Vinoth Kumar Selvaraj

OpenStack Bootcamp

Build and operate a private cloud environment effectively

Packt

OpenStack Bootcamp

Build and operate a private cloud environment effectively

Vinoth Kumar Selvaraj

Packt

BIRMINGHAM - MUMBAI

OpenStack Bootcamp

First published: November 2017

Production reference: 1241117

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78829-330-3

www.packtpub.com

Credits

Author Vinoth Kumar Selvaraj	**Copy Editors** Ulka Manjrekar Vikrant Phadkay
Reviewer Ashutosh Narayan	**Project Coordinator** Shweta H Birwatkar
Commissioning Editor Vijin Boricha	**Proofreader** Safis Editing
Acquisition Editor Chandan Kumar	**Indexer** Aishwarya Gangawane
Content Development Editor Deepti Thore	**Graphics** Tania Dutta
Technical Editor Akash Patel	**Production Coordinator** Nilesh Mohite

About the Author

Vinoth Kumar Selvaraj is an enthusiastic computer science engineer from Tamil Nadu, India. He works as a DevOps engineer at Cloudenablers Inc. As an active moderator on Ask OpenStack, he consistently answers and provides solutions for questions posted on the Ask OpenStack forum. Based on karma points, he has ranked 20 out of 20,000 members in the Ask OpenStack forum. He has also written many OpenStack-related articles for superuser.openstack.org and hosts a dedicated website for his works on OpenStack at http://www.hellovinoth.com/.

You can visit his LinkedIn page at https://www.linkedin.com/in/vinothkumarselvaraj/ and tweet to him @vinoth6664.

Vinoth has also worked as a technical reviewer on books by Packt such as *Openstack Cloud Security*, *Learning OpenStack High Availability*, *Openstack Essentials*, and *Learning OpenStack [Video]*.

Acknowledgments

I would like to dedicate this book to my Amma, Appa, Anna and friends for their love and support. My special thanks to all my mentors that I've had over the years.

Thiruvalluvar

Rathinasabapathy

Krishnakumar Narayanan

Venkatesh Perumal

Satyabrata Chowdhury

Konda Chendil

Vinu Francis

Praveen Tummalapalli

Monisha

Vishnu Prabakaran

and the entire Cloudenablers team

Without learning and support from these teachers, there was no chance of what I have done today, and it is because of them and others that I may have missed on the list here that I feel compelled to pass my knowledge on to those willing to learn.

About the Reviewer

Ashutosh Narayan hails from a small town called Deoghar in Jharkhand, where he grew up. He is currently based in Bengaluru, India. His interests are multifarious and diverse. He loves art, playing musical instruments, nature, photography, blogging, and most of all, spending time with his family.

He holds a bachelor's degree in information science and engineering and is an open source technology enthusiast and contributor with 9+ years of IT experience. He is an OpenStack Foundation Member and has contributed to the openstack-manual project. Ashutosh also became a member as an Individual supporter of LINUX FOUNDATION for a year. He has experience across various domains: programming, Linux system administration, cloud computing, and DevOps. He attends conferences, events, and regular meet-ups from the area of his interests in and around the city, where he shares experiences and lessons on technology.

I would like to thank my beloved mother, Late Smt. Kanchan Narayan and my father, Sri Rajendra Narayan, two of my greatest teachers who have always given me strength. Thanks to my wife, Anjulika, who supported me in spite of all the time I spent away from her, and my two lovable sisters, Anamika and Priyanka. Without them, I am nothing. Finally, thanks to the project coordinators for making this book what it is now.

Customer Feedback

Table of Contents

Preface

OpenStack is a free and opensource software platform for cloud computing, mostly deployed as an infrastructure-as-a-service (IaaS). The Bootcamp approach has short, intensive, and practical content comprising of a lot of real-world examples, *OpenStack Bootcamp* will provide the main architecture of OpenStack clouds, configuration of each OpenStack component and debugging techniques. Besides in-depth coverage of OpenStack technologies, hands-on exercises are also provided to make reader better understand and provide analysis of real-world cloud use cases and operation scenarios, covering design, customization and optimization.

What this book covers

`Chapter 1`, *Day 1 - Build Your Camp*, discusses the design principles of OpenStack and Cloud in general, the types of distributions, and setting up the lab environment for the hands-on chapters ahead.

`Chapter 2`, *Day 2 - Know Your Battalion*, focuses on an overview of each OpenStack core (KeyStone/Glance/Nova/Neutron/Cinder/Swift) components in detail and the real-world comparisons for the reader to understand with the ease and to learn how to use each of them.

`Chapter 3`, *Day 3 - Field Sketch*, looks at the architecture design and how the components of OpenStack are connected, followed by deployment use cases. The reader will understand the high level architectural design of OpenStack and will be able to plan and design their own deployment use cases so they can build their OpenStack cloud.

`Chapter 4`, *Day 4 - How Stuff Works*, will explain the step-by-step process of the VM provision happens in OpenStack when the user initiate VM create from horizon and the interrelationship between each OpenStack services.

`Chapter 5`, *Day 5 - Networking Strategy*, focuses on OpenStack networking in detail and the extended feature available in OpenStack neutron. This chapter covers the role of floating IPs, the available deployment types in the neutron, how the VM packets being encapsulated t to reach the destination VM and the packet flow between VMs and the internet.

`Chapter 6`, *Day 6 - Field Training Exercise*, will deals with the hands training on how to use the OpenStack Horizon for using all the OpenStack core components will be provided in this chapter. The reader will gain the hands-on experience with the OpenStack horizon and understand how each individual component worked interconnected in bringing up the Cloud environment.

`Chapter 7`, *Day 7 - Collective Training*, will provide with the undisclosed tasks for the readers to take this chapter as exam. The Comprehensive practice with admin and end-user use cases will test the reader's ability in understanding the OpenStack environment.

`Chapter 8`, *Day 8 - Build Your OpenStack*, this chapter will guide through a step-by-step package-based installation of the Ocata OpenStack on the Ubuntu operating system.

`Chapter 9`, *Day 9 - Repair OpenStack*, discusses how to start with troubleshooting each OpenStack components and the respective log files to look in. This chapter will also provide the guidance to the reader on how to start with getting help from OpenStack community.

`Chapter 10`, *Day 10 - Side Arms of OpenStack*, will walk through the overview of additional services available in the OpenStack and its scope in high level. The reader will learn the overview of optional services in OpenStack and its interrelation with the OpenStack core components.

What you need for this book

You can benefit greatly by having a system running VirtualBox (or any of its alternatives) and a virtual machine running Ubuntu 16.04 LTS. The commands and resources discussed in this book are best learned when you can execute them on the test system. Familiarity with the Linux command-line and experience with Linux system administration is expected.

Who this book is for

This book is perfect for administrators, cloud engineers, and operators who wants to jump right into practical knowledge, exercises, and solving the basic problems encountered during deployment, in order to get up to speed with the latest release of OpenStack.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "OpenStack services can be installed either as root or as a user with `sudo` permissions."

Any command-line input or output is written as follows:

```
# nano /etc/sysconfig/network-scripts/ifcfg-eth0
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Clicking the **Next** button moves you to the next screen"

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book-what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of. To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message. If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this book from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/OpenStack-Bootcamp`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books-maybe a mistake in the text or the code-we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
Day 1 - Build Your Camp

OpenStack has a very complex architectural design to understand theoretically. I firmly believe that a hands-on experience with OpenStack will help you to understand the OpenStack design a lot better than just reading through the details.

To unleash the power of learning by doing, I purposely chose to deal with this OpenStack setup and installation chapter first. Though this section will deal with the script-based OpenStack installation, using DevStack to help you learn OpenStack design by having practical hands-on experience with OpenStack, once you have a complete understanding of the OpenStack modular design, `Chapter 8`, *Day 8 - Build Your OpenStack,* will guide the users through an actual step-by-step customized installation of OpenStack.

As the book title implies, the bootcamp series is a new concept that is targeted at someone who is looking for absolute knowledge of OpenStack without wasting time by learning ABC on day 1 and XYZ on day 2. So, I focused more on hands-on exercises for the readers instead of starting with the history and evolution of OpenStack.

Design principles for OpenStack

OpenStack has a modular design, and most of the OpenStack projects and services are capable of being used independently. At a high level, OpenStack services are categorized as core services and optional services. As the name states, the core services have the essential functionality of providing IAAS features for OpenStack. The optional services are like the bells and whistles of OpenStack, which provide the extended functionality for the IAAS platform.

The following listed services are the core components of OpenStack:

- Keystone (identity service)
- Glance (image service)
- Nova (compute service)
- Neutron (networking service)

The following listed services are the optional components of OpenStack:

- Cinder (block storage service)
- Horizon (Dashboard)
- Swift (object storage service)
- Other 20+ projects

We will discuss all the essential components of OpenStack briefly in `Chapter 2`, *Day 2 - Know Your Battalion*.

OpenStack distributions

Before we start building our play camp, it would be good to know about some of the familiar OpenStack distributions available, which are as follows:

- Ubuntu OpenStack
- RedHat OpenStack Platform
- SUSE OpenStack Cloud
- Debian OpenStack
- Mirantis OpenStack
- VMware Integrated OpenStack
- Hyper-C
- HPE Helion OpenStack®
- Oracle OpenStack
- Stratoscale
- IBM Cloud Manager with OpenStack and the list goes on

Don't get confuse the these enterprise distributions of OpenStack with the open source release. The preceding distributions are the enterprise editions of OpenStack with a fully integrated, optimized combination for their selected platforms.

Vanilla OpenStack

Vanilla OpenStack generally refers to an OpenStack without any *special* optimization, which is free and open source, and available at `https://docs.openstack.org/`.

We will look at the step-by-step installation procedure of OpenStack in `Chapter 8`, *Day 8 - Build Your OpenStack* after understanding the OpenStack design completely.

As of now, to build our play camp, we will be using the automated, opinionated script to create an OpenStack development environment quickly. There are various options available to build the OpenStack development environment in one go, such as:

- **DevStack**: For Ubuntu (recommended)
- **PackStack**: For CentOS/RHEL

There are also plenty of volunteer scripts available online to install OpenStack quickly with one command.

In this chapter, we will be focusing on the DevStack installation to bring up our play camp to get hands-on with the OpenStack cloud.

DevStack installation

This guide assumes that you have access to a virtual machine that has a Ubuntu 16.04 LTS operating system installed with a minimum of 6 GB RAM and 30 GB HDD.

Downloading and installing the virtual box and creating new virtual machine is not in the scope of this book. There are lots of free tutorials available online for creating your new virtual machine with the aforementioned specifications.

Prepare your virtual machine

To demonstrate a simple working OpenStack cloud using DevStack, you must have the following requirements configured in the VirtualBox environment to start the DevStack installation:

- Ubuntu 16.04 LTS operating system
- 6 GB RAM
- 40 GB disk space
- 2 vCPUs
- 2 NIC (Adapter 1 - NAT network and Adapter 2 - host-only network)

Adding the second adapter to the operating system requires manual configuration changes in the network interface file. Make sure you have added the second interface with DHCP settings in the `/etc/network/interfaces` file, and that both NICs have obtained the IP address.

Then perform an `apt-get update` and a `dist-upgrade` and reboot the machine.

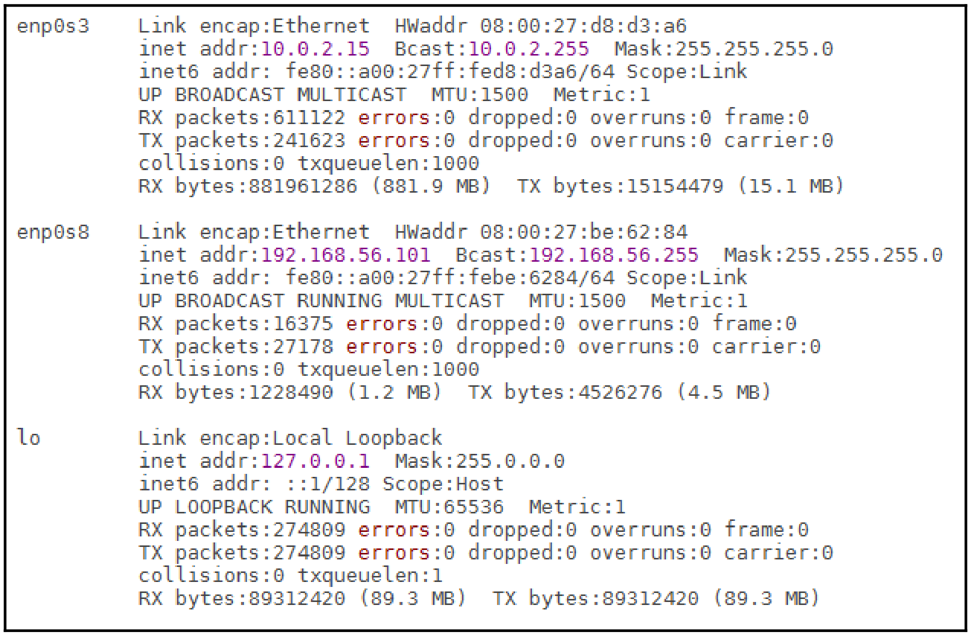
Let's DevStack

Log in to your Ubuntu virtual machine and go through the following steps to complete the prerequisites for installing DevStack.

1. Verify the IP address using the following command:

```
$ ifconfig
```

You will get the following output:



```
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:d8:d3:a6
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fed8:d3a6/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:611122 errors:0 dropped:0 overruns:0 frame:0
          TX packets:241623 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:881961286 (881.9 MB)  TX bytes:15154479 (15.1 MB)

enp0s8    Link encap:Ethernet  HWaddr 08:00:27:be:62:84
          inet addr:192.168.56.101  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:febe:6284/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16375 errors:0 dropped:0 overruns:0 frame:0
          TX packets:27178 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1228490 (1.2 MB)  TX bytes:4526276 (4.5 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:274809 errors:0 dropped:0 overruns:0 frame:0
          TX packets:274809 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:89312420 (89.3 MB)  TX bytes:89312420 (89.3 MB)
```

You can see from the output that the IP address is assigned for both `enp0s3`and `enp0s8` adapters. If you are using the virtual box for the first time, the default IP value will be `10.0.2.x` for the NAT network and `192.168.56.x` for the host-only network.

1. Let's add a stack user. DevStack should be run as a `sudo` user, but not as `root`, so create a `sudo` user named `stack` to run the DevStack installation for building your camp:

   ```
   $ sudo useradd -s /bin/bash -d /opt/stack -m stack
   ```

2. Let's add the stack user to the sudoers. The following command will add the stack user to the sudoers list:

   ```
   $ echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee
   /etc/sudoers.d/stack
   ```

3. Now log in as a stack user and verify that you have appropriate `sudo` privileges:

   ```
   $ sudo su - stack
   $ sudo id
   ```

   At this stage, if the shell is prompted for a password, which means that the `sudo` privileges for the stack user are not configured correctly, ensure that you have followed *step 2* and *step 3* correctly.

4. Let's download DevStack:

   ```
   $ sudo apt-get install -y git-core
   ```

At this stage, you will not be prompted for any password.

The following command will download the DevStack repository into your local machine:

```
$ git clone https://git.openstack.org/openstack-dev/devstack -b
stable/ocata
```

The preceding command will download the stable version of DevStack repository to install OpenStack ocata release. Optionally, you may replace `ocata` with any latest release of OpenStack code name in the preceding command to install the latest OpenStack release.

The downloaded DevStack repository contains the automated script that installs OpenStack and templates for configuration files. Make sure you have working internet connectivity before executing the preceding command:

1. Let's configure DevStack. The downloaded DevStack repository comes with the sample configuration file. By adding a few additional parameters to the sample configuration file, we could use it as an actual configuration file for our default DevStack installation to build our play camp with. The following commands will create a configuration file suitable for our DevStack installation:

   ```
   $ cd devstack
   $ git checkout stable/ocata
   $ cp samples/local.conf .
   $ HOST_IP="192.168.56.101"
   $ echo "HOST_IP=${HOST_IP}" >> local.conf
   ```

In the preceding command, `192.168.56.101` is the IP address of my host-only adapter `enp0s8`. If your machine has a different IP address, you should replace the IP address with your host-only adapter's IP address.

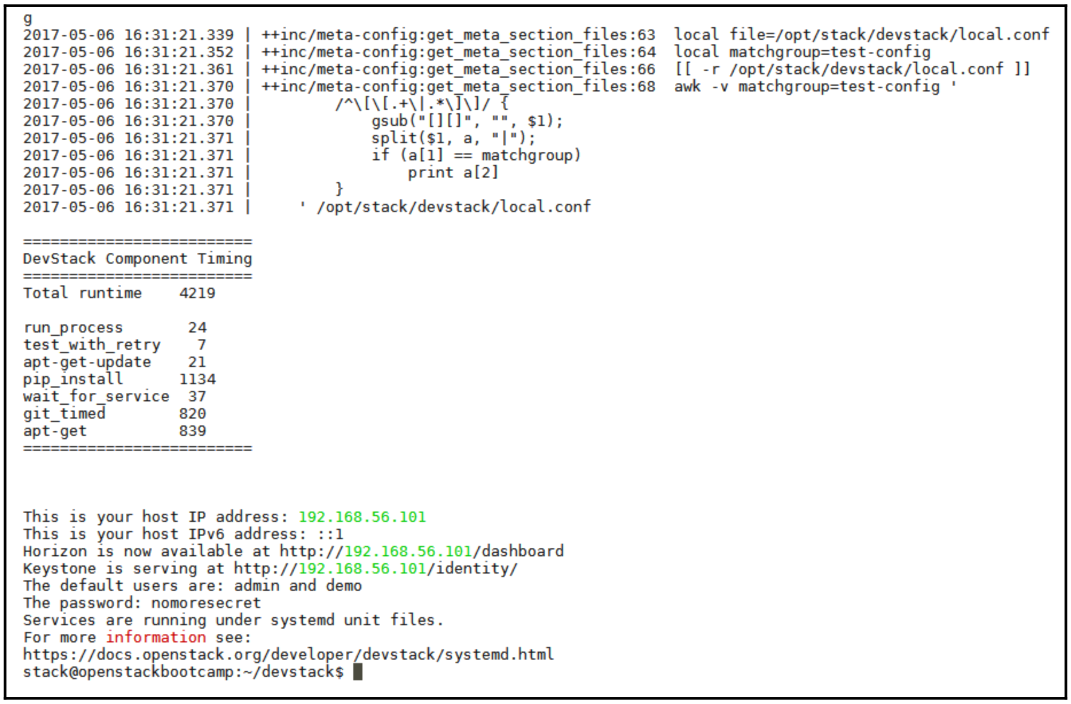
Before proceeding further, it would be useful if you go through your `local.conf` environment file. For your convenience, you could also change the hard-coded admin and database passwords in the `local.conf` file.

2. Let's begin the installation. Make sure you are running the following command inside the `devstack` folder:

   ```
   $ ./stack.sh
   ```

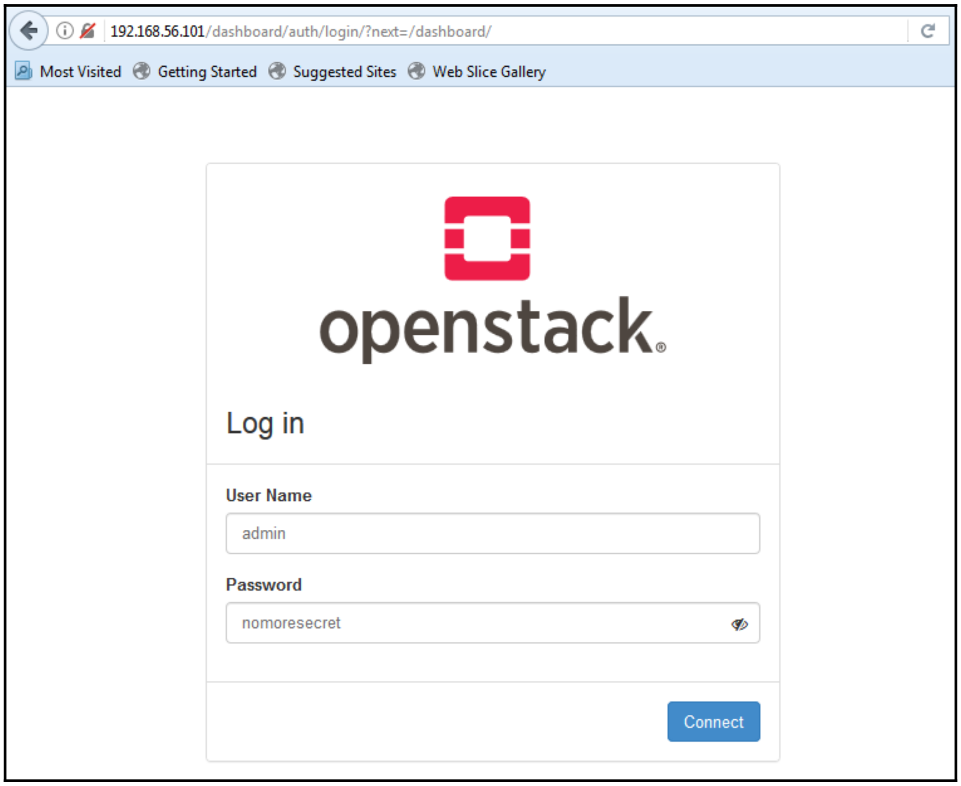
   At this stage, the actual installation of DevStack will begin and will take approximately 20-30 minutes to complete depending on your internet speed and the hardware that you are using.

3. Eureka! On successful installation, you will see an output similar to the following figure:



```
g
2017-05-06 16:31:21.339 | ++inc/meta-config:get_meta_section_files:63  local file=/opt/stack/devstack/local.conf
2017-05-06 16:31:21.352 | ++inc/meta-config:get_meta_section_files:64  local matchgroup=test-config
2017-05-06 16:31:21.361 | ++inc/meta-config:get_meta_section_files:66  [[ -r /opt/stack/devstack/local.conf ]]
2017-05-06 16:31:21.370 | ++inc/meta-config:get_meta_section_files:68  awk -v matchgroup=test-config '
2017-05-06 16:31:21.370 |         /^\[\[.+\|.*\]\]/ {
2017-05-06 16:31:21.370 |             gsub("[][]", "", $1);
2017-05-06 16:31:21.371 |             split($1, a, "|");
2017-05-06 16:31:21.371 |             if (a[1] == matchgroup)
2017-05-06 16:31:21.371 |                 print a[2]
2017-05-06 16:31:21.371 |         }
2017-05-06 16:31:21.371 |     ' /opt/stack/devstack/local.conf

=========================
DevStack Component Timing
=========================
Total runtime    4219

run_process       24
test_with_retry    7
apt-get-update    21
pip_install      1134
wait_for_service  37
git_timed        820
apt-get          839
=========================



This is your host IP address: 192.168.56.101
This is your host IPv6 address: ::1
Horizon is now available at http://192.168.56.101/dashboard
Keystone is serving at http://192.168.56.101/identity/
The default users are: admin and demo
The password: nomoresecret
Services are running under systemd unit files.
For more information see:
https://docs.openstack.org/developer/devstack/systemd.html
stack@openstackbootcamp:~/devstack$
```

4. Kudos! Now you have a working OpenStack cloud. To get a hands-on experience with it, you can access your OpenStack cloud in two ways:
    - **Openstack web UI (Horizon)**: Access the Openstack Horizon via your host machine's browser with the URL and password displayed in your DevStack installation terminal output. In my case, I access my OpenStack UI at `http://192.168.56.101/dashboard/` and the password for the `admin` user is `nomoresecret`:



192.168.56.101/dashboard/auth/login/?next=/dashboard/

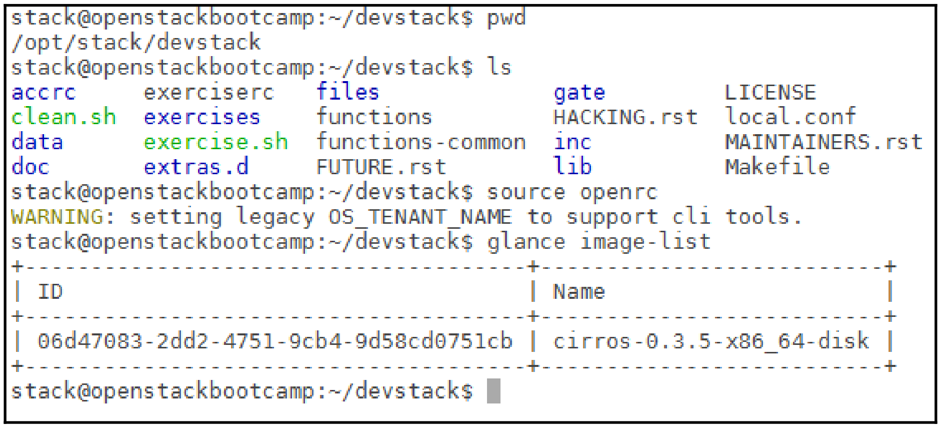
Log in

User Name: admin

Password: nomoresecret

The host-only network allows all the traffic to flow between the virtual machine and the host machine (a laptop in my case). To access your OpenStack services from computers other than your host computer, you can optionally enable the third network adaptor with the bridged adapter network type option in the VirtualBox settings.

- **Openstack command-line tool**: Optionally, you can access the `source openrc` file in your terminal and use the OpenStack command lines to manage your OpenStack cloud. The `openrc` file will be created inside the `devstack` folder after the DevStack installation:



```
stack@openstackbootcamp:~/devstack$ pwd
/opt/stack/devstack
stack@openstackbootcamp:~/devstack$ ls
accrc     exerciserc   files             gate         LICENSE
clean.sh  exercises    functions         HACKING.rst  local.conf
data      exercise.sh  functions-common  inc          MAINTAINERS.rst
doc       extras.d     FUTURE.rst        lib          Makefile
stack@openstackbootcamp:~/devstack$ source openrc
WARNING: setting legacy OS_TENANT_NAME to support cli tools.
stack@openstackbootcamp:~/devstack$ glance image-list
+--------------------------------------+--------------------------+
| ID                                   | Name                     |
+--------------------------------------+--------------------------+
| 06d47083-2dd2-4751-9cb4-9d58cd0751cb | cirros-0.3.5-x86_64-disk |
+--------------------------------------+--------------------------+
stack@openstackbootcamp:~/devstack$
```

5. You should be aware of the following useful DevStack commands that could help you save your valuable time:
   - Reconnect to DevStack by rerunning the installation whenever you reboot your virtual machine using the following command:

```
$ ./stack.sh
```

   - To stop all the OpenStack services started by `./stack.sh`, use the following command:

```
$ ./unstack.sh
```

   - To remove all the DevStack data files from your VM, use the following command:

```
$ ./clean.sh
```

OpenStack community and getting help

The essence of the open source ethos and the community-driven development approach has established OpenStack as one of the fastest-growing and active open source communities in the world. So, whenever you get stuck with any issues in OpenStack, you can make use of this big, global, open-source community.

You can connect to the OpenStack community in order to get assistance in the following ways:

- Visiting the questions and answers forum (`https://ask.openstack.org`)
- Visiting the wiki (`https://wiki.openstack.org`)
- Participate in chats on IRC `#openstack` at `http://webchat.freenode.net/?channels=openstack,openstack-101`
- Join the general mailing list (`http://lists.openstack.org`)
- Join the local user group to attend local events (`https://groups.openstack.org/groups`)

I would strongly recommend that all OpenStack beginners register with the `https://ask.openstack.org/` forum to get started with OpenStack.

Summary

On day 1, we have successfully built our bootcamp using the DevStack scripted installation. Now we have our OpenStack cloud lab up and running to get some hands-on experience in OpenStack. Like I mentioned before, we have various options that are available for us to build the OpenStack development environment in one go. In this book, I have chosen to go with DevStack to build the OpenStack play camp. Alternatively, you could try some of the different options that are available to make your OpenStack play field. Irrespective of your choice, the final output will be the same OpenStack.

We also learned about the design principle of OpenStack and the distributions that are available.

On day 2, we will walk through the overview of OpenStack's core components in detail and make some real-world comparisons.

2
Day 2 - Know Your Battalion

In Day 2, we will walk through the overview of core components in OpenStack. For better understanding, I will relate the OpenStack components to the real-world examples.

OpenStack components are usually compared with the components in **Amazon Web Services** (**AWS**), so OpenStack beginners can understand the design by comparing each component of OpenStack with AWS projects. In this chapter, I will mainly focus on giving the readers real-time examples. I firmly believe that the real-world examples will help the readers to understand even better.

- Core components:
    - KeyStone (identity service)
    - Glance (image service)
    - Nova (computing service)
    - Neutron (networking service)
- Optional components:
    - Cinder (block storage service)
    - Swift (object storage service)

Core components of OpenStack

The core projects are necessary components in Openstack for providing the Infrastructure-as-a-Service. In practise, without any one of the above listed core components, Openstack cannot provision the virtual machine for the user.

KeyStone (identity service)

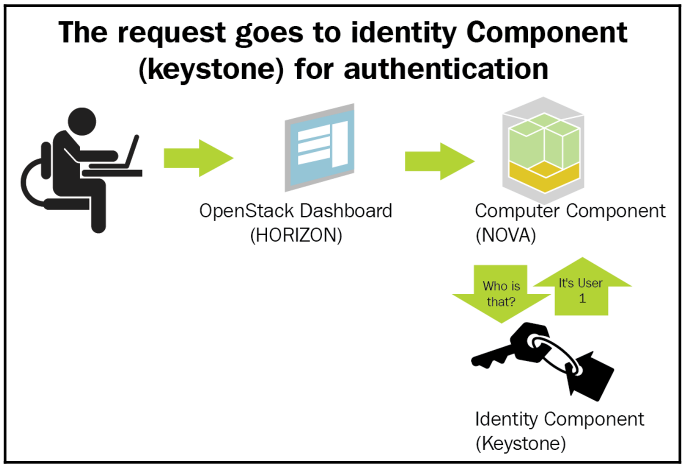
Let's imagine Mr. Neo has joined a company (Say **Cloudenablers**). On his first day before entering his cabin, he is advised to collect his access card from the building access control team.

The building access control team will provide an access card to Mr. Neo. This access card will have a unique ID for each user. The card should be used as an identity (authentication) card and also has authorization information based on the role mapped for the user profile. For example, Mr. Neo could access the server room if his profile is assigned with the data center admin role, or else his access to the server room will be denied.

The building access control team will take care of mapping the selected role to each and every employee based on his designation. For example, the role *data center admin* will not be assigned to employees with a software engineer and HR designation to restrict them from entering the server rooms. The CEO of the company is assigned the *full access role,* which means he could enter any room and access any resources in the company.

Are you wondering how the building access control team is related to the KeyStone service?

Yeah! The building access control team and the KeyStone service have many similarities, and the only difference is that we see the building access control team works in the real world and KeyStone works as a service running in OpenStack to take care of similar actions, such as authentication and authorization of the user in OpenStack:


The request goes to identity Component (keystone) for authentication
OpenStack Dashboard (HORIZON)
Computer Component (NOVA)
Who is that?
It's User 1
Identity Component (Keystone)

KeyStone provides Identity, Token, Catalog, and Policy services for use specifically by projects in the OpenStack family.

Here, we can compare.

Mr. Neo, the end user of OpenStack (mapped with the `_member_` role) is accessing the OpenStack dashboard. We can liken the building access control team to a KeyStone service. The KeyStone service will take care of providing the unique token ID for each user and also assign the selected roles (admin or member) to the user. The provided token ID will be used for authenticating, and the authorization of end users accessing the dashboard based on the role mapped. Like a CEO, the user with an admin role in OpenStack will have full access to all of the services in OpenStack. For example, the user having an admin role could access the flavor list and see all of the VMs running across the tenants, but the user with a `_member_` role could not access the flavor list information and the external network details in OpenStack.

The company Cloudenablers could relate to OpenStack. The building access control team is one among many teams in the company and selectively handles the identity system. Similarly, the KeyStone service is one among many services in OpenStack and explicitly takes care of the identity management system.

Keystone Recap

- Cloudenablers - OpenStack
- Building access control team - KeyStone Service
- Access card - Token ID
- CEO role - Admin role
- Mr. Neo - Member, role User
- KeyStone Service - AWS IAM (**Identity and Access Management**) Service

Nova (computing service)

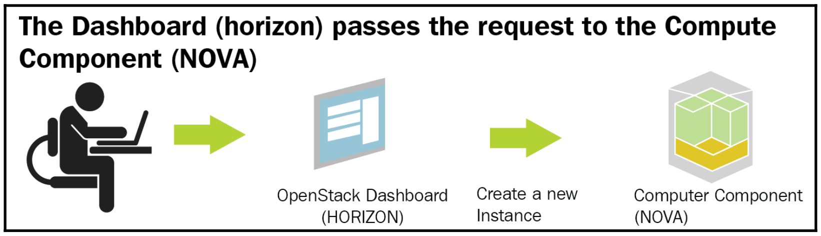
Mr. Neo, using his access card (Unique Token ID), has successfully entered into his workplace. Now it is time for Mr. Neo to get a desktop workstation for his work. So, he raises the request for a new desktop machine and requests the required configuration (such as Ubuntu OS/8GB RAM/500GB HDD) from the *IT hardware allocation team*.

As soon as the request is received, the IT hardware allocation team forward Mr. Neo's unique access ID to the access management system and check whether Mr. Neo is authorized to access the desktop workstation.

Upon successful verification, the IT hardware allocation team will approve his desktop workstation request for the required configuration (Ubuntu OS/8GB RAM/500GB HDD) and forward the request to the desktop assembly unit subdivision.

The desktop assembly team will assemble the new desktop machine with the requested configuration of 8GB RAM and a 500GB hard disk.

Once the desktop machine is assembled, the IT hardware allocation team will seek the image management team for an Ubuntu OS image. We will talk about the image management team in detail while explaining the glance service:


The Dashboard (horizon) passes the request to the Compute Component (NOVA)
OpenStack Dashboard (HORIZON)
Create a new Instance
Computer Component (NOVA)

Nova is an OpenStack project designed to provide massively scalable, on-demand, self-service access to computing resources.

Now, you may have some idea of what the Nova service will do in OpenStack. Here, we could liken the IT hardware allocation team to the Nova-API service.

In OpenStack, Nova, the computing service, will take care of creating the new Virtual Machines with the user requested configuration. When the user requests a new VM, the Nova-API service will contact the KeyStone service to validate the user token ID and confirm that the user is authorized to access the Nova service. On successful verification, the Nova-API service will ask the Nova-Compute sub-demon service to provision a new VM with the user requested configuration.

Like an assembly unit, Nova also has its own sub-demon services such as Nova-Scheduler, Nova-Compute, Nova-API, and Nova-Console auth.

We will see all of these services in detail and their interconnection in `Chapter 3`, *Day 3 - Field Sketch* .

Nova Recap

- IT hardware allocation team - Nova-API Service
- Desktop assembly team - Nova-Compute service
- Image management team - Glance Image service
- Desk allocation team - Nova-Scheduler
- OpenStack Nova Service - AWS **EC2** (**Elastic Compute Cloud**) Service

Glance (image service)

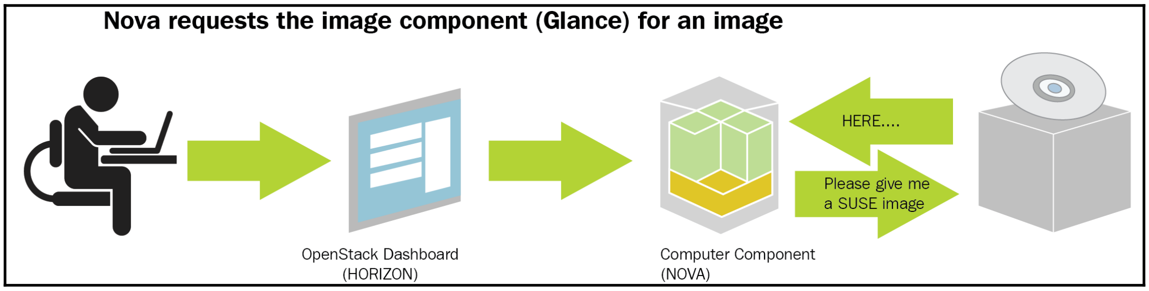
The assembled bare metal is now waiting for a Ubuntu operating system to be installed.

In our case, Mr. Neo has requested a Ubuntu OS, but another user may request any other OS (say, Windows 7) based on their requirements. It is hard for the IT hardware allocation team to manage all operating system ISOs and software bundles. So, the new team, called the image management team, has been created for managing the OS packages for the users.

The image management team will act as an ISO image repository for the IT hardware allocation team and will provide the selected OS for the IT hardware allocation team to install the operating system in the assembled bare metal.

The purpose of the image management team is to maintain the repository for the necessary operating system and to keep the metadata records for each OS bundle. Metadata is the register that maintains the minimal hardware requirements to run the selected OS. For example, the minimum hardware requirements for the Ubuntu 14.04 LTS server OS is 1GB RAM/8GB HDD and similarly 4GB RAM/40GB HDD for Windows 7 OS. In any case, if the minimal resource requirement for any OS is not met, then the access to that selected OS will be repudiated for the IT hardware allocation team, and the same will be reported to the user.

In our case, when the request for Ubuntu OS is received from the IT hardware allocation team, the image management team will contact the access management system to check whether Mr. Neo is authorized to access the Ubuntu OS on his workstation. Once the verification is passed, the image management team will check the minimal hardware requirements for the Ubuntu OS and provide the Ubuntu OS bundle to the IT hardware allocation team to proceed with the OS installation:


Nova requests the image component (Glance) for an image
OpenStack Dashboard (HORIZON)
Computer Component (NOVA)
HERE....
Please give me a SUSE image

Glance image services include discovering, registering, and retrieving **virtual machine** (**VM**) images.

Similarly, the glance service serves as an image management team for the Nova service in OpenStack. The job of the glance service is to manage the OS image repository for the user. In simple words, similar to a DVD pouch in our home that contains different operating system images in an organized way to access whenever we install a new OS on our machine.

Whenever the Nova service requests the OS image bundle, the Glance image service will first contact the KeyStone service with the user's token ID to check whether the user is authorized to access the image service or not. Once all of the verification including minimal hardware requirements are passed, the glance service will provide the requested OS image bundle to the Nova service to begin the OS installation.

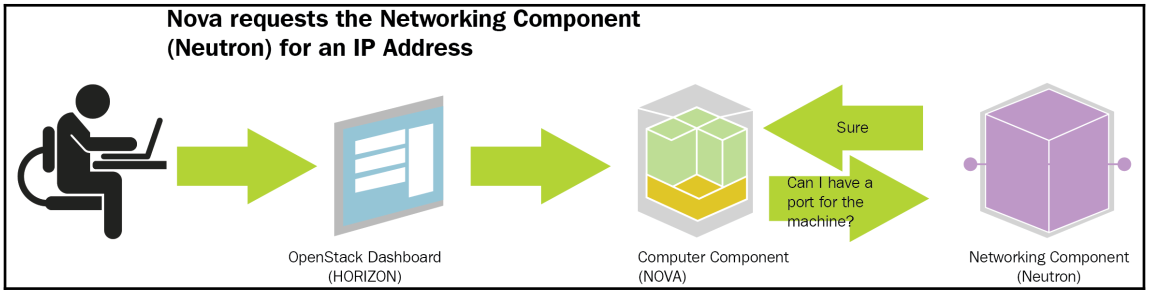
Image Recap

- Image management team - Glance Image Service
- IT hardware allocation team - Nova Service
- Access management system - KeyStone Service
- Glance Image Service - AWS AMI (**Amazon Machine Images**) Service

Neutron (networking service)

As we all know, any workstation in the office environment with no network connectivity is useless. The network configuration and management are not simple tasks to achieve. So, the networking team is dedicated to taking care of all networking related activities starting from NIC allocation to DHCP IP assignment. Before beginning the OS installation process, the IT hardware allocation team will request the Networking Team to map Mr. Neo's desktop workstation to the network dedicated to his project.

When the request from the IT allocation team is received, the networking team will check Mr. Neo's token ID with the access management system to verify whether Mr. Neo is authorized to access his project network or not. Once the verification is accepted, the networking team will allot a NIC for Mr. Neo's workstation and bond the NIC with his project's private network:


Nova requests the Networking Component (Neutron) for an IP Address
Sure
Can I have a port for the machine?
OpenStack Dashboard (HORIZON)
Computer Component (NOVA)
Networking Component (Neutron)

OpenStack Neutron is an SDN networking project focused on delivering **networking-as-a-service** (**NaaS**) in virtual compute environments.

Similarly, the neutron service will act as a networking team in OpenStack and manage all of the network related activities. When the request for a vNIC is received, the neutron service will check the KeyStone service to confirm that the user is authorized to access the selected network. Then, it will allot the vNIC to the new Virtual Machine and plot the vNIC to the selected private network.

The neutron functionaries are not limited to providing a NIC and IP pool for the workstation. However, the neutron functionality covers everything involved in advanced networking.

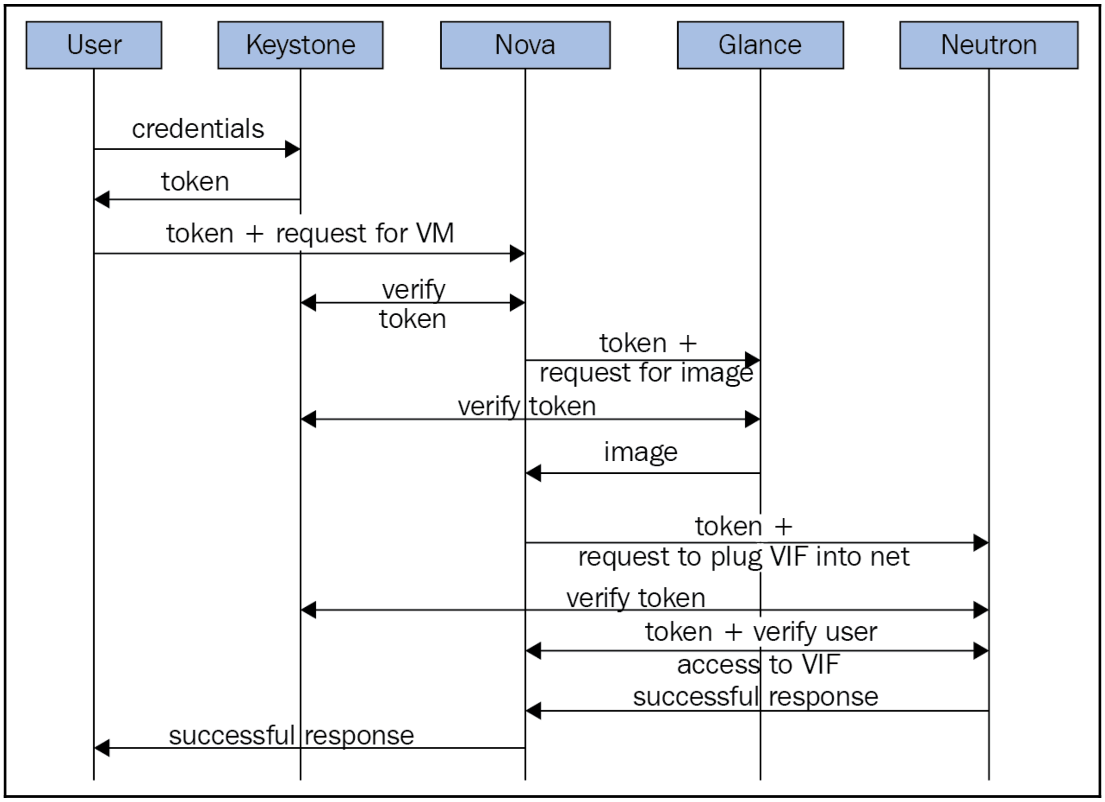
