming SSH traffic:

```
# Create the security group
$ openstack security group create ssh_demo

# Add rule for port 22
```

```
$ openstack security group rule create --remote-ip 50.50.50.50/32 \
--dst-port 22:22 --ingress --protocol tcp ssh_demo
```

The next section is not required, but shows how to make a security group allowing incoming traffic to the common HTTP ports, 80 and 443.

Example security group for HTTP incoming traffic:

```
$ openstack security group rule create --remote-ip 0.0.0.0/0 \
--dst-port 80:80 --ingress --protocol tcp SECURITY_GROUP

$ openstack security group rule create --remote-ip 0.0.0.0/0 \
--dst-port 443:443 --ingress --protocol tcp SECURITY_GROUP
```

Assign Storage

Storage can be assigned through volumes. A volume in OpenStack is like a removable USB drive that can be attached to instances as seen fit. Cinder is the OpenStack service that allows volumes to be created. A volume can also be used to boot an instance.

Create and attach a volume

Use the following command to create a volume with size 15GB:

```
$ openstack volume create volume_1 --size 15
```

List the newly created volume:

```
$ openstack volume list
+--------------------------------------+--------------+-----------+------+------------------
| ID                                   | Name         | Status    | Size | Attached to
+--------------------------------------+--------------+-----------+------+------------------
| 1fe4f750-62fc-4c53-b661-44481d0f72b3 | volume_1     | available |   15 |
+--------------------------------------+--------------+-----------+------+------------------
```

Attach the volume to an instance:

```
$ openstack server add volume $INSTANCE_UUID $VOLUME_UUID
```

Create an instance

If at this step and you have followed the previous steps, you should have everything needed to create the instance. Again, this example will demonstrate how to create an instance on a private network.

NOTE! – If your infrastructure does not need to be on a public network, then ensure the appropriate parts are created on a private network, and be sure the security groups are set to only allow the needed traffic through. This information goes a long way in improving overall security when using OpenStack.

Before creating an instance

Before creating an instance, you will need to collect some information.

Needed details:

- The flavor, image, and network UUIDs
- SSH key
- Security group

You can list the above using OpenStackClient. The next section will provide examples using each command and their outputs.

NOTE! – When listing items using OpenStackClient, almost everything will have a UUID. However, if a UUID does not exist, you'll need to use the name specified in the `ID` column or the name of the item itself.

Here are commands that can be used to collect the needed information:

Summary of commands:

```
$ openstack flavor list
$ openstack image list
$ openstack network list
$ openstack security group list
$ openstack keypair list
```

Detailed commands:

List flavors:

```
$ openstack flavor list
+--------------+--------------+-------+------+-----------+-------+-----------+
| ID           | Name         |   RAM | Disk | Ephemeral | VCPUs | Is Public |
+--------------+--------------+-------+------+-----------+-------+-----------+
| hc1.large    | hc1.large    |  8192 |   80 |         0 |     8 | True      |
| hc1.medium   | hc1.medium   |  4096 |   40 |         0 |     4 | True      |
| hc1.micro    | hc1.micro    |  1024 |   10 |         0 |     1 | True      |
| hc1.small    | hc1.small    |  2048 |   20 |         0 |     2 | True      |
| hc1.xlarge   | hc1.xlarge   | 16768 |  160 |         0 |    16 | True      |
| hram1.medium | hram1.medium | 16768 |   40 |         0 |     4 | True      |
+--------------+--------------+-------+------+-----------+-------+-----------+
```

Flavors are a way to set things like the amount of RAM, disk space, and CPU cores to be assigned to an instance. The `Is Public` heading says that this flavor is shared among all OpenStack projects.

List images:

```
$ openstack image list
+--------------------------------------+-----------------------------+--------+
| ID                                   | Name                        | Status |
+--------------------------------------+-----------------------------+--------+
| 22c437b3-18cd-4af0-bd3d-ad26c85bb00f | CentOS 7 (ce7-x86_64)       | active |
| cd18f302-d3bf-49a1-8f5d-a6805404b9ff | CentOS 8 (ce8-x86_64)       | active |
| 507ca3c5-ed57-4997-b21d-bc0f32406322 | Debian 10 (buster-amd64)    | active |
| c19c4d9d-59c9-49e4-8e5f-988d3c1ae4d4 | Debian 9 (stretch-amd65)    | active |
| 46b14073-bd4e-4aa2-b901-63c86ab15961 | Ubuntu 18.04 (bionic-amd64) | active |
| 40b09133-a1a0-4ea1-a8aa-958541314ac5 | Ubuntu 20.04 (focal-amd64)  | active |
+--------------------------------------+-----------------------------+--------+
```

Images are typically an operating system image and usually can be found publicly available. CentOS has a repository of cloud instances for example.

Images can also created using a snapshot of an instance.

List networks:

```
$ openstack network list
+--------------------------------------+-------------------------------------------------------
| ID                                   | Name
+--------------------------------------+-------------------------------------------------------
```

```
| 0db14de2-9dd0-469f-a76c-46b99119d607 | External
| bdb2dbfc-e3ef-4da0-a79b-fe48c4a413fb | Internal
| fcd8166c-0fdb-4bc9-ab29-ce54a8bd5959 | HA network tenant b9e8639372014c0b85cbfaffa6e1b5a8
+--------------------------------------+-----------------------------------------------
```

List security groups:

```
$ openstack security group list
+--------------------------------------+-----------+-----------------------------+----------
| ID                                   | Name      | Description                 | Project
+--------------------------------------+-----------+-----------------------------+----------
| 06785fc8-492c-477b-9928-9708cb550a14 | ssh_demo  | ssh_demo                    | b9e8639372
| 9e20fbef-9eca-4029-8111-625945409ae2 | default   | Default security group      | 6a654535b8
| cc25e2d6-6a5c-469f-ba89-c2e04e6f6850 | SSH       | General SSH security group | b9e8639372
+--------------------------------------+-----------+-----------------------------+----------
```

List SSH key pairs:

```
$ openstack keypair list
+--------------+-------------------------------------------------+
| Name         | Fingerprint                                     |
+--------------+-------------------------------------------------+
| st_timespace | a7:ab:bd:9c:78:85:e1:a1:c4:07:0f:6d:e9:36:0b:68 |
+--------------+-------------------------------------------------+
```

Procedure to create an instance

Base command needed to make an instance:

```
$ openstack server create
```

From this command you will specify additional flags such as the SSH key, the network UUID, the instance name, and the like.

For the full list of options to make an instance, run:

```
$ openstack help server create
```

The command to create an instance will look something like this:

```
$ openstack server create --image IMAGE_UUID \
--flavor hc1.medium --network NETWORK_UUID \
--key-name SSH_KEY_NAME --security-group SECURITY_GROUP_UUID \
INSTANCE_NAME
```

Note that IMAGE_UUID is the UUID of the image you want to use, NETWORK_UUID is the UUID of the network to be associated with the instance, SSH_KEY_NAME is the name of the SSH key, SECURITY_GROUP_UUID is the UUID of the security group to use, and INSTANCE_NAME is the name to call the instance.

NOTE – By default, the instance creation will occur in the background. You can add `--wait` to the flags to have the command wait until the instance creation is done which will show you the status of instance creation.

Example instance creation:

Here are the collected details to make an instance from the previous section:

- `Flavor`: hc1.small
- `Image UUID`: 40b09133-a1a0-4ea1-a8aa-958541314ac5
- `Network UUID`: bdb2dbfc-e3ef-4da0-a79b-fe48c4a413fb
- `SSH key`: st_timespace
- `Security group UUID`: cc25e2d6-6a5c-469f-ba89-c2e04e6f6850

Full command:

```
$ openstack server create --image 40b09133-a1a0-4ea1-a8aa-958541314ac5 \
--flavor hc1.small --network bdb2dbfc-e3ef-4da0-a79b-fe48c4a413fb \
--key-name st_timespace --security-group \
cc25e2d6-6a5c-469f-ba89-c2e04e6f6850 openstackclient_js_demo
```

After creating the instance, verify the build process by running `$ openstack server show INSTANCE_NAME`.

The `status` column will indicate the status. See the Nova compute API documentation for a list of instance status meanings and additional commands that can be used to troubleshoot any issues.

You also may find the table output is too large for your display. Add `--fit-width` to the command to force the table to fit to your display's width making reading easier.

Troubleshooting

NOTE! – This section is incomplete but will receive updates in the future.

There may be an issue with spawning the instance. Using the above command will display any errors with the build process under the `fault` field.

Troubleshooting Instance Error Status

Sometimes, instance creation will not succeed which can be for a variety of reasons. Running `openstack server list` will show the **Status** of an instance.

Here's an example of an instance with Status, "ERROR":

```
$ openstack server list
+--------------------------------------+------------------------------+---------+------------
| ID                                   | Name                         | Status  | Networks
+--------------------------------------+------------------------------+---------+------------
| 35d624fe-785d-4915-aa7e-4bb580b29325 | centos_instance_2            | ERROR   |
--------------------------------------+------------------------------+---------+------------
```

Get more information on the error status by running:

```
$ openstack server show $INSTANCE_UUID --fit-width
```

Look for the **fault** row for the reason as to why the instance failed to create.

It is suggested to look over https://docs.openstack.org/nova/train/admin/support-compute.html for common issues that may arise and how to troubleshoot them.

Location of OpenStack service logs

You may need to look at service's logs to get a better idea of an issue. The default log location for an OpenStack deployment is `/var/log/$service_name`, however OpenStack in this case has been deployed using `kolla-ansible` and logs are stored in `/var/log/kolla/$service_name` on each hardware node.

The next guide in this series will explain how to connect an instance to the provider network, allowing Internet access.

Next Steps

The next guide in this series will explain how to SSH into this instance.

If you wish to skip the SSH guide, see the managing images in OpenStack guide.

SSH into an Instance

This guide serves to explain how you can SSH into an instance.

The OpenStackClient can be used to SSH into instances. Instances can be created on both public and private networks. The location from where the SSH command is being issued has to be able to access the instance over the network. Typically instances created either on the provider network or that have been assigned a floating IP can be accessed from any machine that is connected to the Internet.

Access Instance associated with a Public IP

If the instance has a floating IP or is on the provider network, then the instance can be accessed by any machine that has OpenSSH installed or from any machine that has OpenStackClient installed.

Using SSH

To SSH into an instance, the machine you connect from has to be able to connect to the SSH port (typically 22) of the instance. This means the security group associated with the instance must allow SSH traffic from the machine you intend to SSH from. See the create an instance guide for how to create security groups.

An example command to SSH into an instance:

```
$ ssh -i ~/.ssh/KEY USER@50.50.50.50
```

In this example, `~/.ssh/KEY` is your private key, **USER** is the SSH username, and **50.50.50.50** is an IP you can connect to from the machine you intend to SSH from.

Using OpenStackClient

It is also possible to use OpenStackClient to SSH into a machine. To be able to use OpenStackClient for this purpose, the machine you connect from must have SSH access to the instance. Again, ensure the instance has a security group that allows SSH access from the host you intend to connect with.

Base command to SSH into an instance:

```
$ openstack server ssh
```

Example command to SSH into an instance which has been created on the provider network:

```
$ openstack server ssh --login centos --identity ~/.ssh/ssh_key --address-type fixed cf491bc
```

`--address-type` can be public, private, or fixed

A **fixed** `--address-type` means the IP assigned to the instance is a static IP. When an instance is created on the public network, a fixed IP will be assigned to it.

Access Instance associated with a Private IP

It is also possible to SSH into an instance that is on a private network. This will have to be done from one of the hardware nodes which has to be associated with the appropriate private network. The private key of the SSH key pair should be on that node as well.

Listed is the instance in question to connect to:

```
$ openstack server list
+--------------------------------------+------------------------------+---------+------------
| ID                                   | Name                         | Status  | Networks
+--------------------------------------+------------------------------+---------+------------
```

```
| e93b3344-6d78-4273-880f-220b7fbec417 | test_5                       | ACTIVE  | Internal=19
+--------------------------------------+------------------------------+---------+------------
```

It can be seen the IP associated with it is on a private network.

Determine what compute node the instance is on:

```
$ openstack server show e93b3344-6d78-4273-880f-220b7fbec417
+-------------------------------------+-----------------------------------------------------
| Field                               | Value
+-------------------------------------+-----------------------------------------------------
| OS-DCF:diskConfig                   | AUTO
| OS-EXT-AZ:availability_zone         | nova
| OS-EXT-SRV-ATTR:host                | hc1.example.com
| OS-EXT-SRV-ATTR:hypervisor_hostname | hc1.example.com
```

Note that some of the output is truncated.

Next, SSH into the appropriate compute node and then find the appropriate private network. This can be done by listing the network namespaces on that node.

List network namespaces:

```
# ip netns
qrouter-4dc1debc-ecf3-42e1-89c6-e2b99fc2c3dd (id: 0)
qdhcp-a54fc8a3-89b1-4ec3-a441-79c6cfe0e915 (id: 3)
qdhcp-55d31bd5-77ba-4ed0-ab6e-99554b33aa90 (id: 1)
```

List the interfaces for a network namespace using the format `ip netns exec $network_namespace $command`, so for example:

```
# ip netns exec qdhcp-a54fc8a3-89b1-4ec3-a441-79c6cfe0e915 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
164: tapaa57977f-ca: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether fa:16:3e:a3:db:34 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.2/24 brd 192.168.0.255 scope global tapaa57977f-ca
       valid_lft forever preferred_lft forever
    inet 169.254.169.254/16 brd 169.254.255.255 scope global tapaa57977f-ca
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fea3:db34/64 scope link
       valid_lft forever preferred_lft forever
```

From here the output shows the subnet `192.168.0.2/24` listed and `192.168.0.186` is on that network, so to SSH into this instance, an example command would take a form like this:

```
# ip netns exec qdhcp-a54fc8a3-89b1-4ec3-a441-79c6cfe0e915 ssh -i $ssh_key centos@192.168.0.
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Mon Aug 24 21:59:47 2020 from 192.168.0.2
[centos@test-5 ~]$
```

Next Steps

The next guide explains how to manage images in OpenStack.

OpenStack Images

Images in OpenStack are what powers instances. This guide will explain how to use images that are already uploaded into Glance, the OpenStack service responsible for images and how to upload your own.

An image is a file that contains a bootable operating system. Many different cloud image sources are available for download from major operating system providers like CentOS, Ubuntu, and Debian to name a few. You can also make your own images from scratch or create them from volumes or running instances. Snapshots of instances can be created which can serve both as a backup and also a template for other instances.

It is possible to create your own images however this process is outside the scope of this guide. For more information on how to do this, see the documenation from OpenStack.

List and upload images

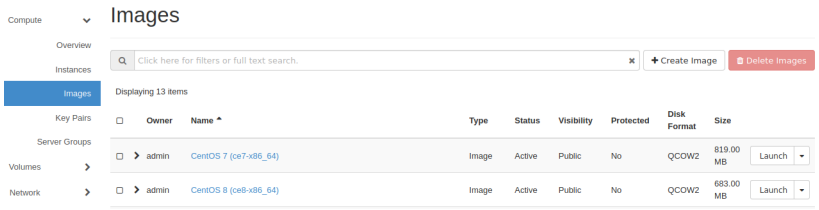
This section will cover how to view and upload images into OpenStack. Either Horizon or the command line using OpenStackClient can be used.

Using Horizon

In Horizon, the images can be found on the left within the **Compute** tab under **Images**. From this interface, you could launch an instance using one of the images and also upload your own.

List images

Images in Horizon:



Upload and create images

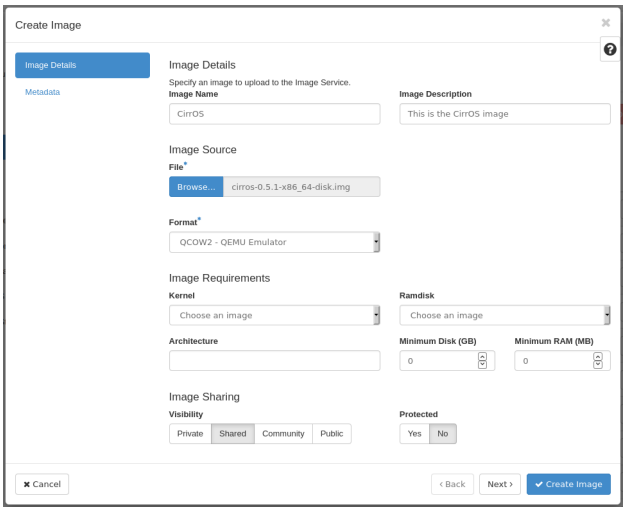
In the same Horizon interface, you can upload an image, make your own or make an image snapshot from a pre-existing instance or volume.

Many operating system providers have cloud images available for download already. For example, see the image repository from CentOS.

Similarly, other major Linux operating system creators provide images that can be used in OpenStack.

To upload an image, login to Horizon, navigate to **Compute**, then **Images**. From there, click the **Create Image** button. A form will appear where you can fill out the needed details. Click the **?** icon for assistance with specifics.

Upload an image:



This example will walk through uploading the CirrOS image.

Ensure the image to upload has been downloaded to your machine. CirrOS has been downloaded from the CirrOS download page. The latest version is `0.5.1`. Be sure to download the latest version of the image needed.

With the image on your machine, the form details can be filled.

- **Image Details**
    - **Image Name** - specify the name for the uploaded image
    - **Image Description** - option field used to describe the image
- **Image Source**
    - **File** - this is the location of the image file on your machine
    - **Format** - the format should be **QCOW2 - QEMU Emulator**

Optionally, you can specify requirements, like minimum disk space and RAM, for the image under the **Image Requirements** heading.

Using the command line

You can also view and upload images using OpenStackClient over the command line.

List images

To view available images, use:

```
$ openstack image list
+--------------------------------------+---------------------------------+--------+
| ID                                   | Name                            | Status |
+--------------------------------------+---------------------------------+--------+
| 6986aaf9-e602-4307-9c4f-6377795ff890 | CentOS 7 (ce7-x86_64)           | active |
| d3f027f2-730f-4794-9288-44ed5c69050d | CentOS 8 (ce8-x86_64)           | active |
| 29c5d143-9fda-4f8e-b364-c2dc29851101 | Debian 10                       | active |
| 6eabc356-c373-4260-8f6f-7d83ff80df21 | Snap-1                          | active |
| 994b6e02-b8c4-4652-80bf-3d56c92b4b19 | Ubuntu 18.04 (bionic-amd64)     | active |
| 20b56b2d-1af5-46e1-a5c6-fa1f9a45245d | Ubuntu 19.10 (eoan-amd64)       | active |
| d35559ff-c9fd-4c18-be8b-1a74eca1eb38 | Ubuntu 20.04 (focal-amd64)      | active |
+--------------------------------------+---------------------------------+--------+
```

To get more information about an image, run:

```
$ openstack image show ID
```

where **ID** can be the Name or the ID column in the above output.

Example showing the details of an image:

```
$ openstack image show fa8eb9bd-9ccc-4d3f-b87b-6edb5450a57a --fit-width
+------------------+------------------------------------------------------------------------------+
| Field            | Value
+------------------+------------------------------------------------------------------------------+
| checksum         | 1d3062cd89af34e419f7100277f38b2b
| container_format | bare
| created_at       | 2020-09-09T20:50:25Z
| disk_format      | qcow2
| file             | /v2/images/fa8eb9bd-9ccc-4d3f-b87b-6edb5450a57a/file
| id               | fa8eb9bd-9ccc-4d3f-b87b-6edb5450a57a
| min_disk         | 0
| min_ram          | 0
| name             | cirros
| owner            | 5ad1f9e795604f4390d274d7388c4b9f
```

The output of the above command has been truncated.

Upload an image

To upload an image, run:

```
openstack image create
```

For assistance with the CLI options, run:

```
openstack help image create
```

OpenStack provides an operating system called CirrOS that has the minimum requirements to be an operating system that is generally used to test instance creation.

To upload this CirrOS image into the glance service, you'll need to first download it to where the OpenStackClient lives, then the base command `openstack image create` can be used to upload the image.

Steps to upload the CirrOS image:

Obtain the latest CirrOS image:

```
$ wget http://download.cirros-cloud.net/0.5.1/cirros-0.5.1-x86_64-disk.img
```

Upload the image into glance:

```
$ openstack image create cirros --container-format bare --disk-format \
qcow2 --file PATH_TO_CIRROS_IMAGE
```

Note that **PATH_TO_CIRROS_IMAGE** should be path to the image file.