We will walk through OpenStack Neutron and its extended features in detail in `Chapter 5`, *Day 5 - Networking Strategy*.

Neutron Recap

- Networking Team - Neutron Service
- Access management system - KeyStone Service
- IT hardware allocation team - Nova Service
- Neutron networking - AWS **VPC*** (**Virtual Private Cloud**) Service

OpenStack service interaction layout

The following bird's-eye outline diagram shows the core services of OpenStack interacting with one another in bringing up the new virtual machine to the user:



At this stage, Mr. Neo has received his Ubuntu workstation with 8GB RAM, 500GB HDD and an effective network connectivity. From the above flow, starting from Mr. Neo entering the office to receiving his workstation, you can understand how all four teams worked together in bringing up the workplace. By now, it is reasonable to say that Mr. Neo would not receive his workstation if any one of the four teams is not working.

Similarly, the services mentioned above are the necessary facilities in OpenStack for providing the **Infrastructure-as-a-Service** (**IAAS**) platform for the end users. Moreover, for its necessity, these four services are known as core services of OpenStack.

Optional components

The optional components are the projects that will add an extended feature to the Infrastructure-as-a-Service platform in OpenStack. This component is categorized under additional projects of OpenStack, mainly because the project is necessarily not required in the OpenStack to provide a basic IAAS platform.

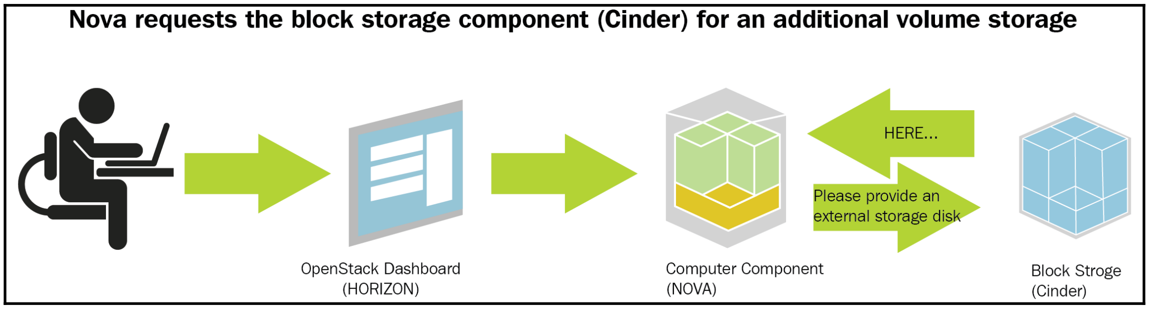
Cinder (block storage service)

Let's suppose it has been a year since Mr. Neo started using his workstation. His daily task saves loads and loads of data onto his hard disk. Mr. Neo noticed that he would run out of his 500 GB internal hard drive space in the next few days. So, he decided to demand a new external hard drive (portable HDD) from the storage team and raised the new request for an external hard drive of 1 TB in size.

When the storage team received the request, it will verify the token ID of Mr. Neo with the access management team to confirm that the user is authorized to receive an additional storage.

Once the authorization is verified, the storage team will provide him with an external 1 TB hard disk.

We all know the advantage of using the portable hard drive. Mr. Neo has the flexibility to connect and mount the external HDD permanently and use it as a secondary internal hard drive for his workstation or optionally, he could also mount the external HDD temporarily to backup all necessary files and detach the external HDD from his workstation:


Nova requests the block storage component (Cinder) for an additional volume storage
HERE...
Please provide an external storage disk
OpenStack Dashboard (HORIZON)
Computer Component (NOVA)
Block Stroge (Cinder)

Cinder virtualizes the management of block storage devices and provides end users with a self-service API for requesting and consuming those resources without requiring any knowledge of where their storage is deployed or on what type of device.

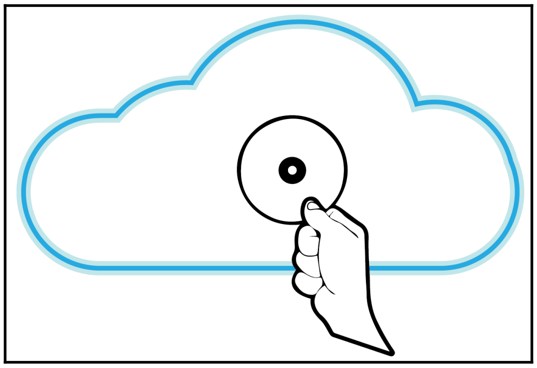
Similarly, the cinder service will do the storage team's job in OpenStack. The Cinder block storage service will take care of providing the additional disk volume to the user. Once the new disk volume is allocated to the user tenant (project), the user has the flexibility to map (attach) the volume storage to any VMs between the same project. A Cinder volume is very similar to an external HDD; you cannot attach the single cinder volume to two or more virtual machines at the same time, however we could connect the single cinder volume to any virtual machine, one at a time.

Cinder Recap

- Storage Team - Cinder block storage service
- Access management team - KeyStone service
- Cinder Volume Service - AWS **EBS** (**Elastic Block Storage**) service

Swift (object storage service)

Mr. Neo still has some important files that need to be retrieved even if his workstation or external HDD got corrupted. So, he decided to use Google Drive/Dropbox cloud storage. He uploaded all his extensive archives to Google Drive to access them anytime, even if he has no access to his own workstation in case of system repair:



TIP

The OpenStack Swift offers cloud storage software so that you can store and retrieve lots of data with a simple API.

The Swift object storage service is very similar to the Google Drive kind of Cloud storage system. For the end user, the Swift service will act as a simple object storage system that can store any file types of any size. However, behind the screen, it has an elaborate architecture for saving each and every file in its replica and retrieve back mechanism.

We will walk through the OpenStack Swift object storage service in detail in the forthcoming `Chapter 10`, *Day 10 - Side Arms of OpenStack*.

Swift Recap

- Google Drive or Dropbox Cloud Storage - Swift Object Storage
- OpenStack Swift Object Storage Service - AWS S3 (**Simple Storage Service**)

Summary

In Day 2, we learned the overview of each OpenStack core and secondary (KeyStone/Glance/Nova/Neutron/Cinder/Swift) component in detail and also compared the OpenStack components with real-world scenarios.

Apart from the previous discussed core and optional projects, at present, the Ocata release has 30+ projects officially added in OpenStack umbrella, and most of the projects are not yet production-ready. We will discuss the most commonly known projects from the OpenStack additional services list in the `Chapter 10`, *Day 10 - Side Arms of OpenStack*.

In the next chapter, we will walk through the architecture design of OpenStack and how the core components of OpenStack are interconnected.

3
Day 3 - Field Sketch

In the previous chapter, I compared the OpenStack services with real-world examples to understand the primary job of each OpenStack component. However, in the last chapter, I covered only the high-level view of each OpenStack service. In this chapter, I will walk you through all the sub-components of each OpenStack service and their interconnection in detail--in short, the OpenStack architecture.

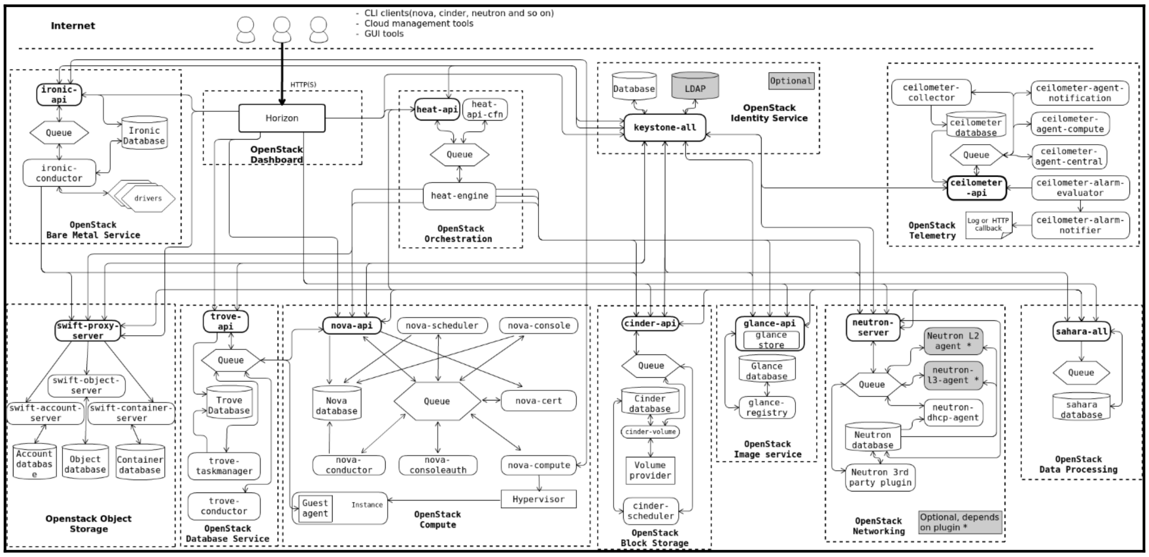
Before we begin, let's recap our real-world examples from the previous chapter to understand the complex OpenStack architecture with ease:

- **Access control team**: KeyStone service
- **IT hardware allocation team**: Nova service
- **Desktop assembly team**: `nova-compute` service
- **Desk allocation team**: `nova-scheduler`
- **Image management team**: Glance image service
- **Networking team**: Neutron service
- **Storage team**: Cinder block storage service
- **Google Drive or Dropbox**: Swift object storage

OpenStack - logical architecture

To design, deploy, and configure the OpenStack cloud, one must understand the logical structure of OpenStack. The OpenStack logical diagram explains all the *HOW* questions in the OpenStack:

- How do all OpenStack services authenticate through a common identity service?
- How do individual services interact with each other through APIs?
- How are OpenStack services composed?
- How is an **Advanced Message Queuing Protocol** (**AMQP**) message broker used for communication between the processes of one service?
- How can users access OpenStack?

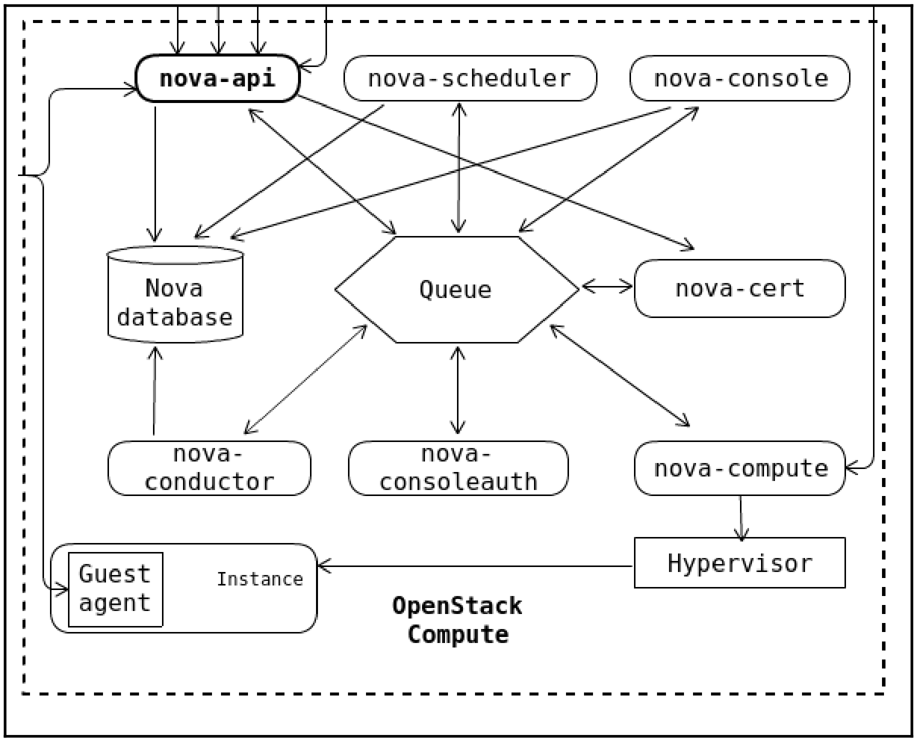


The preceding diagram depicts the most common architecture for an OpenStack cloud. As you can see from the OpenStack logical architecture diagram, each OpenStack service is composed of several daemons and processes.

Compute service - Nova

The compute service consists of various services and daemons including the `nova-api` service, which accepts the API requests and forwards them to the other components of the service.

The following diagram shows how various processes and daemons work together to form the compute service (Nova) and interlink between them:



OpenStack compute consists of the following services and daemons.

nova-api service

All services in OpenStack have at least one API process that accepts and responds to end user API calls, preprocesses them, and passes the request to the appropriate process within the service.

For the OpenStack compute service, we have the `nova-api` service, which listens and responds to the end user compute API calls. The `nova-api` service takes care of initiating most orchestration activities, such as provisioning new virtual machines.

nova-api-metadata service

The metadata service delivers the instance-specific data to the virtual machine instances. The instance-specific data includes `hostname`, `instance-id`, `ssh-keys`, and so on. The virtual machine accesses the metadata service via the special IP address at `http://169.254.169.254`.

nova-compute service

Underneath, the entire lifecycle of the virtual machine is managed by the hypervisors. Whenever the end user submits the instance creation API call to the `nova-api` service, the `nova-api` service processes it and passes the request to the `nova-compute` service. The `nova-compute` service processes the `nova-api` call for new instance creation and triggers the appropriate API request for virtual machine creation in a way that the underlying hypervisor can understand.

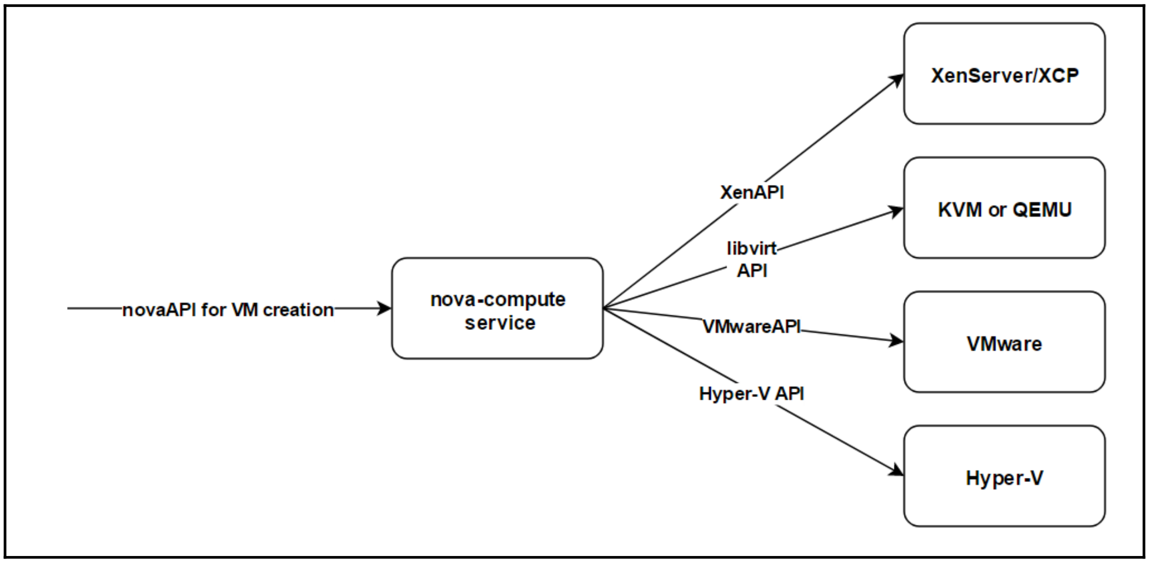
For example, if we choose to use the KVM hypervisor in the OpenStack setup, when the end user submits the virtual machine creation request via the OpenStack dashboard, the `nova-api` calls will get sent to the `nova-api` service. The `nova-api` service will pass the APIs for instance creation to the `nova-compute` service. The `nova-compute` service knows what API the underlying KVM hypervisor will support. Now, pointing to the underlying KVM hypervisor, the `nova-compute` will trigger the `libvirt-api` for virtual machine creation. Then, the KVM hypervisor processes the `libvirt-api` request and creates a new virtual machine.

OpenStack has the flexibility to use multi-hypervisor environments in the same setup, that is, we could configure different hypervisors like KVM and VMware in the same OpenStack setup. The `nova-compute` service will take care of triggering the suitable APIs for the hypervisors to manage the virtual machine lifecycle.

To know more about the multi-hypervisor OpenStack environment, visit the following link:
`http://www.hellovinoth.com/multi-hypervisor-openstack-integrating-vmware-vcenter-and-kvm-hypervisors-with-openstack/`

nova-scheduler service

When we have more than one compute node in your OpenStack environment, the nova-scheduler service will take care of determining where the new virtual machine will provision. Based on the various resource filters, such as RAM/CPU/Disk/Availability Zone, the nova-scheduler will filter the suitable compute host for the new instance:



nova-conductor module

The nova-compute service running on the compute host has no direct access to the database because if one of your compute nodes is compromised, then the attacker has (almost) full access to the database. With the nova-conductor daemon, the compromised node cannot access the database directly, and all the communication can only go through the nova-conductor daemon. So, the compromised node is now limited to the extent that the conductor APIs allow it.

The nova-conductor module should not be deployed on any compute nodes, or else the purpose of removing direct database access for the nova-compute will become invalid.

nova-consoleauth daemon

The `nova-consoleauth` daemon takes care of authorizing the tokens for the end users, to access a remote console of the guest virtual machines provided by the following control proxies:

- The `nova-novncproxy` daemon provides a proxy for accessing running instances through a VNC connection.
- The `nova-spicehtml5proxy` daemon provides a proxy through a SPICE connection

nova-cert module

Used to generate X509 certificates for euca-bundle-image, and only needed for the EC2 API.

The queue (AMQP message broker)

Usually, the AMQP message queue is implemented with RabbitMQ or ZeroMQ. In OpenStack, an AMQP message broker is used for all communication between the processes and daemons of one service. However, the communication between the two different services in OpenStack uses service endpoints.

For example, the `nova-api` and `nova-scheduler` will communicate through the AMQP message broker. However, the communication between the `nova-api` service and `cinder-api` service will carry through the service endpoints.

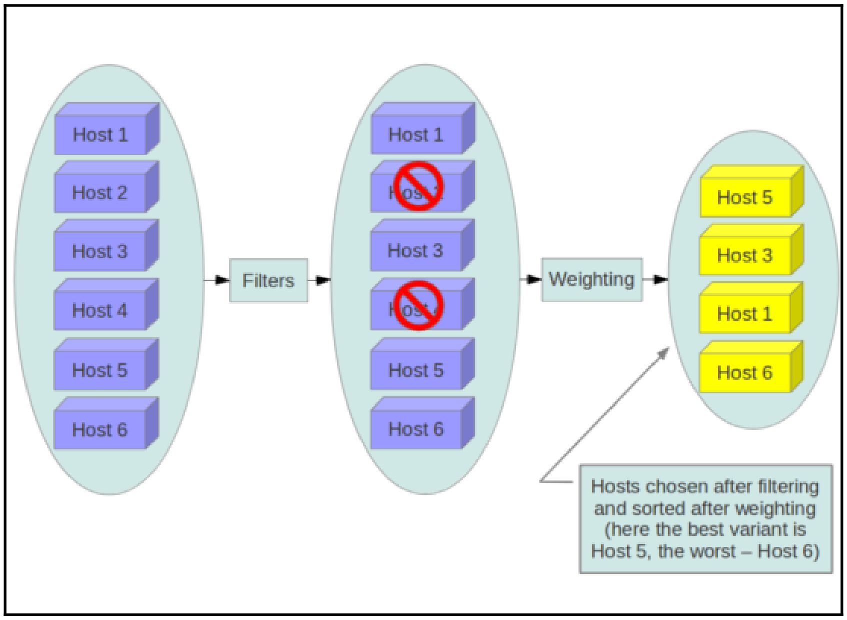
Database

Most of the OpenStack services use an SQL database to store the build-time, and run-time states for a cloud infrastructure, such as instance status, networks, projects, and the list goes on. In short, we could say the database is the brain of OpenStack.

The most tested and preferable databases to use in OpenStack are MySQL, MariaDB, and PostgreSQL.

Image service - Glance

The following logical architecture diagram shows how various processes and daemons work together to perform the image service (Glance) in OpenStack, and the interconnection between them:


Host 1
Host 2
Host 3
Host 4
Host 5
Host 6
Filters
Host 1
Host 3
Host 5
Host 6
Weighting
Host 5
Host 3
Host 1
Host 6
Hosts chosen after filtering and sorted after weighting (here the best variant is Host 5, the worst – Host 6)

glance-api service

As I said before, all services in OpenStack have at least one API process, which accepts and responds to the user API calls and passes the request to the appropriate process within the service.

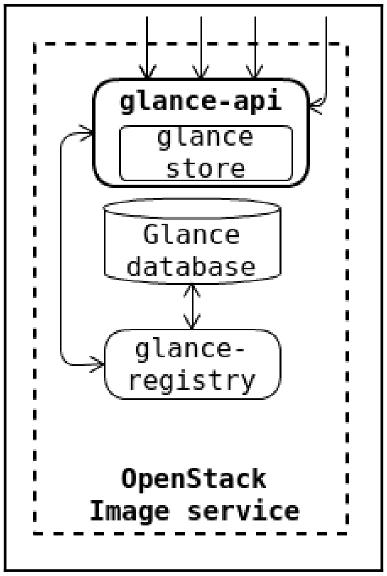
For the OpenStack image service, the `glance-api` service processes the image API calls for image discovery, retrieval, and storage.