List the newly uploaded image:

```
$ openstack image list
+--------------------------------------+---------------------------------+--------+
| ID                                   | Name                            | Status |
+--------------------------------------+---------------------------------+--------+
| fa8eb9bd-9ccc-4d3f-b87b-6edb5450a57a | cirros                          | active |
+--------------------------------------+---------------------------------+--------+
```

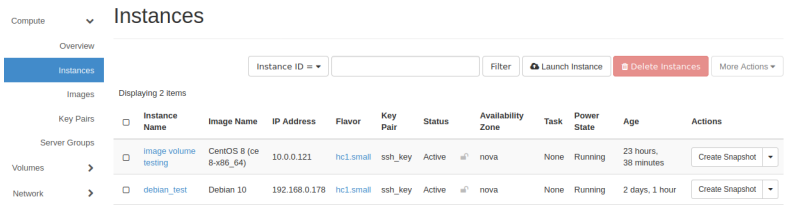
Create Image Snapshot

Snapshot using Horizon

An image can also be created from an existing instance by making use of the image snapshot feature.

In the dashboard, look for **Compute**, then **Instances**. From there, select an instance, and click **Create Snapshot** from the drop down on the right.

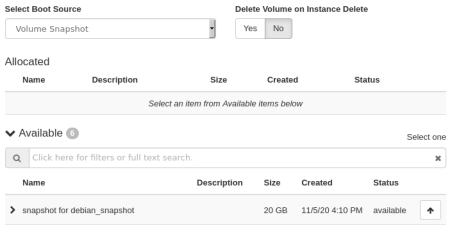
Create image snapshot:



This will create an image of that instance and the state of the disk will be preserved. You can use this to launch other instances. Should you want to launch an instance using this snapshot, it will be listed in the **Launch Instance** form under the **Select Boot Source** dropdown as a volume snapshot.

Create snapshot form:

Create Instance form showing the volume snapshot recently created:



Snapshot Using the command line

A snapshot of an instance can be made using the command line with OpenStackClient.

The command to create a snapshot is:

```
$ openstack server image create --name SNAPSHOT_NAME INSTANCE_NAME
```

- **SNAPSHOT_NAME** is the name to call the snapshot
- **INSTANCE_NAME** is the name of the instance the snapshot is created from. The instance UUID could also be used.

After creating a snapshot verify success by:

- `openstack image list`
- `openstack image show SNAP_SHOT_UUID` - SNAP_SHOT_UUID refers to the UUID of the snapshot made.

Create and verify an instance snapshot

Make a snapshot of the **debian_test** instance:

```
$ openstack server image create --name debian_snapshot debian_test --fit-width
+------------------+-------------------------------------------------------------------------------
```

```
| Field            | Value
+------------------+-------------------------------------------------------------------------
| checksum         | d41d8cd98f00b204e9800998ecf8427e
| container_format | bare
| created_at       | 2020-11-05T16:21:25Z
| disk_format      | qcow2
| file             | /v2/images/161eb918-f470-4bb8-9572-43ee9eb27ceb/file
| id               | 161eb918-f470-4bb8-9572-43ee9eb27ceb
| min_disk         | 20
| min_ram          | 0
| name             | debian_snapshot
| owner            | 5ad1f9e795604f4390d274d7388c4b9f
```

List available snapshots:

```
$ openstack image list --fit-width
+--------------------------------------+---------------------------------+--------+
| ID                                   | Name                            | Status |
+--------------------------------------+---------------------------------+--------+
| 6986aaf9-e602-4307-9c4f-6377795ff890 | CentOS 7 (ce7-x86_64)           | active |
| d3f027f2-730f-4794-9288-44ed5c69050d | CentOS 8 (ce8-x86_64)           | active |
| 29c5d143-9fda-4f8e-b364-c2dc29851101 | Debian 10                       | active |
| 161eb918-f470-4bb8-9572-43ee9eb27ceb | debian_snapshot                 | active |
+--------------------------------------+---------------------------------+--------+
```

The snapshot created is listed as the last entry under ID of **161eb918-f470-4bb8-9572-43ee9eb27ceb**.

List details of the snapshot:

```
$ openstack image show 161eb918-f470-4bb8-9572-43ee9eb27ceb --fit-width
+------------------+-------------------------------------------------------------------------
| Field            | Value
+------------------+-------------------------------------------------------------------------
| checksum         | d41d8cd98f00b204e9800998ecf8427e
| container_format | bare
| created_at       | 2020-11-05T16:21:25Z
| disk_format      | qcow2
| file             | /v2/images/161eb918-f470-4bb8-9572-43ee9eb27ceb/file
| id               | 161eb918-f470-4bb8-9572-43ee9eb27ceb
| min_disk         | 20
| min_ram          | 0
| name             | debian_snapshot
| owner            | 5ad1f9e795604f4390d274d7388c4b9f
```

Ceph, Cinder and Swift

Ceph, Cinder, and Swift are used in OpenStack for data storage. Ceph is the backend storage software used and is not an OpenStack service, but rather the backend system where data for Cinder and Swift is stored. Cinder is OpenStack's service for block storage and Swift is the service responsible for object storage.

The purpose of this guide is to briefly explain each of these services and how they relate to one another.

Ceph

Ceph is a unified, distributed storage system designed for excellent performance, reliability and scalability and is the backend storage used by this deployment of OpenStack.

The Octopus release of Ceph is currently being used.

Ceph Data Durability

Ceph has different strategies for data durability, or data redundancy. See Ceph's documentation for information on how Ceph stores data.

Typically a Ceph configuration will have three replicas of data per pool as this is the best tradeoff between performance and data durability. If data durability and cost are not as important, Ceph can be configured with two OSDs per pool, with the tradeoff being less physical hardware used to store data, meaning less cost. Another option in Ceph for replicating data is called erasure coding. This method is a software approach to replicating data that requires less disk space than using three replica OSDs at the expense of greater CPU and RAM usage.

Cinder

Cinder is a block storage service for OpenStack which allows volumes to be created. A volume is a detachable storage device and can only be attached to one instance at a time.

This service virtualizes the management of block storage devices. End users make use of the storage service without needing to know where the storage is deployed or on what type of device it is stored. Drivers can be created which allow for multiple storage backend solutions, including local via LVM (Logical Volume Manager), Ceph, and others.

More Information

For more information about Cinder, check out these resources.

- Cinder OpenStack wiki
- Cinder's OpenStack documentation

What is block storage?

Block storage is strategy of storing data where the data is stored in fixed size chunks called blocks. The address of where the block is located is the only metadata associated with each block.

Swift

Swift is the OpenStack service that handles object storage. This service allows containers to be created that can store objects, for example a collection of images, audio files, or a combination of any sort of files. Files can be uploaded through the Swift API or using Horizon.

Swift uses Ceph's RadosGW as a means to handle object storage, effectively meaning Ceph is used to store Swift data.

More Information

For more information about Swift, check out these resources.

- Swift OpenStack wiki
- Swift's OpenStack documentation

What is object storage?

Object storage is a strategy of storing data in various formats of varying sizes. Objects are stored as entire objects, so an mp3 file is stored completely as that along with any metadata associated with that object. The problem object storage solves is data scalability.

How do Cinder and Swift relate to Ceph?

Both Cinder and Swift can use different storage backends, ranging from local disks or a storage cluster like Ceph.

This deployment of Cinder and Swift both use Ceph as a backend for data storage.

Swift has been configured to use Ceph's Object Gateway also known as **RADOS Gateway**.

Using the Ceph CLI for Common Operational Tasks

This guide will go over some of the common tasks to do using the Ceph's CLI tool.

Common tasks

Get status of ceph cluster:

```
$ ceph status
```

List ceph pools:

```
$ ceph osd lspools
```

List overall disk usage:

```
$ ceph df
```

Get status of all OSDs:

```
$ ceph osd status
```

Get status of monitors:

```
$ ceph mon stat
```

Get status of placement groups (PGs):

```
$ ceph pg stat
```

List amount of disk usage per OSD:

```
$ ceph osd df
```

Change number of replicated OSDs for a pool:

```
$ ceph osd pool set $POOL size $COUNT
```

More detailed ``ceph status`` output

Get Ceph cluster status using `ceph status`:

```
# ceph status
cluster:
  id:     228c6e47-b60e-4710-9bcb-54316f752857
  health: HEALTH_WARN
          1 pools have many more objects per pg than average
          clock skew detected on mon.hc2, mon.hc3

services:
  mon: 3 daemons, quorum hc1,hc2,hc3 (age 21h)
  mgr: hc2(active, since 8d), standbys: hc1, hc3
```

```
  osd: 3 osds: 3 up (since 21h), 3 in (since 22h)
  rgw: 3 daemons active (hc1.rgw0, hc2.rgw0, hc3.rgw0)

task status:

data:
  pools:   13 pools, 225 pgs
  objects: 15.65k objects, 59 GiB
  usage:   169 GiB used, 2.5 TiB / 2.6 TiB avail
  pgs:     225 active+clean

io:
  client:   29 KiB/s rd, 34 op/s rd, 0 op/s wr
```

Ceph Replication, Compression, and Erasure Coding

Ceph is naturally resilient to data loss. It accomplishes this by replicating the same set of data several times. In other words, Ceph stores multiple copies of the same set of data. Data is either replicated across several Object Storage Daemons (OSDs) or is replicated using erasure coding.

This guide will explain a bit how Ceph has been configured for Private Cloud Core and also cover adjusting the number of replicated OSDs per pool.

Ceph Replication

With Private Cloud Core, Ceph has been distributed across each hard drive on each hardware node and the data in each pool is being replicated across three OSDs.

The configuration is put into place using `ceph-ansible` and the repository for this software is located in Github. Use of `ceph-ansible` assumes a base understanding of how Ansible works.

Ceph has a concept of pools and there are typically several pools that store different sets of data. Each pool can be configured with different replication options, such as erasure coding for one or some pools and other pools can be set to have multiple OSDs. More on the options for configuring pools can be found in Ceph's documentation.

Another way to replicate data in Ceph is to use erasure coding. This is a software approach to data replication that uses less disk space, but has more overhead in terms of CPU and RAM usage. See the InMotion Hosting support center for more information on this subject.

With erasure coding, this replication type must be set upon creation of the Ceph pool and cannot be changed after a pool has been created. If you have a set of data replicated using erasure coding and want to change how data is replicated, a pool with the replication strategy intended needs to be created then the data would be migrated to that pool.

Here's an example of the pools you may see in the Ceph cluster using the `ceph osd lspools` ceph CLI command:

```
$ ceph osd lspools
1 device_health_metrics
2 images
3 volumes
4 vms
5 backups
6 metrics
7 manila_data
8 manila_metadata
9 .rgw.root
10 default.rgw.log
11 default.rgw.control
12 default.rgw.meta
13 default.rgw.buckets.index
```

See the documentation from Ceph for details on how Ceph handles data durability, otherwise known as data replication.

Setup replication across 2 OSDs

Let's say you wanted the `volumes` pool to have replication between two OSDs. The command to set that up would look like this:

```
$ ceph osd pool set volumes size 2
set pool 3 size to 2
```

Setup replication across 3 OSDs

The recommended number of replicated OSDs is three and to set that for the `volumes` pool, this command needs to be used:

```
$ ceph osd pool set volumes size 3
set pool 3 size to 3
```

Erasure Coding

For information on erasure coding in general, see the InMotion Hosting support center.

OpenStack CLI for Common Operational Tasks

Background

OpenStackClient (OSC) is the name of a command line interface for OpenStack which can be used to administer an OpenStack cloud. The same functionality found in the Horizon interface can also be found using OpenStackClient.

Get started

OpenStackClient will need to be installed before it can be used. See this guide for installation instructions.

For a full list and explanation of the available options use `openstack help` or refer to the OpenStack documentation.

Additional command line applications exist for some services such as Nova and Cinder but will eventually be deprecated. While these command line utilities can be used, it is recommended that the OpenStack cloud be administered using soley the OSC. The goal of OpenStack client is to provide all the needed commands to administer an OpenStack cloud under one application.

Output formatting

Typically when OpenStackClient commands are issued, a table of formatted data is returned. When doing batch operations you may want to extract just the **UUID** of an item. It is possible to have OpenStackClient return exactly the information you need which can be useful for scripts or running the same action on multiple items.

For example:

```
output formatters:
  output formatter options

  -f {csv,json,table,value,yaml}, --format {csv,json,table,value,yaml}
                        the output format, defaults to table
  -c COLUMN, --column COLUMN
```

```
                        specify the column(s) to include, can be repeated to
                        show multiple columns
  --sort-column SORT_COLUMN
                        specify the column(s) to sort the data (columns
                        specified first have a priority, non-existing columns
                        are ignored), can be repeated
```

Say you want just the **UUID** of all the servers on a host. You can run something like this to achieve that goal (where `example_host` is the host you are working with):

```
$ openstack server list --host example_host -f value -c ID
```

Common Tasks

Below is a list of common operational tasks that can be done with the OpenStack Client. This guide will include examples of how to perform the following tasks:

- Manage OpenStack users, including listing, creating, updating and removing users
- Manage instances by creating them, stopping and starting them, creating a snapshot, plus much more.
- Live migration of instances
- Troubleshoot instance issues
- Upload images
- Create a network
- Create security groups
- Manage SSH key pairs
- Collect details about OpenStack environment

Manage OpenStack users

In OpenStack there exists the admin user account which has the ability to create additional users. Typically the admin account is used only when that level of privilege is needed otherwise individual user accounts should be used when interacting with an OpenStack cloud.

The following commands can be used to list, create, update, and remove OpenStack users:

List users:

```
$ openstack user list
```

Create a user where **PROJECT_NAME** is the name of the project, **PASSWORD** is the password to set, and **USERNAME** is the username:

```
$ openstack user create --project PROJECT_NAME --password PASSWORD \
USERNAME
```

The base command to update a user is `openstack user set` USERNAME where USERNAME is the username in question.

Using that base command, you can enable or disable a user account or change details, such as the email address associated with the user.

Disable a user:

```
$ openstack user set USERNAME --disable
```

Enable a user:

```
$ openstack user set USERNAME --enable
```

Change the email address, where **EMAIL_ADDRESS** should be the email to set:

```
$ openstack user set USERNAME --email EMAIL_ADDRESS
```

Instance Management

Create an instance

See the Day 1 guide for information on how to create an instance.

Stop and start an instance

Stop an instance:

```
$ openstack server stop
```

Start an instance:

```
$ openstack server start
```

Create an instance snapshot

Here's an example:

```
$ openstack server image create --name SNAPSHOT_NAME INSTANCE_NAME
```

You can verify the snapshot has been created by using both `openstack image list` to find the newly created snapshot, then `openstack image show SNAP_SHOT_UUID` to get details on that snapshot.

Perform live migration of instances

Sometimes it is necessary to migrate instances from one compute node to another if for example a compute node needs maintenance. It is possible to live migrate instances before bringing down that node.

Before you begin, you'll need the server UUID, the host that server is currently running on and the host to migrate to.

List servers:

```
$ openstack server list
+--------------------------------------+-----------------------------+---------+------------
| ID                                   | Name                        | Status  | Networks
+--------------------------------------+-----------------------------+---------+------------
| 8de97c6f-bd4e-4f23-b8ee-1d9841082760 | test_7                      | ACTIVE  | Internal=19
| 120fc769-ec99-4025-b456-320d8a17a158 | test_6                      | ACTIVE  | Internal=19
| e93b3344-6d78-4273-880f-220b7fbec417 | test_5                      | ACTIVE  | Internal=19
+--------------------------------------+-----------------------------+---------+------------
```

Get the host of the `test_5` server:

```
$ openstack server show e93b3344-6d78-4273-880f-220b7fbec417
+-------------------------------------+-------------------------------------------------------
| Field                               | Value
+-------------------------------------+-------------------------------------------------------
| OS-DCF:diskConfig                   | AUTO
| OS-EXT-AZ:availability_zone         | nova
| OS-EXT-SRV-ATTR:host                | hc1.example_host
```

The above output of `openstack server show` is truncated.

Find the compute node to migrate to:

```
$ openstack compute service list
+----+----------------+-----------------------+----------+---------+-------+--------------
| ID | Binary         | Host                  | Zone     | Status  | State | Updated At
+----+----------------+-----------------------+----------+---------+-------+--------------
| 18 | nova-scheduler | hc1.example_host      | internal | enabled | up    | 2020-09-14T21:
| 57 | nova-scheduler | hc2.example_host      | internal | enabled | up    | 2020-09-14T21:
| 66 | nova-scheduler | hc3.example_host      | internal | enabled | up    | 2020-09-14T21:
|  3 | nova-conductor | hc1.example_host      | internal | enabled | up    | 2020-09-14T21:
|  9 | nova-conductor | hc2.example_host      | internal | enabled | up    | 2020-09-14T21:
| 15 | nova-conductor | hc3.example_host      | internal | enabled | up    | 2020-09-14T21:
| 24 | nova-compute   | hc3.example_host      | nova     | enabled | up    | 2020-09-14T21:
| 27 | nova-compute   | hc1.example_host      | nova     | enabled | up    | 2020-09-14T21:
| 30 | nova-compute   | hc2.example_host      | nova     | enabled | up    | 2020-09-14T21:
+----+----------------+-----------------------+----------+---------+-------+--------------
```

You can either choose the host to migrate to or one could be automatically selected. The destination host should have a functioning `nova-compute` service running.

Before migrating to a specific host, ensure the host has enough resources:

```
$ openstack host show hc2.example_host
+-----------------------+------------+-----+-----------+---------+
| Host                  | Project    | CPU | Memory MB | Disk GB |
+-----------------------+------------+-----+-----------+---------+
| hc2.example_host      | (total)    |   8 |     64243 |    2682 |
| hc2.example_host      | (used_now) |   0 |     10240 |       0 |
| hc2.example_host      | (used_max) |   0 |         0 |       0 |
+-----------------------+------------+-----+-----------+---------+
```

Perform the migration to a specific host:

```
$ openstack --os-compute-api-version 2.79 server migrate \
--live-migration --host hc2.example_host \
e93b3344-6d78-4273-880f-220b7fbec417
```

Note that to use the `--host` flag, the Nova API version must be specified using `--os-compute-api-version`. In this case, the maximum version that can be used is `2.79`. More on the Nova API version release history is here.

Running `openstack server list` should show the instance status as `MIGRATING`:

```
$ openstack server list
+--------------------------------------+----------------------------+-----------+----------
| ID                                   | Name                       | Status    | Networks
+--------------------------------------+----------------------------+-----------+----------
| 8de97c6f-bd4e-4f23-b8ee-1d9841082760 | test_7                     | ACTIVE    | Internal=
| 120fc769-ec99-4025-b456-320d8a17a158 | test_6                     | ACTIVE    | Internal=
| e93b3344-6d78-4273-880f-220b7fbec417 | test_5                     | MIGRATING | Internal=
+--------------------------------------+----------------------------+-----------+----------
```

Confirm the instance has been migrated to the destination host:

```
$ openstack server show e93b3344-6d78-4273-880f-220b7fbec417
+-------------------------------------+-------------------------------------------------------
| Field                               | Value
+-------------------------------------+-------------------------------------------------------
| OS-DCF:diskConfig                   | AUTO
| OS-EXT-AZ:availability_zone         | nova
| OS-EXT-SRV-ATTR:host                | hc2.example_host
| OS-EXT-SRV-ATTR:hypervisor_hostname | hc2.example_host
| OS-EXT-SRV-ATTR:instance_name       | instance-0000008d
| OS-EXT-STS:power_state              | Running
| OS-EXT-STS:task_state               | None
| OS-EXT-STS:vm_state                 | active
```

```
| OS-SRV-USG:launched_at               | 2020-08-24T21:51:42.000000
| OS-SRV-USG:terminated_at             | None
| accessIPv4                           |
| accessIPv6                           |
| addresses                            | Internal=192.168.0.186
| config_drive                         |
| created                              | 2020-08-24T21:51:16Z
| flavor                               | hc1.small (hc1.small)
| hostId                               | a4a089ceb3b247eeb47dde6f58ed85444cf18bf763453e7fbf77
| id                                   | e93b3344-6d78-4273-880f-220b7fbec417
| image                                | CentOS 8 (ce8-x86_64)
| key_name                             | nw_1
| name                                 | test_5
| progress                             | 0
| project_id                           | 5ad1f9e795604f4390d274d7388c4b9f
| properties                           |
| security_groups                      | name='basic_webserver_group'
| status                               | ACTIVE
| updated                              | 2020-09-15T21:53:49Z
| user_id                              | 43317575cccc440fbcb38a1f23b45125
| volumes_attached                     |
+--------------------------------------+-----------------------------------------------------
```

Troubleshoot instance issues

> ***Note***
>
> This section will be filled out as common scenarios occur.