glance-registry service

As its name reads, the registry service takes care of storing, processing, and retrieving metadata about the images. Notably, the registry service only deals with the image metadata, not the image itself. Metadata includes image information such as size and type.

Identity service - KeyStone

The following logical architecture diagram shows how the identity service in OpenStack is designed to process the authentication and the authorization:



The exception to all other OpenStack services, the identity service has no dedicated API service to process, and listens to the API calls. However, the actual work is done by distinct processes.

Server (keystone-all)

A centralized server processes the authentication and authorization requests using a RESTful interface.

Drivers (optional)

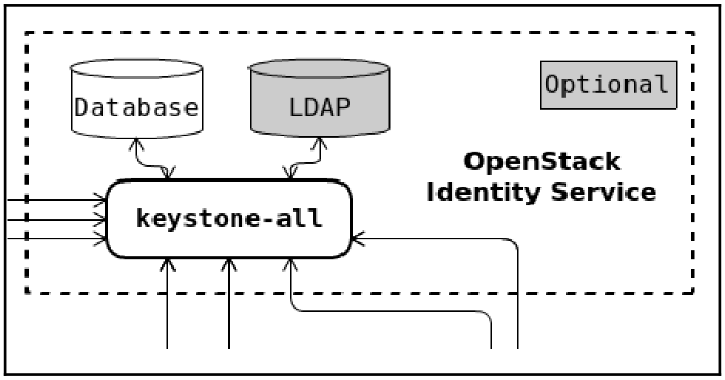
With the help of the backend drivers, the centralized Keystone server can be integrated with the selected service backend, such as LDAP servers, Active Directory, and SQL databases. They can be used for accessing the identity information from the common repositories external to OpenStack.

Networking service - neutron

The logical architecture diagram in *Neutron server* section shows how various processes and daemons work together to perform the networking service (neutron) in OpenStack and the interrelation between them.

Neutron server

The neutron server accepts and routes API requests to the suitable OpenStack neutron plug-in to process the request. The neutron-server service runs on the network node (in most cases, the network node is combined with the controller node). We will study the OpenStack deployment topology later in this chapter to know more about controller/network/compute nodes:



plugin agent (neutron-*-agent)

While the neutron server acts as the centralized controller running on the network node, the `neutron-*-agent` runs on the compute node to manage the local virtual switch (vswitch) configuration. Moreover, the agents receive messages and instructions from the Neutron server (via plugins or directly) on the AMQP message bus and then the actual networking related commands and configuration are executed by the `neutron-*-agent` on the compute and network node.

I have listed a few of the most commonly used neutron agents here:

- `neutron-lbaas-agent`: LBaaS agent
- `neutron-linuxbridge-agent`: Linuxbridge agent
- `neutron-macvtap-agent`: Macvtap L2 Agent
- `neutron-metadata-agent`: Metadata agent
- `neutron-metering-agent`: Metering agent
- `neutron-mlnx-agent`: Mellanox plugin agent
- `neutron-openvswitch-agent`: Open vSwitch plugin agent
- `neutron-sriov-agent`: SR-IOV agent
- `neutron-vpn-agent`: VPN agent

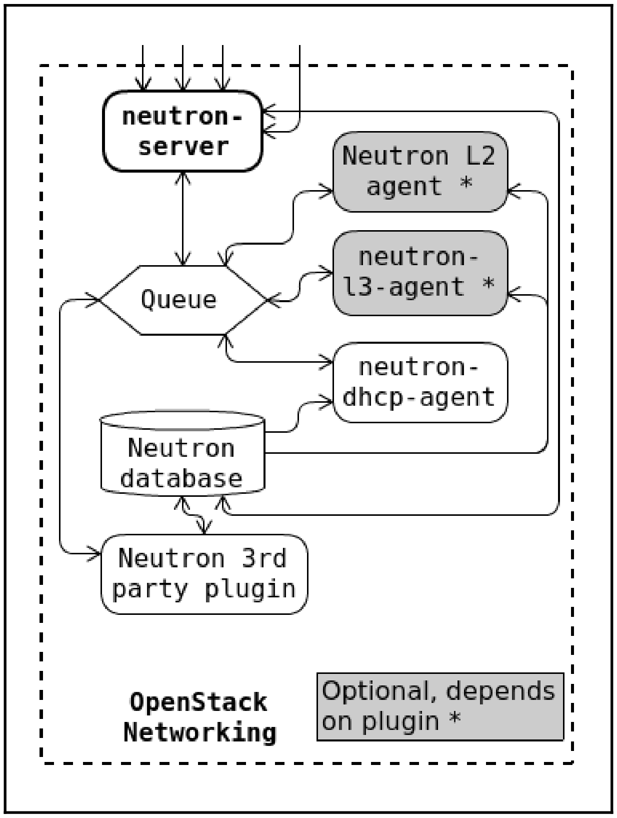
We will walk through all the agents, plug-ins, and the neutron architecture details in the upcoming `Chapter 5`, *Day 5 - Networking Strategy*.

DHCP agent (neutron-dhcp-agent)

OpenStack networking is very similar to networking in the real world. The Virtual machines require **Layer 2** (**L2**) network connectivity minimally. The `neutron-dhcp-agent` acts as a DHCP server to the tenant network, and takes care of providing DHCP IP to the instance.

L3 agent (neutron-l3-agent)

Like I mentioned previously, in the real world, you need internet/external network access for your workstation. For the external network connectivity, in the real world, we use a router (L3 functionality). Similarly, in OpenStack networking, we use the `neutron-l3-agent` to provide L3/NAT forwarding functionality to the virtual machines:

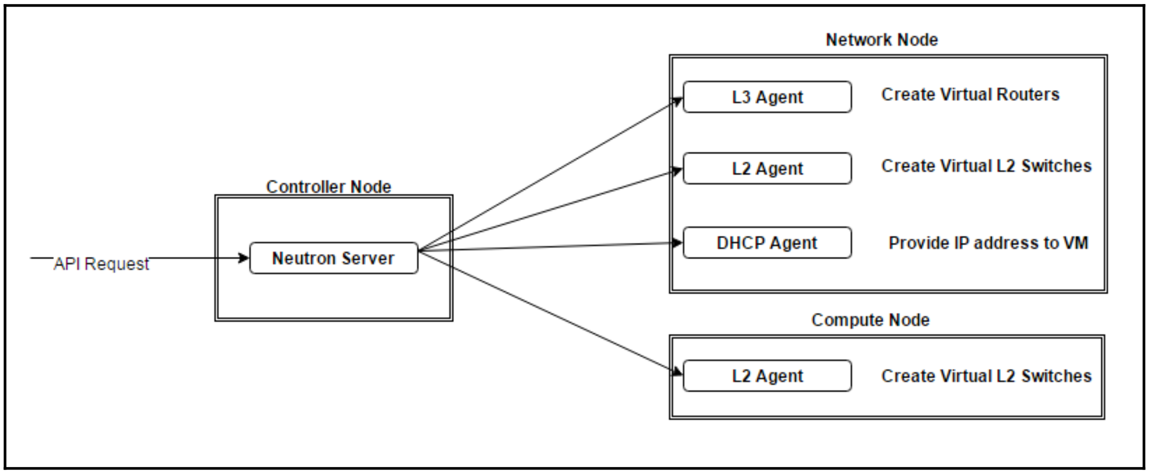


Key points to remember:

1. The server provides the API, manages the database, and so on.
2. Plug-ins manage agents.
3. Agents provide layer 2/3 connectivity to instances and handle physical-virtual network transition.

Block storage service - Cinder

The logical architecture diagram shows how various processes and daemons work together to perform the block storage service (Cinder) in OpenStack, and the interconnection between them:



cinder-api service

For the block storage service, the `cinder-api` service accepts API requests and routes them to the `cinder-volume` for action.

cinder-scheduler daemon

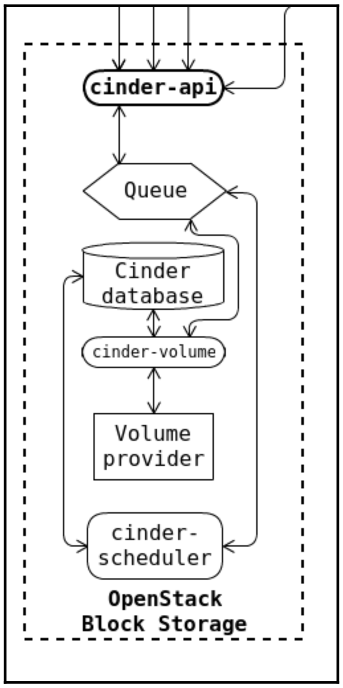
Similar to the `nova-scheduler` for compute service, `cinder-scheduler` plays the same role for the block storage service. The `cinder-scheduler` daemon schedules and routes the requests to the appropriate volume service based upon your filters in the configuration settings, such as storage capacity, availability zones, volume types, and capabilities.

cinder-volume Service

After filtering and choosing the suitable volume service, the `cinder-scheduler` will route the storage request to the `cinder-volume` services to process them. Then, the cinder-volume service interacts directly with the block storage service to provide a required storage space. Notably, the `cinder-volume` service could interact with different storage providers at the same time through the various device drivers available.

cinder-backup daemon

Whenever a request is received for volume backup/restore operations, the `cinder-backup` daemon will interact with the backup targets such as Swift object storage or NFS file store for backing up or restoring the volumes of any type:



Whenever the end user submits the API call to create a new Cinder volume through the Horizon dashboard, the following Cinder component interaction will take place:

1. Client issues request to create volume.
2. The `cinder-api` process validates the request and puts a message onto the AMQP queue for processing.
3. The `cinder-volume` takes the message off of the queue, sends a message to `cinder-scheduler` to determine which backend to provision volume into.
4. `cinder-scheduler` generates a candidate list based on current state and requested volume standard (size, availability zone, volume type).
5. `cinder-volume` iterates through the selected candidate list by invoking the backend driver plugin.
6. The backend driver creates the requested volume through interactions with actual storage.
7. `cinder-volume` process collects necessary volume information and posts a response message to AMQP queue.
8. `cinder-api` responds with information including the status of creation request, volume UUID, and so on, to the client.

Object storage - Swift

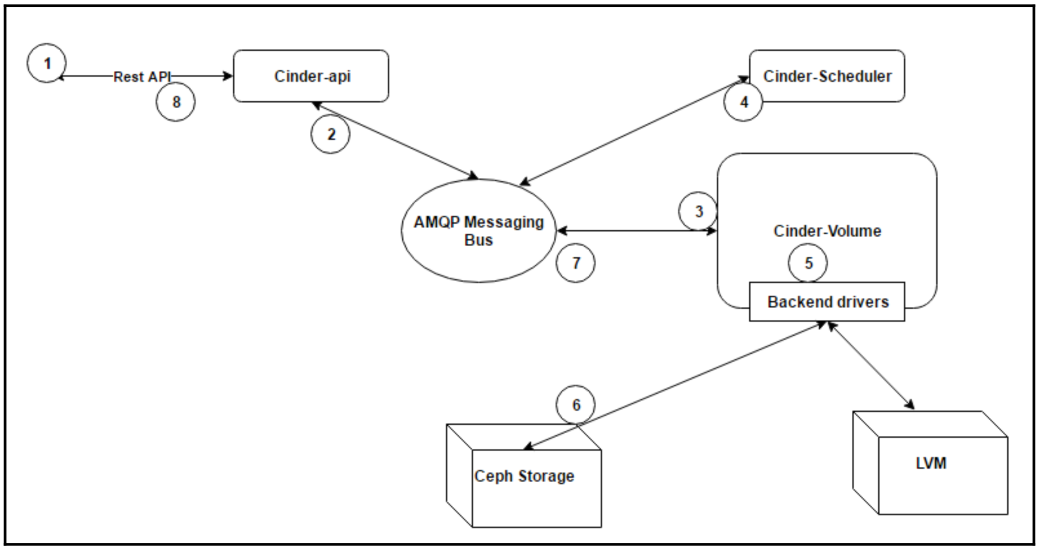
As I mentioned earlier, object storage is very similar to cloud storage, such as Google Drive or Dropbox. As an end user, we can only experience the end user window of Google Drive or Dropbox for storing and retrieving the files. However, the actions carried out by the group of processes and daemons behind the end user screen to save and recover the file has very complex object storage architecture.

The Swift service is quite different from other OpenStack services because we can configure Swift services as standalone services to provide only the object storage services to the end users, without setting IAAS features.

For OpenStack, the Swift service is rated under the additional services, not the core one because the primary purpose of OpenStack is to provide **Infrastructure-as-a-Service** (**IAAS**) to end users. However, Swift is not a mandatory service to bring up the IAAS feature.

As an additional service, the Swift service can be configured with other OpenStack services like Glance and Cinder for storing/retrieving the Glance images and to back up the `cinder-volumes` respectively.

The following logical architecture diagram shows how various processes and daemons work together to bring up the object storage service (Swift) in OpenStack, and the interconnection between the services:



Proxy servers (swift-proxy-server)

For Swift-object storage, we have a proxy server service that accepts OpenStack object storage APIs and raw HTTP requests. The API and HTTP requests include file upload, create folder (containers), and modify the metadata.

Account servers (swift-account-server)

The account server service handles the request to process the metadata information for the individual accounts and the list of the containers mapped for each account.

Container servers (swift-container-server)

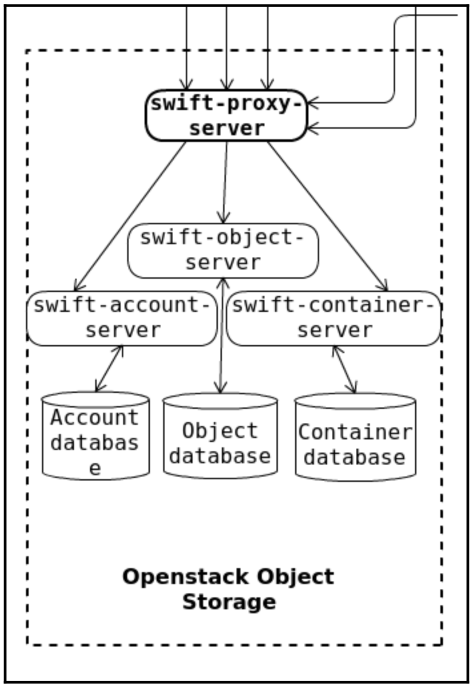
The container server handles the requests about container metadata and the list of objects within each container. The objects stored in the container have no information about the actual storage location, but have information about the particular container where the objects get stored.

Object servers (swift-object-server)

The object server is responsible for managing the actual objects, such as files, on the storage nodes.

Telemetry service - ceilometer

The OpenStack secondary service that handles the metering of all other OpenStack service resource usage is the ceilometer service. With the help of agents, the ceilometer service will collect loads and loads of metering data about all the OpenStack service usage. The collected metrics can be used for billing the resource and can also used for triggering the alarms when the obtained metrics or the event data breaks the defined threshold. For better performance, the ceilometer service is usually configured with a dedicated NoSQL database such as MongoDB, where as the other OpenStack services use SQL databases like MySQL:



Like I mentioned previously, Ceilometer can be used for alarming and billing. Starting from the OpenStack release Newton, the alarm functionality of the ceilometer service is decoupled and added under a new OpenStack project called aodh (Telemetry Alarming service).

ceilometer-agent-compute

The ceilometer agent runs on the compute node to collect the host and the virtual machines resource utilization statistics at regular polling intervals and send the collected statistics to the ceilometer collector to process them.

The following are a few notable meters collected by the `ceilometer-agent-compute`:

Meter name	**Description**
`memory.usage`	RAM used by the instance from the allocated memory
`memory.resident`	RAM used by the instance on the physical machine
`cpu`	CPU time used
`cpu_util`	Average CPU utilization
`disk.read.requests`	Number of read requests
`disk.write.requests`	Number of write requests
`disk.latency`	Average disk latency
`disk.iops`	Average disk IOPS
`disk.device.latency`	Average disk latency per device
`disk.device.iops`	Average disk IOPS per device
`network.incoming.bytes`	Number of incoming bytes
`network.outgoing.bytes`	Number of outgoing bytes

ceilometer-agent-central

Apart from the metrics collected from the compute node, the `ceilometer-agent-central` takes care of collecting the resource utilization statistics at regular polling intervals from the other OpenStack services such as Glance, Cinder, and neutron.

Here, I have listed a few important metrics collected by `ceilometer-agent-central`:

OpenStack image service	
`image.update`	Number of updates on the image
`image.upload`	Number of uploads on the image
`image.delete`	Number of deletes on the image
`image.download`	Image is downloaded
OpenStack block storage	
`volume.size`	Size of the volume
`snapshot.size`	Size of the snapshot
OpenStack object storage	
`storage.objects`	Number of objects
`storage.objects.size`	Total size of stored objects
`storage.objects.containers`	Number of containers
OpenStack identity	
`identity.authenticate.success`	User successfully authenticated
`identity.authenticate.pending`	User pending authentication
`identity.authenticate.failure`	User failed to authenticate
OpenStack networking	
`bandwidth`	Bytes through this l3 metering label
`router.update`	Update requests for this router
`ip.floating.create`	Creation requests for this IP
`ip.floating.update`	Update requests for this IP

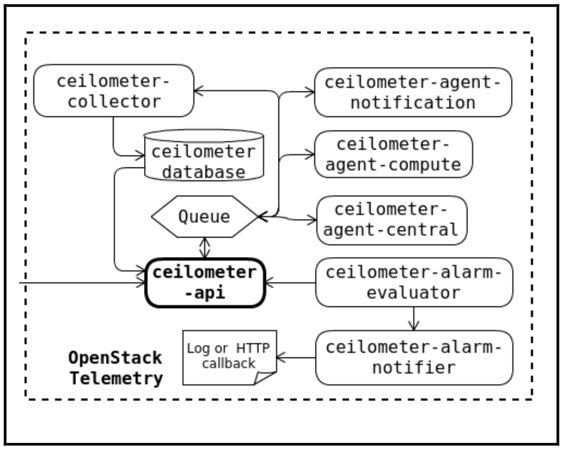
To learn more about the ceilometer measurements, visit:
`https://docs.openstack.org/admin-guide/telemetry-measurements.html`.

ceilometer-agent-notification

Unlike the other two ceilometer agents, the notification agent does not work with the polling method. However, when a new action is carried out by any OpenStack service, the incident will be communicated through the AMQP bus. The ceilometer-notification agent monitors the message queues for notifications and consumes the messages generated on the notification bus, then transforms them into Ceilometer metering data or events.

ceilometer-collector

As the name states, the ceilometer collector runs on the central management server and collects the metering data from all the ceilometer agents mentioned previously. The collected metering data is stored in a data store, or can be configured to send to the external monitoring service, such as the Nagios monitoring service:



Orchestration service - Heat

In OpenStack, the heat service takes care of the arranging and coordination of automated tasks, ultimately resulting in the process workflow for managing the entire lifecycle of infrastructure and applications within OpenStack clouds. The human readable format code in a text file, namely hot templates, are used by the heat service to manage the lifecycle of resource within the cloud.

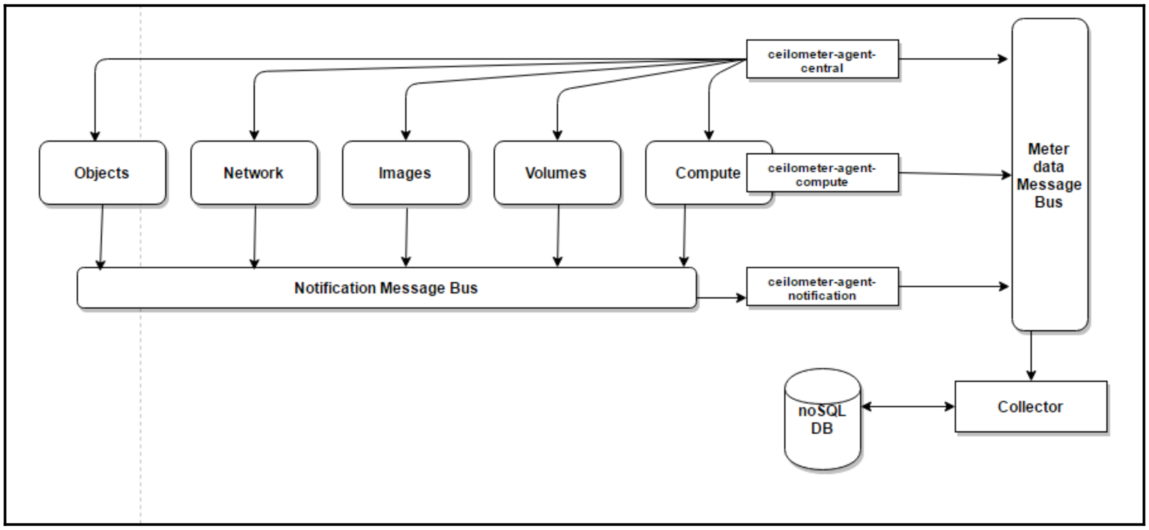
In simple words, any sequence of activities carried out by the end user using OpenStack Horizon can be coded in a template and then the same operations can be accomplished by the heat service by processing that template in a heat engine.

For example, the end user activities in Horizon are as follows:

1. Click the **Create VM** button.
2. Select the **Ubuntu** image.
3. Choose the `m1.small` flavor.
4. Choose the `my_key` keypair.
5. Hit the **Launch** button.

The same action can be accomplished using the following template:

```
heat_template_version: 2015-04-30

description: Simple template to deploy a single compute instance
resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      key_name: my_key
      image: ubuntu-trusty-x86_64
      flavor: m1.small
```



heat-api component

The `heat-api` service accepts and responds to the OpenStack-native REST API and passes the request to the heat engine over Remote Procedure Call.

heat-api-cfn component

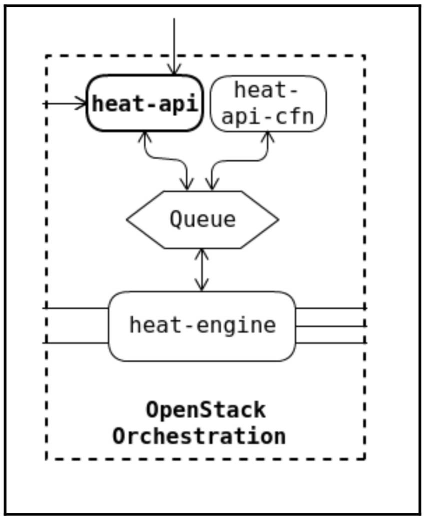
The heat engine is also compatible with AWS Cloud Formation templates. The `heat-api-cfn` accepts the API request and sends them to the heat engine over RPC to process them.

heat-engine

The heat engine handles the templates orchestration and reports back to the API consumer.

OpenStack - conceptual architecture

The following conceptual architecture diagram summarizes the offerings and the relationship between all services in OpenStack:



To sum up:

- KeyStone handles authentication and authorization
- Nova handles the entire lifecycle of virtual machines by managing the hypervisor
- Glance provides plug and play OS images to the Nova service for virtual machine creation
- Neutron handles the networking connectivity for the virtual machines
- Cinder provides secondary (in some cases, primary) hard disk, to the virtual machines
- Swift offers storage space to store Glance images and Cinder volume backup

Designing your deployment

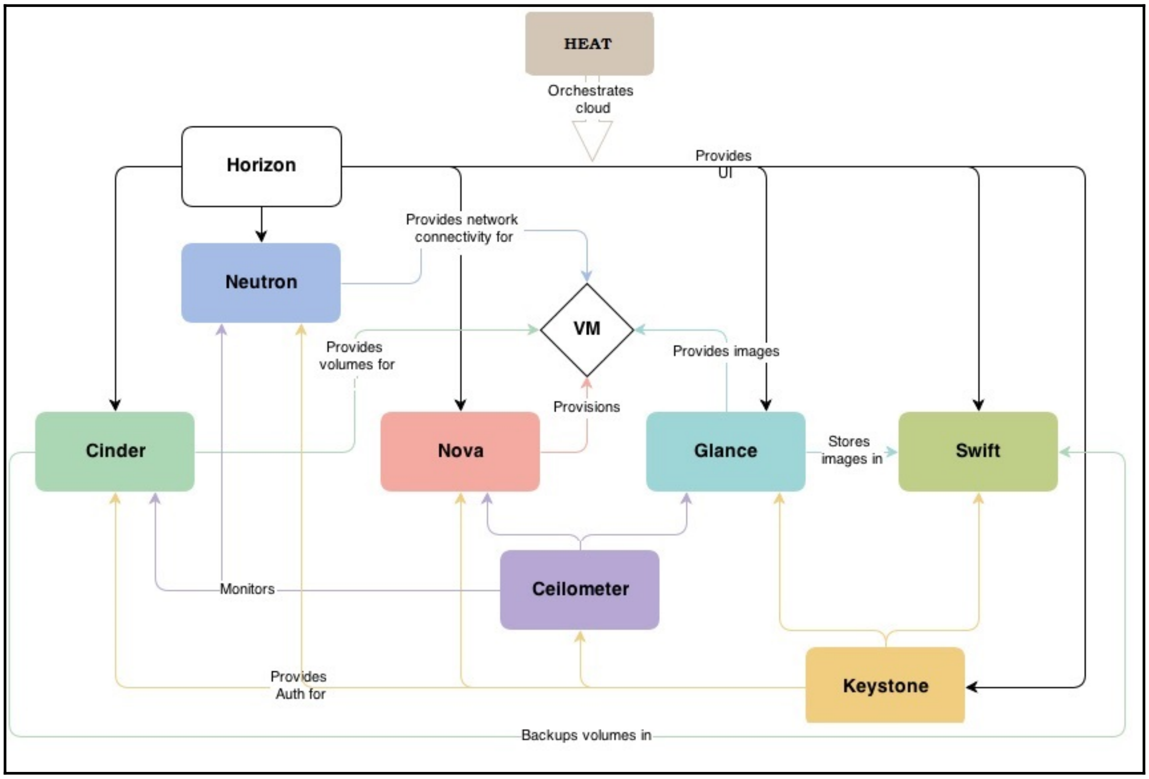
After swimming across the complex OpenStack architecture, now you have a good understanding of how individual processes and daemons in each OpenStack service work together in building the Infrastructure-as-a-Service platform (OpenStack).

OpenStack is modular in nature and capable of running each OpenStack service on an independent server. At the same time, it is not a good practice to deploy each OpenStack service in individual nodes as it only adds to the server maintenance cost and not the performance.

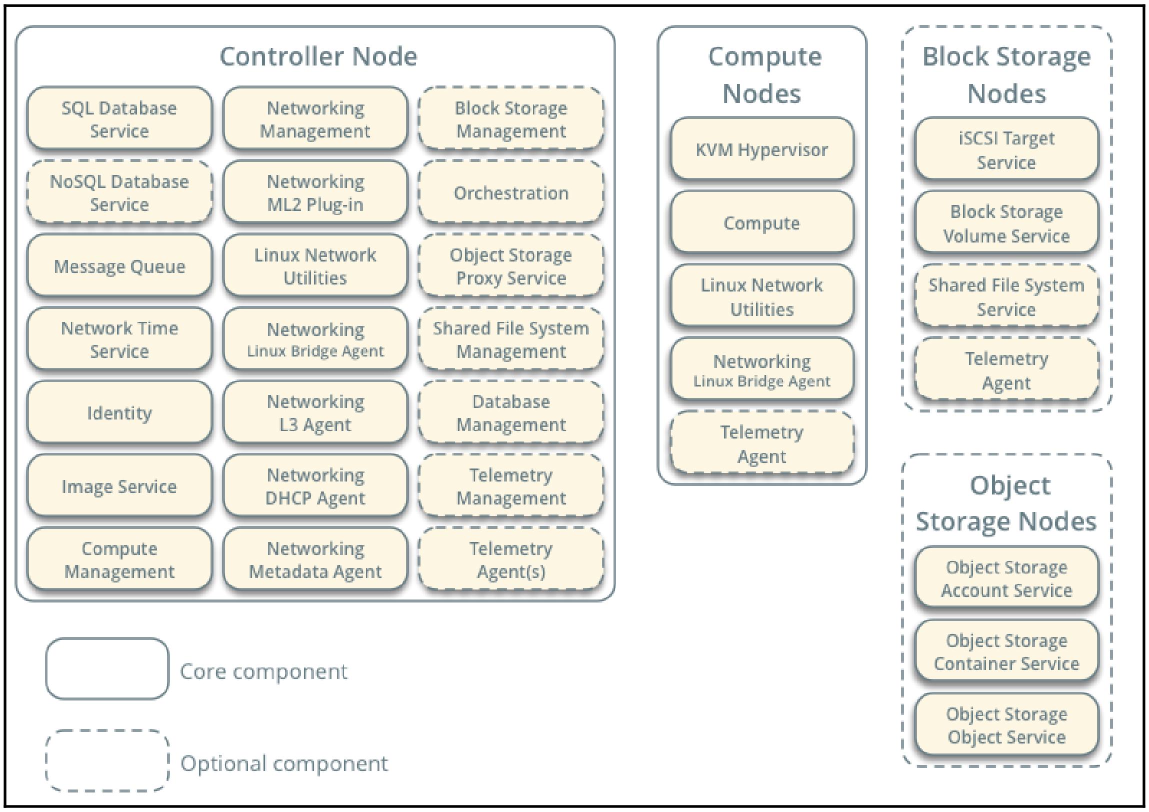
Due to its modular design, OpenStack components have the flexibility to get deployed on a single server to bring up the IAAS platform. I used to call such setups all-in-one node OpenStack setups similar to the one we built using DevStack as part of `Chapter 1`, *Day 1 - Build Your Camp*.

When planning your OpenStack design, you have full freedom to mix and match any OpenStack services on any server. However, it is always good to know the recommended design before we start putting our combination in it. In this section, I will walk you through some of the choices one should consider when designing and building the OpenStack cloud.

The following service layout diagram shows the best combination to run the OpenStack services across the nodes:



From the preceding diagram, you could see the KeyStone, Glance, Nova (excluding `nova-compute`), neutron, and database running on the controller node. The `nova-compute` service and the necessary networking agents are running on the compute node. The Cinder service and the Swift services are configured to run on the storage node. Usually, the server with heavy CPU and RAM is allocated to run as the compute node. The servers with a large disk and I/O are assigned to run the storage services such as Swift and Cinder:



As we know, all the virtual machines will be created and running on the compute node, and each virtual machine holds an isolated CPU and RAM resource from the compute node. So, when designing the compute cluster, you must consider the number of processors and the RAM that the servers have before allocating the server to compute cluster. In the world of virtualization, we also have the flexibility to overcommit the resources available. Overcommitting the resources is the practice of assuming the extra virtual memory with no guarantee that physical storage for it exists.

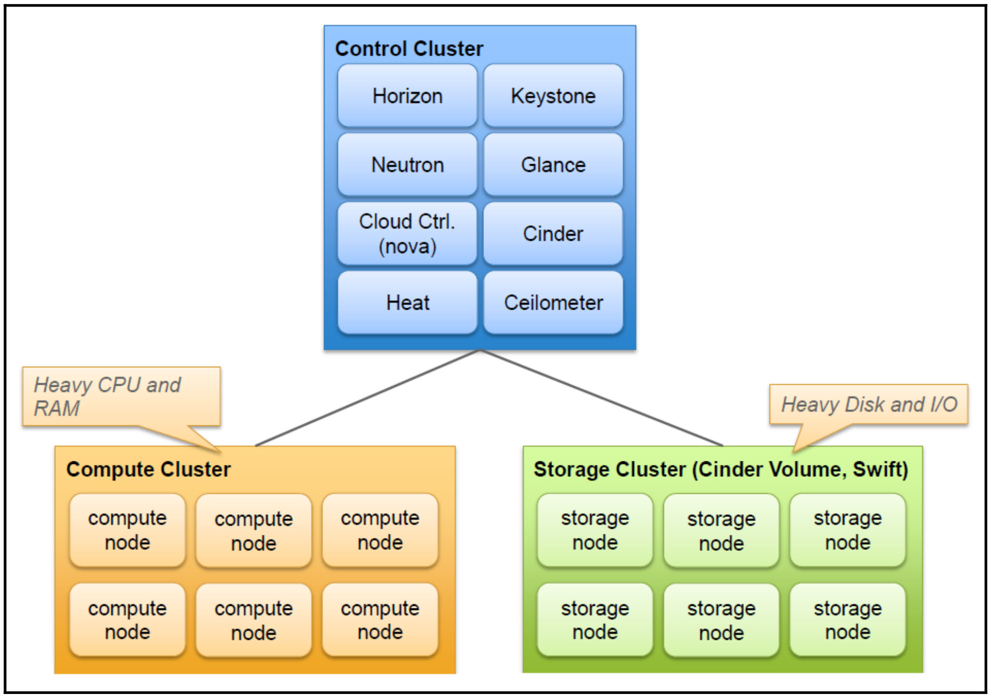
For example, if we have a server with 16GB RAM, and overcommit the memory in 2:1 ratio, then the operating system would tell the hypervisor that we have *16 x 2 = 32* GB RAM in our system, which allows the hypervisor to create 8 VMs with 4GB RAM each.

The Linux system allows resources to overcommit for CPU, RAM, and disk. In OpenStack, the default overcommit ratio for CPU resource is 16:1, and for RAM is 1.5:1.

The compute nodes can be scaled horizontally, which means we could add new compute nodes to the cloud setup on the go without changing the configuration settings in any other nodes in the OpenStack cloud. When a new compute node is added to the cloud environment, then it will start sharing the messages with the controller node, which will result in putting more processes and network weight on the controller. So, we must balance the controller resources by choosing the right servers to run the controller services.

Typical production design

The following diagram shows the typical production-ready OpenStack cloud environment with load balancers and high availabilities in services:



For the production-ready OpenStack environment, we must configure at least three controller nodes to run all of the services in high availability mode. Like I mentioned earlier, the load and resources usage of a controller node is directly proportional to the number of compute nodes in the cloud setup. So, an increase in the number of compute nodes should be balanced with the horizontal scaling of the controller node. In large scale environments, the number of controllers for high availability may vary depending on the size of the cloud. The load balancer servers will help balance the API request load across the three highly available controller nodes. You will have noticed the storage clusters in the preceding diagram; the servers with high IOPS are preferred for Cinder services. The volumes created in the Cinder backend devices are linked directly to the virtual machines for read and write operations, which required high IOPS devices for the virtual machine to perform faster. Likewise, the servers preferred for Swift should have huge storage space, not necessarily the high IOPS disk, as most of the data stored through the Swift service are Cinder volume backups and Glance images, which are not getting used frequently.