Upload images

To create an image, use:

```
openstack image create
```

For assistance with the CLI options, use:

```
openstack help image create
```

OpenStack provides an operating system called CirrOS that has the minimum requirements to be an operating system that is generally used to test instance creation.

This example will explain how to upload the CirrOS image into glance.

To get this CirrOS image into the glance service, you'll need to first download it to where the OpenStackClient lives, then the base command `openstack image create` can be used to upload the image.

Download CirrOS and upload into glance:

```
# grab the latest CirrOS image
$ wget http://download.cirros-cloud.net/0.5.1/cirros-0.5.1-x86_64-disk.img

# upload the image into glance
$ openstack image create cirros --container-format bare --disk-format
qcow2 --file PATH_TO_CIRROS_IMAGE
```

List the newly uploaded image:

```
$ openstack image list
+--------------------------------------+---------------------------------+--------+
| ID                                   | Name                            | Status |
+--------------------------------------+---------------------------------+--------+
| fa8eb9bd-9ccc-4d3f-b87b-6edb5450a57a | cirros                          | active |
+--------------------------------------+---------------------------------+--------+
```

Create a private network

Listed are the steps needed to create a private network and connect it to the provider network. Variables are presented in all capital and should be replaced accordingly.

Create a network:

```
$ openstack network create NETWORK_NAME
```

Create a subnet on that network:

```
$ openstack subnet create --subnet-range 10.0.0.0/24 --network NETWORK_NAME
SUBNET_NAME
```

Create a router that will connect to an external, public-facing network:

```
$ openstack router create ROUTER_NAME
```

Add the subnet to the router:

```
$ openstack router add subnet ROUTER_NAME SUBNET_NAME
```

Add the external network gateway:

```
$ openstack router set --external-gateway EXTERNAL_NETWORK_UUID \
ROUTER_NAME
```

Create security groups

Here's an example that opens inbound traffic for all IPs on ports 80 and 443.

Create a security group where **SECURITY_GROUP** is the name of the security group:

```
$ openstack security group create SECURITY_GROUP
```

List security groups:

```
$ openstack security group list
+--------------------------------------+----------------------+-------------------------
| ID                                   | Name                 | Description
+--------------------------------------+----------------------+-------------------------
| 8639e3c5-47ce-4072-a1f5-1c1e931a8f75 | default              | Default security group
| ebffcf78-52d9-436c-81db-5ea788a0c33d | devstack             |
| ec8a02ba-4bc2-4b78-a555-902caead87fe | basic_webserver_group | This will open standard por
+--------------------------------------+----------------------+-------------------------
```

List details of a security group:

```
$ openstack security group show UUID
```

Open ports 80 and 443 on all IP ranges for ingress TCP traffic:

```
$ openstack security group rule create --remote-ip 0.0.0.0/0 \
--dst-port 80:80 --ingress --protocol tcp SECURITY_GROUP

$ openstack security group rule create --remote-ip 0.0.0.0/0 \
--dst-port 443:443 --ingress --protocol tcp SECURITY_GROUP
```

Add SSH public key

It is recommended an SSH public key be uploaded and this is possible through the OSC.

Running `openstack keypair create KEY_NAME`, where KEY_NAME is the name of the SSH key pair, is sufficient for generating an SSH private and public key. The output will return the private key, which should be kept somewhere private and inaccessible to others.

You can also upload a public key from a key pair using `openstack keypair create --public-key PATH_TO_PUBLIC_KEY KEY_NAME`, where PATH_TO_PUBLIC_KEY is the file path to the public key.

Collect details about OpenStack environment

Show the role of each hardware node in an OpenStack cloud

```
$ openstack host list
+------------------------+-----------+----------+
| Host Name              | Service   | Zone     |
+------------------------+-----------+----------+
| hc1.edu.learnstack.org | scheduler | internal |
| hc2.edu.learnstack.org | scheduler | internal |
| hc3.edu.learnstack.org | scheduler | internal |
| hc1.edu.learnstack.org | conductor | internal |
| hc2.edu.learnstack.org | conductor | internal |
| hc3.edu.learnstack.org | conductor | internal |
| hc3.edu.learnstack.org | compute   | nova     |
| hc1.edu.learnstack.org | compute   | nova     |
| hc2.edu.learnstack.org | compute   | nova     |
+------------------------+-----------+----------+
```

Backup OpenStack Service and Ceph Configurations

This guide will explain how to make backups of OpenStack and Ceph configurations.

Background Information

Our Private Cloud Core deployments make use of kolla-ansible and ceph-ansible to deploy the OpenStack services that comprise your cloud and the storage backend, Ceph.

OpenStack service configurations are kept within MariaDB databases as well as within the filesystem. An OpenStack service would be `nova`, `cinder`, or `glance`, to name a few examples.

Backup OpenStack services using kolla-ansible

Kolla-ansible allows you to back up the MariaDB databases associated with OpenStack services by using MariaDB's `mariabackup` function.

NOTE! - A good backup policy is to not store backups in the same location where production data lives, but rather to store backups offsite.

For an authoritative source of information on kolla-ansible and using mariabackup see these documentation links:

- kolla-ansible
- MariaDB database backup and restore

The MariaDB database backup and restore guide goes into full detail on how to enable backups, create them, and restore the backups.

Using kolla-ansible

To start using kolla-ansible, an environment needs to be created. This section explains the steps needed to create that environment.

To get a quick idea of what is required, here is a high-level overview of the steps:

```
# Copy kolla-ansible configuration from fm-deploy docker container
$ docker cp fm-deploy:/opt/kolla-ansible /opt/kolla-ansible

# Navigate to /opt/kolla-ansible
$ cd /opt/kolla-ansible

# Initialize a Python virtual environment
$ virtualenv .venv

# Activate the virtual environment
$ source .venv/bin/activate

# Install kolla-ansible using requirements.txt
$ pip install -r requirements.txt

# Set the SSH private key so kolla-ansible can connect to each host
$ export EXTRA_OPTS="--private-key /root/.ssh/fm-deploy"
```

The above takes care of preparing the kolla-ansible environment.

These steps are explained in more detail below.

Prepare and use kolla-ansible

Follow these steps to learn how to prepare and use kolla-ansible.

Relevant files

- Globals file: `/etc/kolla/globals.yml`
- Iventory file: `/etc/fm-deploy/kolla-ansible-inventory`

Note! – The above files may not be present on each host. Inspect each host until you find the above, and perform the kolla-ansible run from this host.

Preparation

Step 1 - Prepare environment

From the Docker container called `fm-deploy`, copy `/opt/kolla-ansible` to the local filesystem:

```
$ docker cp fm-deploy:/opt/kolla-ansible /opt/kolla-ansible
```

Step 2 – Prepare Python virtual environment and install requirements

Create a Python virtual environment from which `kolla-ansible` will be used:

```
$ cd /opt/kolla-ansible
$ virtualenv .venv
$ source .venv/bin/activate
(.venv) $ pip install -r requirements.txt
```

Step 3 – Export the private key

In order for kolla-ansible to make changes to each host, the SSH private key path needs to be set as an environment variable:

```
$ export EXTRA_OPTS="--private-key /root/.ssh/fm-deploy"
```

Backup kolla-ansible Files

There are important files related to kolla-ansible that should be backed up.

These files are:

- `/etc/kolla/globals.yml`
- `/etc/kolla/passwords.yml`

Should they be lost, it will not be possible to manage the OpenStack cloud.

Create copies of these files and store them in an offsite backup location.

NOTE! – Due to the sensitive nature of these files, ensure the files are transported using an encrypted connection and are stored in a secure, private location.

Create OpenStack Database Backups

With kolla-ansible prepared, you can create a full backup of all OpenStack service databases using `mariabackup`.

To create backups of the OpenStack databases, use:

```
$ kolla-ansible -i /etc/fm-deploy/kolla-ansible-inventory <command>
```

Note that `<command>` is a placeholder for the kolla-ansible command to run and in this case is `mariadb_backup`.

The full command to create database backups is:

```
$ kolla-ansible -i /etc/fm-deploy/kolla-ansible-inventory mariadb_backup
```

This will create a Docker volume called **mariadb_backup** that can be used to restore OpenStack databases.

The output of the `kolla-ansible` run will let you know to what host the backups were sent.

To see the Docker volume, SSH into the host where the backups were made, and run:

```
# docker volume ls | grep mariadb_backup
local               mariadb_backup
```

Restore OpenStack Database Backups

This procedure outlines how to perform a full database restore. An incremental restoration can also be completed.

See MariaDB's documentation for information on how to use `mariabackup` to make incremental backups.

The previous `kolla-ansible` run created a Docker volume called **mariadb_backup**. This volume can be used to restore the OpenStack service databases.

Step 1 – Create Docker container

To restore this backup, create a new Docker container using the previously created volume:

```
# docker run --rm -it --volumes-from mariadb --name dbrestore \
--volume mariadb_backup:/backup kolla/centos-binary-mariadb:train-centos8 \
/bin/bash
```

Step 2 – Perform full backup of databases

Once in that container, these series of commands can be run to perform a full backup of all OpenStack service databases:

```
$ cd /backup/
$ rm -rf /backup/restore/
$ mkdir -p /backup/restore/full
$ gunzip mysqlbackup-10-08-2020-1597091449.qp.xbc.xbs.gz
$ mbstream -x -C /backup/restore/full < mysqlbackup-10-08-2020-1597091449.qp.xbc.xbs
$ mariabackup --prepare --target-dir /backup/restore/full
```

Step 3 – Stop MariaDB

Once the above commands are successfully run, stop the MariaDB Docker instance using:

```
# docker stop mariadb
```

Step 4 – Restore database backups

Navigate back into the container with the mariabackup volume mounted and either move or remove the contents of `/var/lib/mysql` and then move the contents of `/backup/restore/full` into `/var/lib/mysql`:

```
$ mkdir /backup/mariadb_original/
$ mv -v /var/lib/mysql/{*,\.[^\.]*} /backup/mariadb_original/
$ mv -v /backup/restore/full/{*,\.[^\.]*} /var/lib/mysql/
```

Step 5 – Start MariaDB

Finally, start the MariaDB Docker container:

```
# docker start mariadb
```

OpenStack Configuration Backups

In addition to backing up the MariaDB OpenStack databases, it is imperative the OpenStack service files are backed up. Services include `nova`, `swift`, and `glance`, to name a few.

All service configuration files are stored on the hardware nodes within the folder `/etc/kolla/`.

To backup the service's configuration files, copy `/etc/kolla` to an offsite backup location.

Ceph Configuration Backups

The configuration files for Ceph are stored on each hardware node within the folder `/etc/ceph`. Backing up this folder is sufficient for backing up Ceph's configuration.

To backup Ceph's configuration, copy `/etc/ceph` to an offsite backup location.

Backing up Client Data and Disaster Recovery

This guide will talk about ways client data can be backed up and how to restore that data in the event it becomes necessary.

It is suggested a good backup policy be in place in the event data needs to be restored. This policy will depend on various factors, such as importance of data, how much data needs to be stored, how far back in time to store data, what data to keep backups of, cost, and other factors that may be relevant to your OpenStack cloud.

Ceph acts as the distributed storage backend for this deployment of OpenStack. Ceph is naturally self-healing and there is no single point of failure, however it is still possible for it to fail, although this will be a rare occurence. The more replicas used will only decrease the likelihood of failure at the expense of cost. This deployment of Ceph uses three replicas and is generally recommended as a good starting point.

If data loss is of the utmost importance, it is recommended that RBD mirroring within Ceph be setup. This means another Ceph cluster will have to exist and the data be mirrored between them. The additional ceph cluster can be setup in another data center to further decrease the failure domain.

Where is client data currently stored?

All OpenStack client data is stored in Ceph pools.

This section will explain how to see the data stored in Ceph.

By running `rados lspools` from one of the OpenStack hardware nodes, you can see the individual ceph pools and what data is stored in each pool.

The following are the ceph pools where data is stored:

```
# rados lspools
device_health_metrics
images
volumes
vms
backups
metrics
manila_data
manila_metadata
.rgw.root
default.rgw.log
default.rgw.control
default.rgw.meta
default.rgw.buckets.index
```

The configuration for these pools is maintained using `ceph-ansible`.

You can see the contents of each pool by running `rbd ls -l POOL_NAME`.

An example listing the **images** pool:

```
# rbd ls -l images
NAME                                        SIZE     PARENT  FMT  PROT  LOCK
20b56b2d-1af5-46e1-a5c6-fa1f9a45245d        500 MiB            2
20b56b2d-1af5-46e1-a5c6-fa1f9a45245d@snap   500 MiB            2  yes
26a0fde5-69e7-4d85-ae4e-e167e295ecfa            0 B            2
26a0fde5-69e7-4d85-ae4e-e167e295ecfa@snap       0 B            2  yes
```

How to create backups of volumes

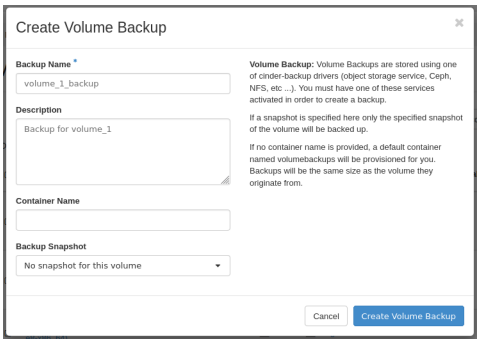
Using Horizon

Volumes can be backed up using Horizon. When backups are made they will be created within the **backups** ceph pool.

To create volume backups in Horizon, navigate to the **Volumes** tab, then to the next **Volumes** tab. This will display current volumes.

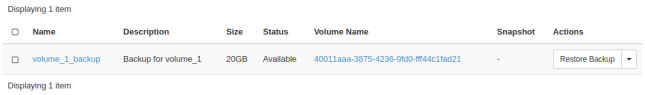
Find the dropdown next to the volume to back up, and click the “Create Backup” button.

Fill out the relevant details:



The volume backup should show as in progress:

And when done it should show listed like so:



Using OpenStackClient

Volume backups can also be created using OpenStackClient.

The following explains how to list volumes and make a volume backup.

List volumes and obtain UUID of volume to back up:

```
$ openstack volume list --fit-width
+--------------------------------------+-----------------------+-----------+------+---------
| ID                                   | Name                  | Status    | Size | Attached
+--------------------------------------+-----------------------+-----------+------+---------
| 40011aaa-3875-4236-9fd0-fff44c1fad21 |                       | in-use    |   20 | Attached
| 663419d7-df14-4472-9eb2-1f4c976103e9 | CentOS 8 (ce8-x86_64) | in-use    |   20 | Attached
|                                      |                       |           |      | /dev/vda
| e9d98e7b-5837-45bc-a0b1-5d9e50d7e686 |                       | in-use    |   20 | Attached
| 057b53e7-eba9-4d2d-bec3-184240c59b29 |                       | available |   20 |
| 15848ac7-67db-460c-8be1-be1dcbb286f0 |                       | available |   20 |
+--------------------------------------+-----------------------+-----------+------+---------
```

Create volume backup of volume UUID `663419d7-df14-4472-9eb2-1f4c976103e9`:

```
$ openstack volume backup create 663419d7-df14-4472-9eb2-1f4c976103e9 --force
+-------+--------------------------------------+
| Field | Value                                |
+-------+--------------------------------------+
| id    | 0481a8ce-e571-45bc-b133-ac5496dce181 |
| name  | None                                 |
+-------+--------------------------------------+
```

NOTE!: The `--force` flag is needed to make a backup of a volume that is in use.

List volume backups:

```
$ openstack volume backup list
+--------------------------------------+-----------------+----------------------+-----------+
| ID                                   | Name            | Description          | Status    |
+--------------------------------------+-----------------+----------------------+-----------+
| 0481a8ce-e571-45bc-b133-ac5496dce181 | None            | None                 | available |
+--------------------------------------+-----------------+----------------------+-----------+
```

And finally for more detail on the volume backup created, you can use `openstack volume backup show UUID` where **UUID** is the volume backup UUID:

```
$ openstack volume backup show 0481a8ce-e571-45bc-b133-ac5496dce181
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| availability_zone     | nova                                 |
| container             | backups                              |
| created_at            | 2020-12-15T16:55:39.000000           |
| data_timestamp        | 2020-12-15T16:55:39.000000           |
| description           | None                                 |
| fail_reason           | None                                 |
| has_dependent_backups | False                                |
| id                    | 0481a8ce-e571-45bc-b133-ac5496dce181 |
| is_incremental        | False                                |
| name                  | None                                 |
| object_count          | 0                                    |
| size                  | 20                                   |
| snapshot_id           | None                                 |
| status                | available                            |
| updated_at            | 2020-12-15T16:56:09.000000           |
| volume_id             | 663419d7-df14-4472-9eb2-1f4c976103e9 |
+-----------------------+--------------------------------------+
```

Backing up ceph pool data

Depending on your needs for data protection, it may be useful to back up data in the Ceph pools to a third party, such as an Amazon S3 bucket for example.

What to do in the event of a hard drive failure?

There is no system in place that monitors for hardware failures, however it something being considered for future releases.

If it has been determined a hard drive has failed, a ticket will need to be created from the Flex Metal Central control panel. This ticket will be routed to our data center team who will replace the failed drive and will alert you when this task is done.

For now, monitoring of the hardware is something that will need to be set up. Multiple monitoring solutions exist already. Icinga or Nagios are two options that immediately stand out.

OpenStack Hardware Failures

The purpose of this guide is to go over failure and maintenance scenarios in the OpenStack cloud that could occur and what to do to address them.

Things can and will go wrong. It is good to be prepared for these events.

What should be done should a hardware node fail?

If a hardware node fails or needs to come down for maintenance, you should know what steps to take. Depending on the maintenance required, you will either require the assistance of our data center staff or you can do the maintenance yourself.

Planned maintenance

This section describes the steps needed to take a hardware compute node out of the cloud in the event work needs to be done on it or the cloud needs to be reduced in size.

NOTE! - If you know a node or nodes need maintenance that require a hardware modification you'll need to create a ticket from the Flex Metal Central control panel to our data center staff to perform that task for you.

For official documentation on this subject, see OpenStack's Compute Node Failures and Maintenance guide.

The general work flow for bringing a compute node down will involve first disabling that node, finding the instances on that node, migrating those instances to another node, and removing any ceph Object Storage Daemons (OSDs). Optionally, you can migrate the instances back to the original node when the maintenance is done.

OpenStackClient will be required to perform the maintenance.

Procedure for removing a compute node

Start with disabling the `nova-compute` service on the appropriate node:

```
$ openstack compute service set --disable --disable-reason \
maintenance COMPUTE_NODE_NAME nova-compute
```

List the instances on that node:

```
$ openstack server list --host COMPUTE_NODE_NAME --all-projects
```

Migrate the instances to another node:

```
$ openstack server migrate INSTANCE_UUID --live-migration
```

NOTE - This deployment of OpenStack is using ceph as the backend shared storage so there is no need to pass the `--block-migration` flag to `openstack server migrate`.

Because OpenStack has been deployed using `kolla-ansible`, each OpenStack service runs in a docker container.

Stop the nova_compute docker container:

```
# docker stop nova_compute
```

Perform the needed maintenance, and then restart the nova_compute service:

```
# docker start nova_compute
```

Verify the `nova_compute docker` container is running:

```
# docker ps | grep nova_compute
286e1b2e2ae5        kolla/centos-binary-nova-compute:train-centos8
"dumb-init --single-…"   2 months ago        Up 18 minutes
nova_compute
```

Finally, verify the nova service has connected to the messaging service, AMQP:

```
$ grep AMQP /var/log/kolla/nova/nova-compute.log
```

Unplanned maintenance

There are times where unplanned maintenance is required. This section will describe what can be done in the event a compute node goes down unexpectedly.

The primary concern is that instances associated with the compute node that has failed will no longer work.

Ceph failure scenarios and recovery

Ceph by nature is resilient to hardware failure and self-healing.

The primary concern with ceph is failed hard drives. How can an operator be alerted to a failed hard drive? Will ceph continue to function if a drive is lost?

Generally, ceph will continue to function if a drive is lost, however the drive should be replaced as soon as possible.

How do you know if a hard drive has failed?

Currently there is no monitoring for failed ceph drives, however the intention is to monitor for these events in the future. Due to this, it is recommended monitoring of drives be put into place. Software such as Icinga or Nagios are viable options for monitoring.

If it is suspected a drive has failed, you should first determine if this really is the case.