Summary

We have decoupled each OpenStack service from the complex architecture diagram and analyzed individual daemons and processes of each OpenStack service in detail. Now, you can understand how various services work together to bring up the **Infrastructure-as-a-Service** platform. Furthermore, you also learnt the work of an individual daemon and process in OpenStack. We also learnt the design principle of OpenStack cloud and the typical deployment topology.

In the next chapter, `Chapter 4`, *Day 4 - How Stuff Works*, we will walk through the step by step process of how VM provision happens in OpenStack when the user initiates VM creation from the Horizon.

4
Day 4 - How Stuff Works

After the previous chapter, we are now thorough with the architectural design of OpenStack. We have gone through the work of each service and demons of all the core OpenStack components in detail. Now, it is time to see how these OpenStack components work together in bringing up the **virtual machine** (**VM**) when the user initiates VM creation from Horizon.

I firmly believe in the idiom *A picture is worth a thousand words*. It refers to the notion that a complex idea can be conveyed with just a single still image more efficiently than with a description.

So, in this chapter, I will try my best to illustrate the step-by-step process of the VM provision life cycle in OpenStack and the interrelationship between the OpenStack services.

Idle state of OpenStack services

Let's assume we have an OpenStack cloud installed and running in our lab. As part of the installation, we have created a new user and a new project, say, for example, a user called Neo and a project called Matrix. Then, the Neo user is mapped in the matrix project/tenant and assigned the member role. As we have already learned about the different roles and their privileges in `Chapter 2`, *Day 2 - Know Your Battalion*, you have some idea of what freedoms a user with the member role and a user with the admin role can have in accessing the OpenStack services.

It is possible to associate users with multiple projects/tenants, and the same user could also have been granted different roles in various projects. It is also important to be aware that the admin role is global, not per project, so mapping a user to the admin role in any project gives the user administrative rights across the whole OpenStack cloud.

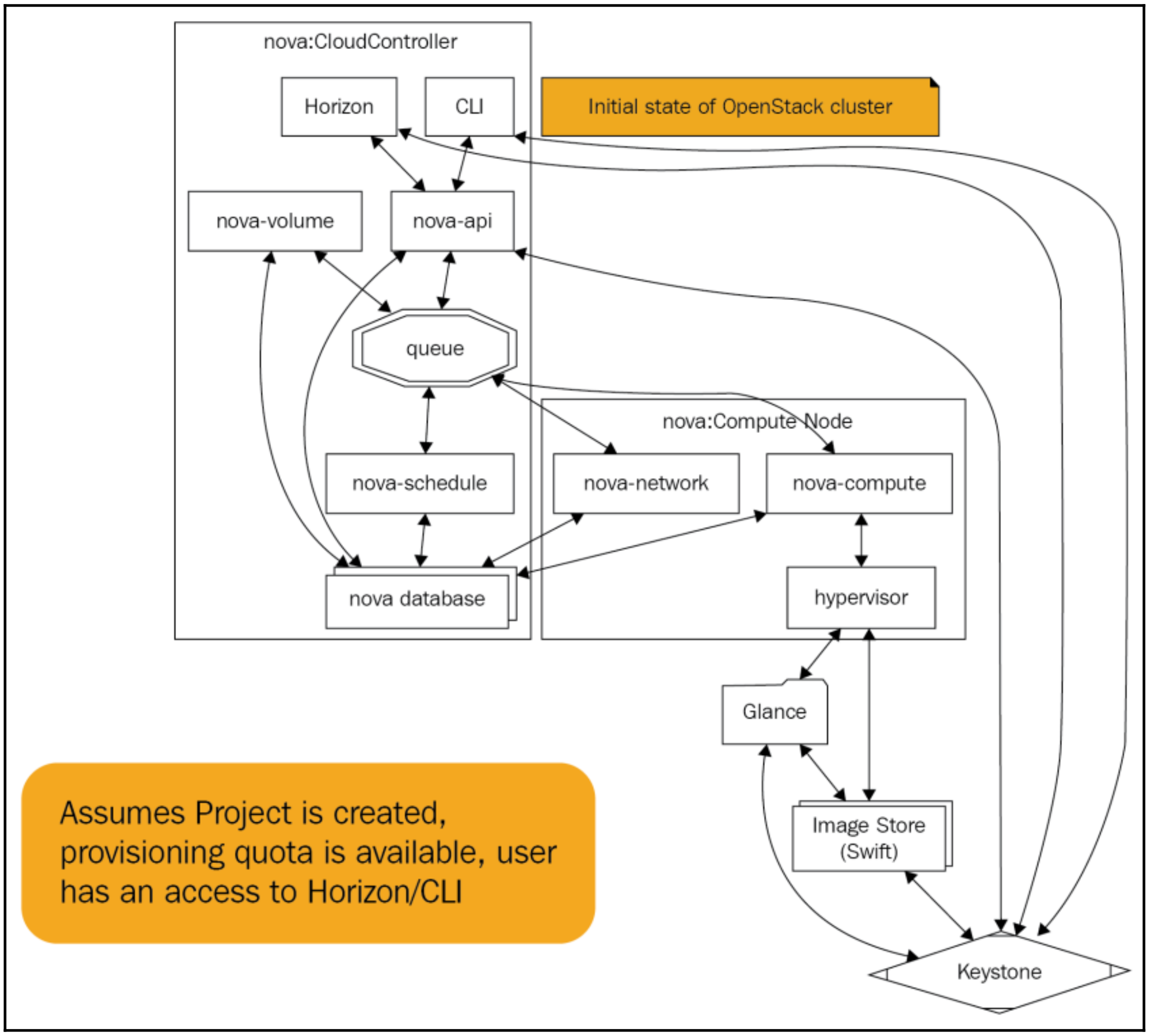
To find out more about roles and project mapping, visit:
`https://docs.openstack.org/ops-guide/ops-users.html`.

Now, coming back to our assumption, the Neo user is mapped in the project called Matrix and granted the member role in our OpenStack cloud. We know users with the member role have standard user access to all of the core OpenStack projects. Neo has access to create a new VM, upload a new image, create a volume, create a new tenant network, and the list goes on. Here, Neo's access permission is limited only within the particular tenant called Matrix.

Now, let's walk through all of the stages of the VM creation process in detail, whenever the user Neo hits the **create VM** button in the OpenStack Horizon dashboard.

For the sake of easy understanding, I have illustrated the flow with legacy `nova-network` and `nova-volume`, which was part of the Nova project and was used for providing the networking service and the volumes service respectively. Now, the `nova-network` and the `nova-volume` services were deprecated and replaced with the dedicated projects in OpenStack for providing networking and volume services, called the neutron and cinder projects respectively.

The following diagram shows the idle state of OpenStack services:



When I say the idle state of OpenStack, I mean that all of the OpenStack services are running and has active, established a connection with databases, and are performing the default works.

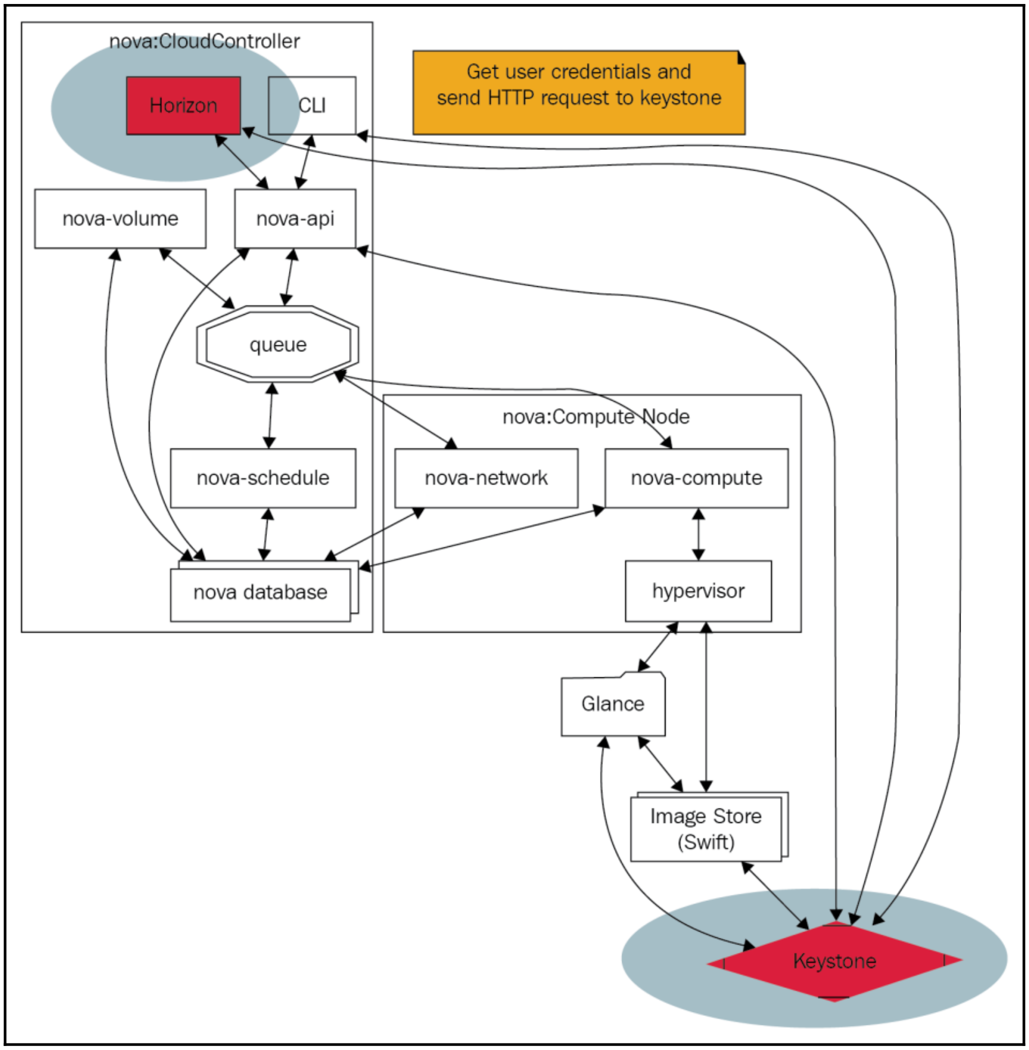
For example, all of the compute nodes periodically publish their status, resources available, and hardware capabilities to the `nova-scheduler` through the queue.

Before we look in-depth at the instance provisioning process, let's list the end user's steps for creating a new VM in OpenStack via Horizon:

1. Log in to Horizon with the provided user credentials.
2. Click the **Create Instance** button.
3. Fill in the prompted **Launch Instance** form with the VM info.
4. Submit the form by hitting the **Launch** button.
5. Wait for the VM status to turn into the running state.

Step 1 - user login - Horizon

When you open the OpenStack Horizon URL in the browser, the OpenStack login page asking for username and password credentials will appear. The end user will enter the user credentials and submit the details. Based on the validation status, a successful attempt will take the end user to the OpenStack service access page. However, a failed attempt will cause an error message asking the end user to verify their user credentials to appear on the login page. This is a process every end user could experience during OpenStack dashboard login. However, as OpenStack administrators/operators, we should know what really happens behind the scenes when users try to log in to Horizon:

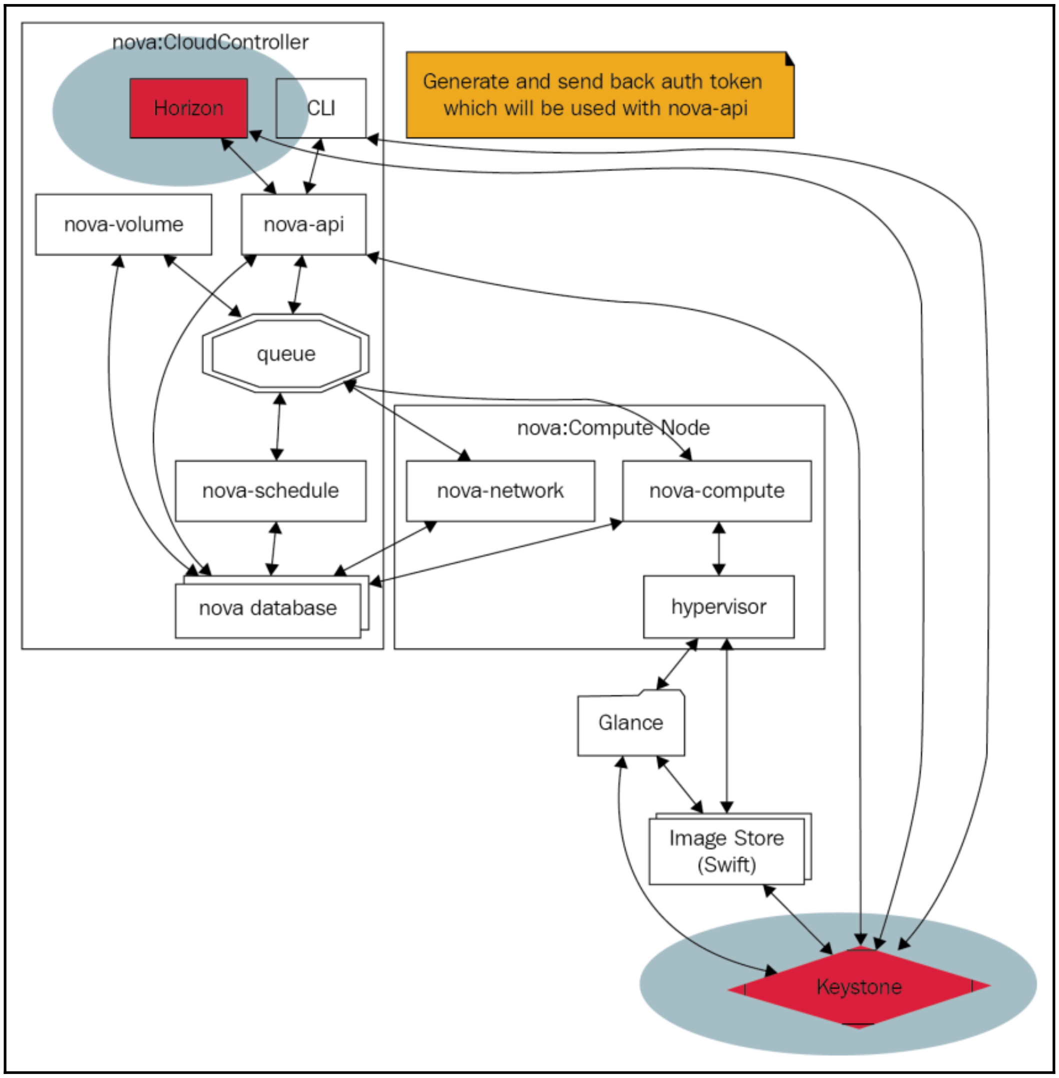


On user login, the following process will take place:

1. Get user credentials from the end user.
2. Submit the user credentials to KeyStone as an HTTP request.

Step 2 - validating AUTH data

The following figure depicts the flow between the OpenStack services for validating the AUTH data:

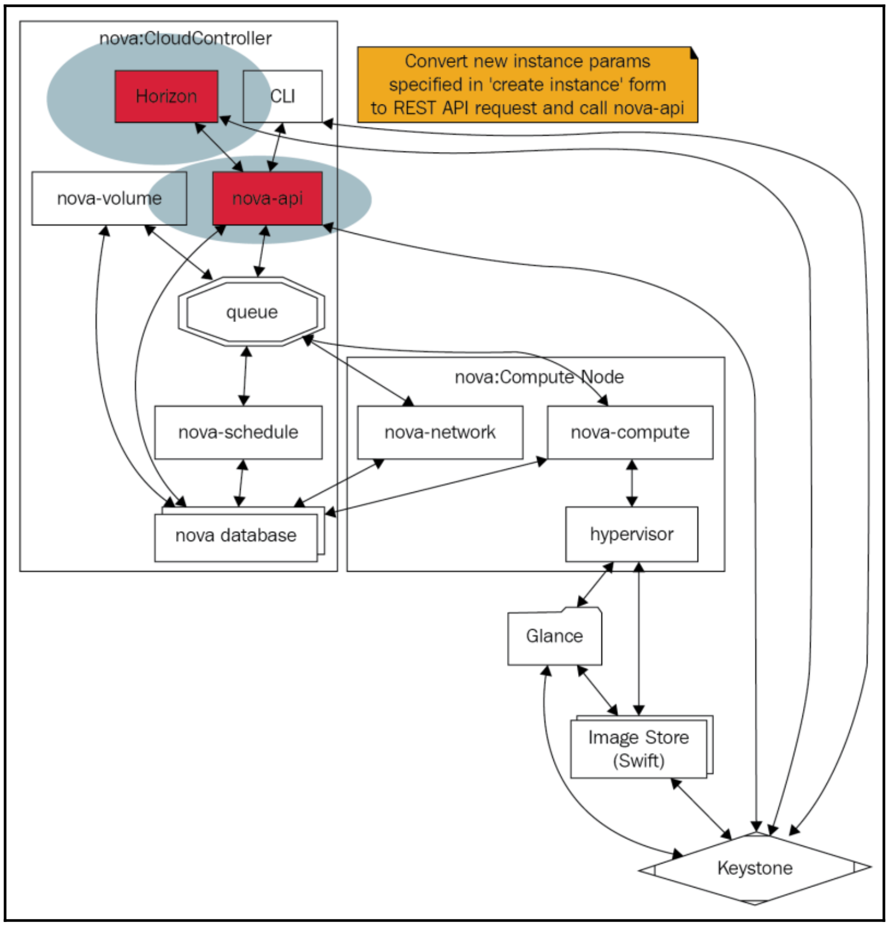


On submitting the user credentials to KeyStone, the following process will take place:

1. KeyStone will verify the authentication data. Upon successful authentication, KeyStone will check the authorization of the authenticated user.