The overall procedure for determining if a drive has failed is to:

- Check Ceph health
- See if the OSD associated with the drive in question can be started if it is stopped
- Check the OSD's mount point using *df -h*
- Use `smartctl` on the drive in question

The following explains these steps in more detail.

Procedure

Reference:

https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/4/html/operations_guide/handling-a-disk-failure

From one of the hardware nodes, perform the following checks:

Check if ceph is healthy:

```
# ceph health
```

Find the location of the OSD within the CRUSH map:

```
# ceph osd tree | grep -i down
```

On the node that houses the OSD, try to start the OSD using `systemctl` where **OSD** is a placeholder for the actual OSD identifier:

```
# systemctl start ceph-osd@OSD.service
```

The systemctl unit file for the OSD will vary depending on which OSD has failed. In this case the systemctl unit file is called `ceph-osd@0.service`.

If a hard drive has failed, our data center team will need to replace it. A ticket will need to made in Flex Metal Central to alert of team of the failure. The drive or drives will be replaced by our team.

Adding and Removing OpenStack Hardware Nodes

If you find the cloud growing, it may come time to add another node to the cloud. Similarly, you may not need hardware any longer.

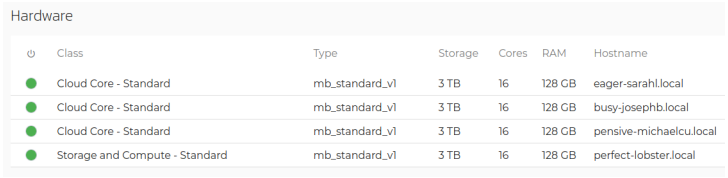
This guide will demonstrate how to add or remove a node from an OpenStack cloud. This is only applicable to those who have added an additional **Storage and Compute** node via Flex Metal Central. If you have not added any additional nodes that means each node is a control plane node and those nodes should **not** be removed.

Types of Hardware Nodes

The following are the current nodes provided through Flex Metal Central.

- **Cloud Core - Standard**
- **Storage and Compute - Standard**

Here is a screenshot showing three **Cloud Core** nodes and one **Storage and Compute** node:



Note that the three **Cloud Core** nodes are part of the original cloud. The **Storage and Compute** node was added to the cloud.

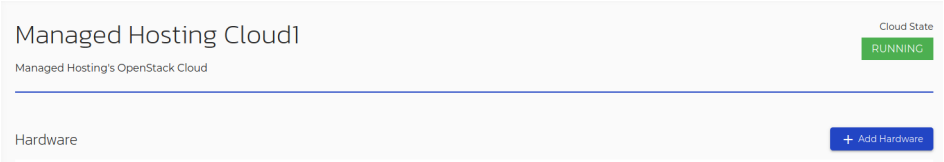
Adding a node

Hardware nodes can be added to an existing cloud using Flex Metal Central.

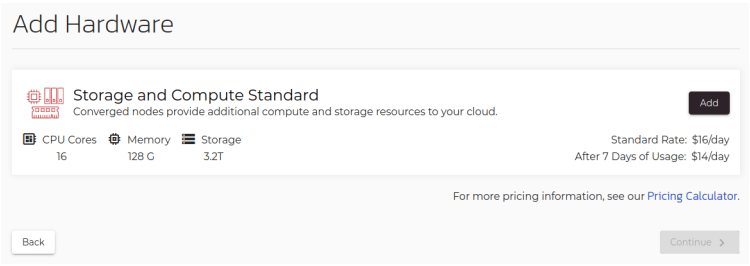
There is currently only one node type that can be added which is **Storage and Compute - Standard**. Future updates will see new node types.

Once logged into Flex Metal Central, navigate to the cloud you are working with. Find the button near the top right that says “Add Hardware”. Clicking this button will allow you to add additional hardware to the cloud.

Add Hardware:



Add a new Storage and Compute - Standard node:



Removing a Storage and Compute Node

There is no automated way to safely remove a compute node. In order to safely remove it you will need to ensure all data is removed from the node, including instances, Ceph data, and anything else.

NOTE! – When you are ready to remove the node, you can do so by clicking the three vertical dots, then the **Remove** link, next to the node in Flex Metal Central. You will see a screen asking you to confirm removal of the node which you must confirm before doing so. It is strongly advised you are sure everything is removed from the node before doing this.

This section will demonstrate the steps needed to remove a hardware node. Note that this is a manual process.

NOTE! – If a compute node needs to be removed and you do not feel comfortable migrating data off of it, it is recommended you reach out to support who can help perform this task for you.

There a number of things that need to be checked and considered before removing a hardware node. In addition the type of hardware node will dictate what needs to occur. This section will describe the general process to remove a **Storage and Compute** node.

Requirements before removing a node

- This task can only be performed using the command line.
- Instances running on the node will need to be migrated to another compute node.
- Ceph OSDs need to be removed from ceph.
- Ceph ansible needs its inventory updated
- OpenStack needs to be updated that a node will be removed.
- Kolla-ansible will need its inventory file updated to reflect removal of the node

The following will go into detail on how to perform each step.

Remove a Storage and Compute node

With the above in mind, this section will expand upon the steps in the previous section.

Reference: https://docs.openstack.org/kolla-ansible/latest/user/adding-and-removing-hosts.html

---

Procedure

Live migrate instances:

You will want to first find all instances on the node, confirm the receiving node has enough resources to host the instances, then perform a live migration of those instances.

NOTE! – Since Ceph is used for disk storage you do not need to account for disk space when moving instances from one compute node to another. You will only need to account for RAM and VCPUs.

Step 1 – Disable instance scheduling on node

The node being removed will need to have instance scheduling disabled. This causes it so no new instances can be created on this node.

To perform this, run `openstack compute service set HOST nova-compute --disable`.

For example, to disable instance scheduling for the `perfect-lobster.local` host run:

```
$ openstack compute service set perfect-lobster.local nova-compute --disable
```

Step 2 – Collect instance information

This guide will assume you have a compute node called `perfect-lobster.local` that needs to be removed from the OpenStack cloud.

In addition, there are three instances on this node:

- **migrate_me-1**
- **migrate_me-2**
- **migrate_me-3**

You can use `$ openstack server list` with additional flags to obtain the details of the instances on `perfect-lobster.local`.

For example:

```
$ openstack server list --host perfect-lobster.local -f value -c ID -c Name
a248c6b2-c4f5-4b5a-82ca-0dc71edd9757 migrate_me-2
ad489cd4-4c0c-42c7-aeae-21da6c00693b migrate_me-3
d143fb86-c7a4-4bc8-b6dc-7969097ef34b migrate_me-1
```

The above makes use of **value** output formatting and specifies the **ID** and **Name** columns as output.

Step 3 – Determine compute host

With the instance information acquired, the next step is to determine what compute host these instances can be migrated to.

To obtain all compute hosts, you can use `$ openstack compute service list`.

For example:

```
$ openstack compute service list
+----+----------------+-------------------------+----------+---------+-------+--------------
| ID | Binary         | Host                    | Zone     | Status  | State | Updated At
+----+----------------+-------------------------+----------+---------+-------+--------------
| 12 | nova-scheduler | eager-sarahl.local      | internal | enabled | up    | 2021-01-28T23
| 51 | nova-scheduler | busy-josephb.local      | internal | enabled | up    | 2021-01-28T23
| 66 | nova-scheduler | pensive-michaelcu.local | internal | enabled | up    | 2021-01-28T23
|  3 | nova-conductor | eager-sarahl.local      | internal | enabled | up    | 2021-01-28T23
| 12 | nova-conductor | busy-josephb.local      | internal | enabled | up    | 2021-01-28T23
| 27 | nova-conductor | pensive-michaelcu.local | internal | enabled | up    | 2021-01-28T23
| 30 | nova-compute   | eager-sarahl.local      | nova     | enabled | up    | 2021-01-28T23
```

```
| 33 | nova-compute   | busy-josephb.local      | nova     | enabled | up    | 2021-01-28T23
| 36 | nova-compute   | pensive-michaelcu.local | nova     | enabled | up    | 2021-01-28T23
| 37 | nova-compute   | perfect-lobster.local   | nova     | enabled | up    | 2021-01-28T23
+----+----------------+-------------------------+----------+---------+-------+--------------
```

What is shown in the above output is there are four total compute hosts. One of them (`perfect-lobster.local`) is being removed from the cluster.

This example will select `eager-sarahl.local` as the new host to migrate the instances to.

Step 4 – Check available host resources

You must first ensure this host has enough resources to contain these instances. To do so, you can run `$ openstack host show HOSTNAME`.

For example:

```
$ openstack host show eager-sarahl.local
+--------------------+----------------------------------+-----+-----------+---------+
| Host               | Project                          | CPU | Memory MB | Disk GB |
+--------------------+----------------------------------+-----+-----------+---------+
| eager-sarahl.local | (total)                          |  16 |    121026 |   11923 |
| eager-sarahl.local | (used_now)                       |   4 |      6144 |      25 |
| eager-sarahl.local | (used_max)                       |   2 |      2048 |      20 |
| eager-sarahl.local | b9e8639372014c0b85cbfaffa6e1b5a8 |   2 |      2048 |      20 |
+--------------------+----------------------------------+-----+-----------+---------+
```

Step 5 – Find instance resource usage

To ensure the instances will fit you will also need to know what resources they consume. This means you will need to know the flavor set for each instance.

You can run something like this to get the flavor for each instance on the `perfect-lobster.local` node:

```
$ openstack server list --host perfect-lobster.local -f value -c ID -c Name -c Flavor
a248c6b2-c4f5-4b5a-82ca-0dc71edd9757 migrate_me-2 hc1.micro
ad489cd4-4c0c-42c7-aeae-21da6c00693b migrate_me-3 hc1.micro
d143fb86-c7a4-4bc8-b6dc-7969097ef34b migrate_me-1 hc1.micro
```

This shows the `hc1.micro` flavor is used by each instance on this host.

To know what resources are consumed by this flavor use `$ openstack flavor show FLAVOR`:

```
$ openstack flavor show hc1.micro
+----------------------------+-----------+
| Field                      | Value     |
+----------------------------+-----------+
| OS-FLV-DISABLED:disabled   | False     |
| OS-FLV-EXT-DATA:ephemeral  | 0         |
| access_project_ids         | None      |
| disk                       | 10        |
| id                         | hc1.micro |
| name                       | hc1.micro |
| os-flavor-access:is_public | True      |
| properties                 |           |
| ram                        | 1024      |
| rxtx_factor                | 1.0       |
| swap                       | 1024      |
| vcpus                      | 1         |
+----------------------------+-----------+
```

This output reveals the amount of VCPUs, RAM, and disk space allocated.

To get tidier output of the flavor details you can run:

```
$ openstack flavor show hc1.micro -c disk -c ram -c vcpus
+-------+-------+
| Field | Value |
+-------+-------+
| disk  | 10    |
| ram   | 1024  |
| vcpus | 1     |
+-------+-------+
```

From this information, it can be determined the receiving host will require 3 VCPUs, 3GB of RAM, and 30GB of disk space since there are three instances to migrate and each is using the `hc1.micro` flavor. Remember, however, that disk space does not need to be accounted for since ceph is used for data storage and the data storage is shared across each node.

Step 6 – Live migrate the instances

You can now safely migrate these instances to the `eager-sarahl.local` node.

The base command to perform the live migration is:

```
$ openstack --os-compute-api-version 2.79 server migrate \
--live-migration --host HOSTNAME INSTANCE_UUID
```

The command to live migrate these three instances for this demonstration is:

```
openstack server list --host perfect-lobster.local -f value -c ID | while read id; do
  openstack --os-compute-api-version 2.79 server migrate --live-migration --host eager-sarah
done
```

You can confirm the status of the live migration by using `$ openstack server list`:

```
$ openstack server list
+--------------------------------------+-----------------------------+-----------+----------
| ID                                   | Name                        | Status    | Networks
+--------------------------------------+-----------------------------+-----------+----------
| a248c6b2-c4f5-4b5a-82ca-0dc71edd9757 | migrate_me-2                | ACTIVE    | Internal=
| ad489cd4-4c0c-42c7-aeae-21da6c00693b | migrate_me-3                | MIGRATING | Internal=
| d143fb86-c7a4-4bc8-b6dc-7969097ef34b | migrate_me-1                | MIGRATING | Internal=
| 85033a0f-66c6-41d4-b679-c7350da2685f | openstackclient_js_demo     | ACTIVE    | Internal=
| 0052cd0f-70fb-4cf7-8b13-2bec350c0e51 | openstackclient_jumpstation | ACTIVE    | Internal=
+--------------------------------------+-----------------------------+-----------+----------
```

Here it can be seen two of the instances are being migrated.

Step 7 – Confirm live migration success

To confirm the live migration completed successfully and the instances are on the new host, you can use something similar to:

```
openstack server list -f value -c ID -c Name | grep migrate_me | while read id name; do
  echo "$name $(openstack server show -f value -c 'OS-EXT-SRV-ATTR:host' $id)"
done
```

The following is the actual output from running the above command:

```
migrate_me-2 eager-sarahl.local
migrate_me-3 eager-sarahl.local
migrate_me-1 eager-sarahl.local
```

This indicates the live migration was successful.

Confirm no instances remain on the original host:

```
$ openstack server list --host perfect-lobster.local
```

If the above returns no output, the live migration was a complete success and you can move on to the next step.

Remove Ceph OSDs:

After the instances have been migrated, it is time to remove the node's OSDs from the Ceph cluster.

Reference: https://docs.ceph.com/en/latest/rados/operations/add-or-rm-osds/

This continues to assume the compute host being removed is `perfect-lobster.local`.

You will need to determine what OSDs are on this host and remove them from ceph.

Step 1 – Determine OSDs

From any of the hardware nodes you can use `# ceph osd tree` to find which OSD is on a particular host.

Example:

```
# ceph osd tree
ID  CLASS  WEIGHT    TYPE NAME                   STATUS  REWEIGHT  PRI-AFF
-1         11.64398  root default
-5          2.91100      host busy-josephb
 1    ssd   2.91100          osd.1                   up   1.00000  1.00000
-3          2.91100      host eager-sarahl
 0    ssd   2.91100          osd.0                   up   1.00000  1.00000
-7          2.91100      host pensive-michaelcu
 2    ssd   2.91100          osd.2                   up   1.00000  1.00000
-9          2.91100      host perfect-lobster
 3    ssd   2.91100          osd.3                   up   1.00000  1.00000
```

This indicates the host `perfect-lobster.local` has only one OSD, with **ID** of 3. This is the OSD that will need to be removed.

Step 2 – Remove the OSD

To remove the OSD, use the command `ceph osd out OSD_NUMBER`.

For example:

```
# ceph osd out 3
marked out osd.3.
```

Following that, watch ceph's status using `ceph -w` to ensure the cluster returns to a healthy state (output truncated):

```
# ceph -w
  cluster:
    id:     a08c2963-1d75-42ef-be50-1fc61419623d
    health: HEALTH_WARN
            Degraded data redundancy: 441/13998 objects degraded (3.150%), 17 pgs degraded

  [...]

  io:
    recovery: 68 MiB/s, 14 keys/s, 11 objects/s

  progress:
    Rebalancing after osd.3 marked out (1s)
      [............................]

2021-01-29T15:43:20.780381+0000 mon.eager-sarahl [WRN] Health check update: Degraded data re
```

```
2021-01-29T15:43:20.817317+0000 mon.eager-sarahl [INF] Health check cleared: PG_DEGRADED (wa
2021-01-29T15:43:20.817342+0000 mon.eager-sarahl [INF] Cluster is now healthy
```

The above shows the cluster in a degraded state as the OSD is outed and that Ceph returns to a healthy state. This indicates no issues occurred with outing the OSD.

Step 3 – Stop the OSD

With the OSD removed, the OSD `systemctl` service on the `perfect-lobster.local` host needs to be stopped.

In this case, the unit file for this OSD is called `ceph-osd@3.service`.

Stop the service:

```
# systemctl stop ceph-osd@3.service
```

Step 4 – Remove OSD from ceph configuration

The OSD will now need to be removed from ceph's configuration.

To do so, you will need to use `ceph osd purge OSD_NUMBER --yes-i-really-mean-it`:

```
# ceph osd purge 3 --yes-i-really-mean-it
purged osd.3
```

Step 5 – Remove OSD from Ceph crush map

The OSD will still need to be removed from the crush map.

To remove the `perfect-lobster` OSD from the crush map, use:

```
# ceph osd crush rm perfect-lobster
```

Step 6 – Confirm OSD has been removed

You can use `ceph osd tree` to confirm the OSD from the node has been removed.

For example:

```
# ceph osd tree
ID  CLASS  WEIGHT   TYPE NAME                   STATUS  REWEIGHT  PRI-AFF
-1         8.73299  root default
-5         2.91100      host busy-josephb
 1    ssd  2.91100          osd.1                   up   1.00000  1.00000
-3         2.91100      host eager-sarahl
 0    ssd  2.91100          osd.0                   up   1.00000  1.00000
-7         2.91100      host pensive-michaelcu
 2    ssd  2.91100          osd.2                   up   1.00000  1.00000
```

Here it can be seen the `perfect-lobster.local` host is no longer present in this output.

These steps take care of the Ceph changes that are required.

Clean up services on remaining nodes:

The next steps are to clean up the host from the compute and network service listing within OpenStack.

The following demonstrate removing the host `perfect-lobster.local` from the network agent list and the compute service list.

Remove host from network agent list:

```
openstack network agent list --host perfect-lobster.local -f value -c ID | while read id; do
  openstack network agent delete ${id}
done
```

Remove host from compute service list:

```
openstack compute service list --os-compute-api-version 2.53 --host perfect-lobster.local -f
  openstack compute service delete --os-compute-api-version 2.53 ${id}
done
```

Confirm the `perfect-lobster.local` has been removed from both section by using:

```
$ openstack network agent list --host perfect-lobster.local
$ penstack compute service list --os-compute-api-version 2.53 --host \
perfect-lobster.local
```

Stop the services running on the node (stop the Docker service):

The next step will be to stop the Docker service on the node being removed.

Use the following to stop the Docker service:

```
# systemctl status docker
```

Submit the request in Flex Metal Central to remove the node:

At this point, you are ready to request the node removal in Flex Metal Central.

Login to Flex Metal Central and go the **Manage** section for the cloud. From here you will see a listing of the hardware nodes. Select the node being removed and find the three vertical dots to the right of it. Clicking this will bring up the option to remove the node. Click **Remove** to initiate that process.

NOTE! – At this time, when you request a node to be removed, it will create a ticket and our support team will handle that request manually and follow up when that is done.

Test the remaining nodes to ensure everything still functions:

Once at this point, the node removal is complete. At this time, it is recommended you test general functionality of the cloud, such as can instances still be created, do the hardware nodes still respond to ping, and the like.

Automating OpenStack

It is possible to automate deploying various OpenStack services, such as instance creation. Terraform is software that can be used to do this.

See InMotion Hosting's documentation on using Terraform with OpenStack through Flex Metal Cloud.

Miscellaneous Guides

This guide stores miscellaneous information and may be useful, however the information in this guide is not needed to go through setting up the cloud.

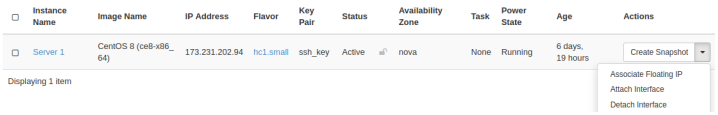
Associate an instance with a private network

This section will explain how to take an instance created on the provider network and associate that with a private network. This may not be a very common task assuming an instance was created on the appropriate network to begin with.

In the event an instance was created but on the wrong network, you can have it associated with another network.

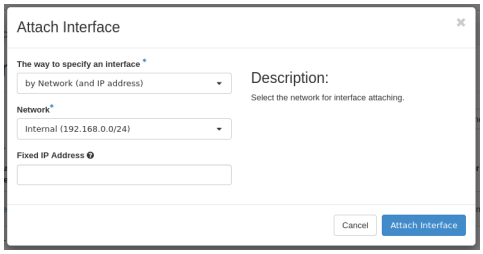
In Horizon, pull up the listing on instances under the **Compute** heading on the left, then find **Instances** under that. Find the instance you're working with then from the dropdown on the far right, choose the **Attach Interface** option.

Attach Interface:



Under the **Network** dropdown choose the appropriate network. This example will choose the **192.168.0.0/24** network.

Attach Interface form:



With that done, the instance is now associated with two networks. To finish moving the instance to the new network, use the same drop down in the instance listing page to detach an interface. Locate the option called **Detach Interface**, then choose the network from the drop down and submit the form to remove it.