2. Don't get confused between the terms *authentication* and *authorization.* Authentication is the process of confirming whether the user exists in the system or not and then validating the password submitted for the user account. On the other hand, authorization is the process of verifying that you have access to something; in our case, verifying the validated (authenticated) user access to the different projects and services in OpenStack.
3. Upon successful response, KeyStone will provide a unique token to the user.
4. The token is a text block with a unique code and roles encapsulated. The token contains information about the user roles and the access limitations to the projects.
5. The unique token provided by KeyStone will get stored in a browser cookie on the client side. Moreover, from now on, the browser will send the token as part of all the API requests to the OpenStack services.
6. The token expiration time is, by default, 1 hour. Upon token expiration, the user will be logged off automatically.

Step 3 - sending an API request to the Nova API

The following figure depicts the interaction of an API request with Nova components:



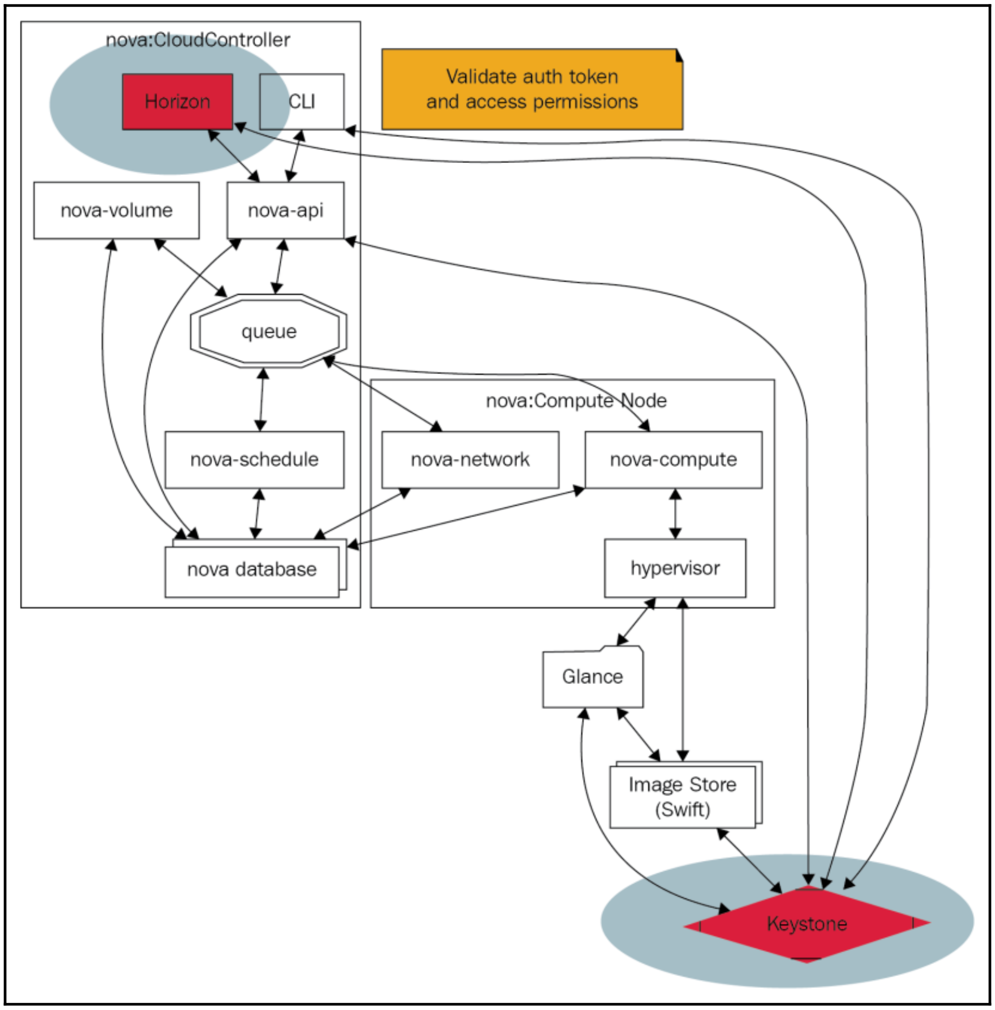
On successful authentication validation, the following process will take place:

1. When the user logs in to OpenStack Horizon and clicks the **Create Instance** button, the **Instance Parameter** form will get prompted. Then, the user will select the VM settings, such as VM name, the number of the VMs, flavor, base image, security group, key pair, and network selection. Optionally, the **launch instance** form also includes an option to create a new cinder volume and has options to inject the user data on the part of the VM creation process.

2. The parameters gathered from the new instance create form will be converted into a REST API request and submitted to the `nova-api` service endpoint.
3. As I mentioned earlier, as part of all API requests to any OpenStack service, the client-side browser will embed the TOKEN saved in the browser cookie with the REST API request and send it to the `nova-api` service.

Step 4 - validating the API token

The following figure depicts the flow of validating the API token:



On submitting the API request to Nova, the following process will take place:

1. `nova-api` accepts the request pointed to its endpoint. Before processing the request, `nova-api` will contact KeyStone to validate the AUTH TOKEN and confirm the user has access to the Nova service and ensure the permission to create a new VM.
2. Before verifying the AUTH TOKEN with the KeyStone service, the Nova service must prove its own identity to KeyStone by submitting the Nova user credentials from the Nova configuration file.

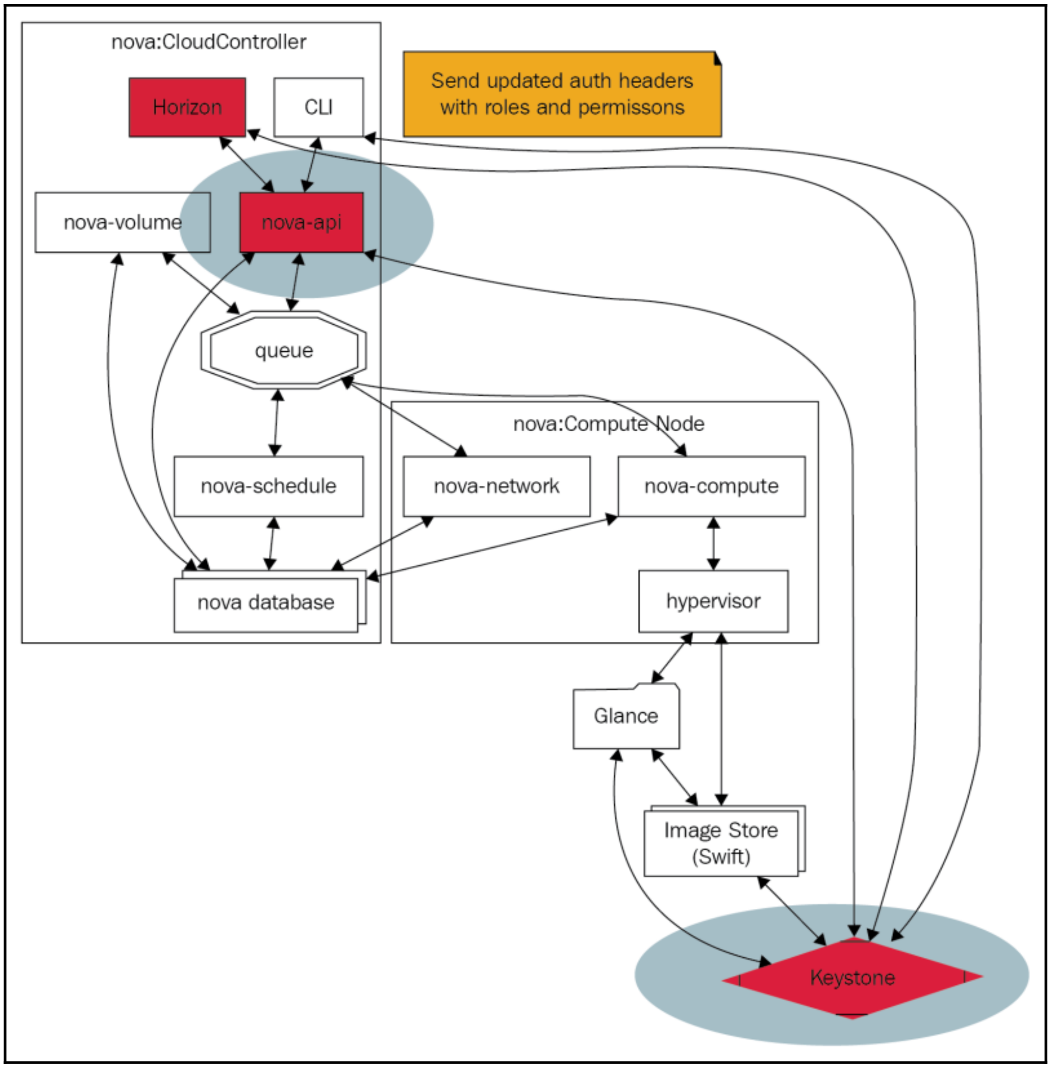
As a part of the OpenStack installation, a username and password will be created for all of the OpenStack services. Then, all of the service users will be mapped to the dedicated project/tenant called Service and will assigned the admin role.

We will walk through all of these procedures during the OpenStack installation steps in `Chapter 8`, *Day 8 - Build Your OpenStack*.

3. The process for verifying the AUTH TOKEN and its own identity is not limited only to the Nova service. All of the OpenStack services will go through the same procedure before processing the actual API request submitted to them.

Step 5 - processing the API request

The following figure depicts the interaction between Nova and KeyStone to process the API request:



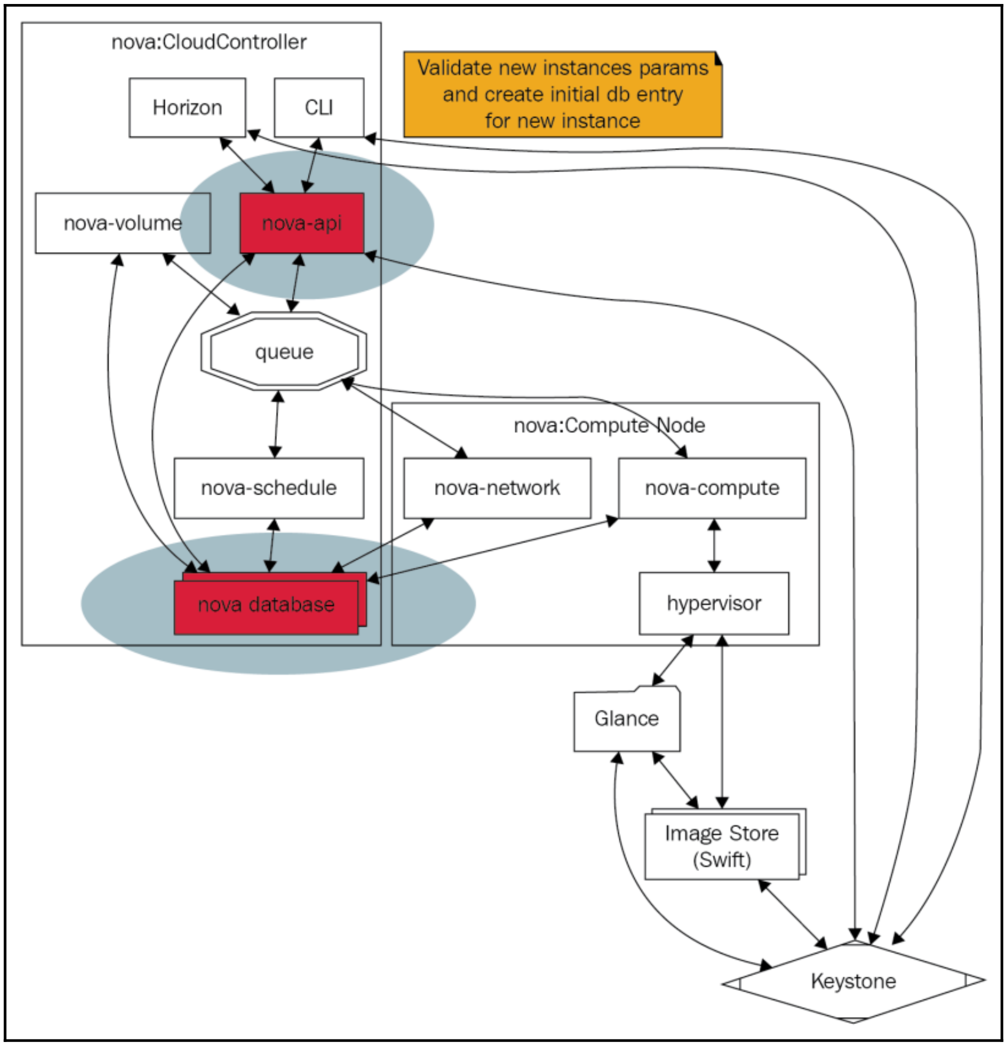
After validating the AUTH token, the following process will take place:

1. On successful validation of the token, KeyStone will send the permission approved response to the `nova-api` service.

2. Then, the `nova-api` service will start to process the **launch instance** request submitted by the end user.
3. From the converted **launch instance** form, the `nova-api` has the following information with the REST-API request:
    - Name of the VM
    - Image as instance source
    - Image name
    - Flavor of the VM, which has RAM/vCPUs/HDD details
    - Name of the security group
    - Name of the key pair that needs to be injected into the VM
    - Name of the tenant network

Step 6 - publishing the provisioning request

The following figure depicts the interaction between `nova-api` and the database service to process the VM provisioning request:



After processing the API request, the following process will take place:

1. The `nova-api` service will interact with the Nova database and create an initial DB entry for new instance creation.

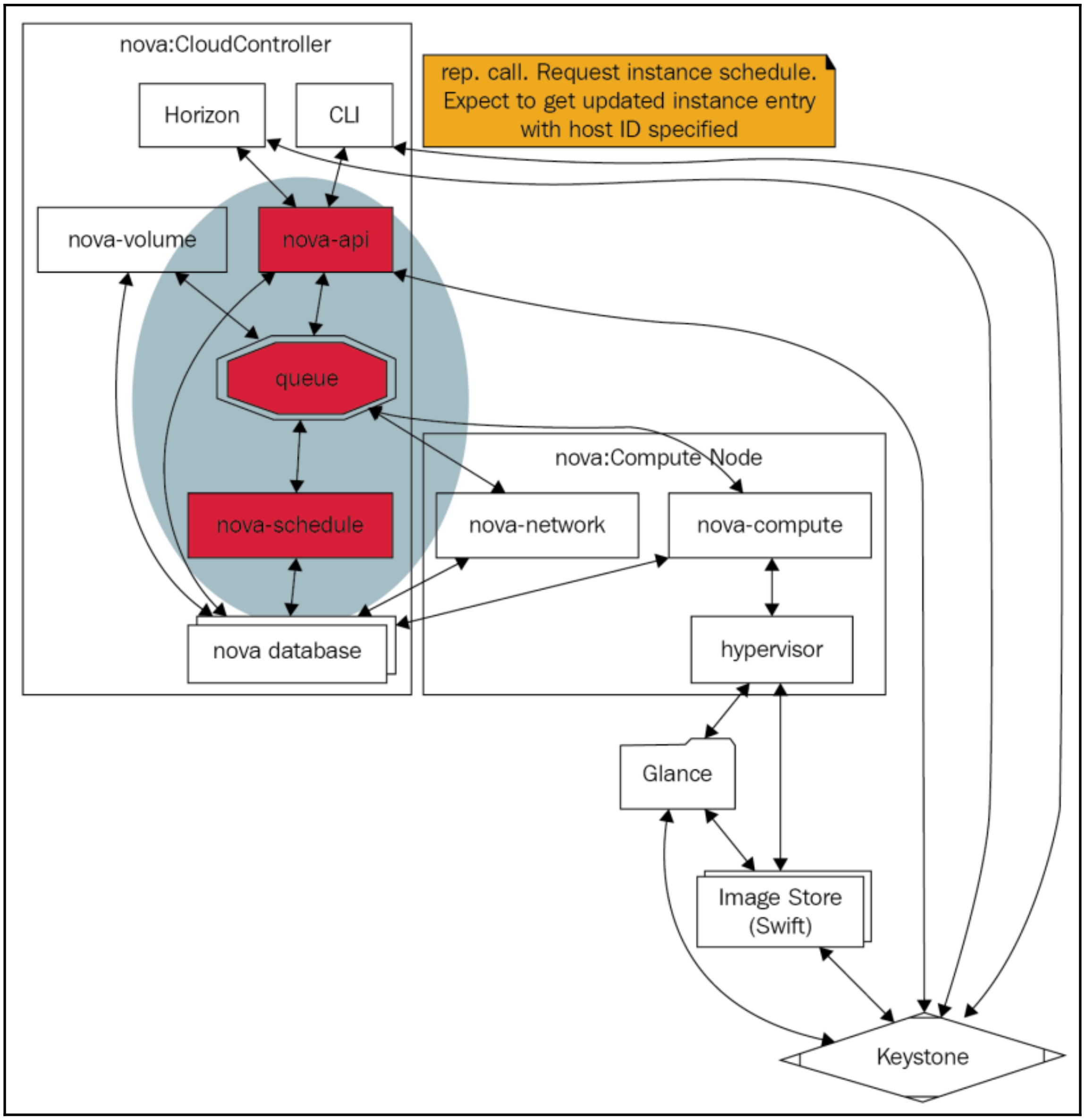
As a part of the OpenStack installation, we will create a dedicated database for each OpenStack service. Each database has its own username and password with full access privileges for the respective OpenStack services. The same information will be configured in the corresponding service configuration files as well.

We will walk through all of these procedures during the OpenStack installation steps in `Chapter 8`, *Day 8 - Build Your OpenStack*.

2. The `nova-api` service will also validate the parameters that were submitted along with the request. The validation process will check whether the chosen flavor passes the minimum requirements of the selected image for the VM.

Step 7 - picking up the provisioning request

The following figure depicts the interaction between the Nova and Queue services to process the VM provisioning:



After publishing the provisioning request, the following process will take place:

1. After creating the initial DB entries for the new VM creation, `nova-api` will send the `rpc.call` request to the `nova-scheduler` daemon excepting to get an updated instance entry with host ID specified.

In OpenStack, all the OpenStack components communicate internally (for example, `nova-api` to `nova-scheduler`) via **Advanced Message Queue Protocol** (**AMQP**) using **Remote Procedure Calls** (**RPCs**) to communicate with one another.
OpenStack messaging has two modes:

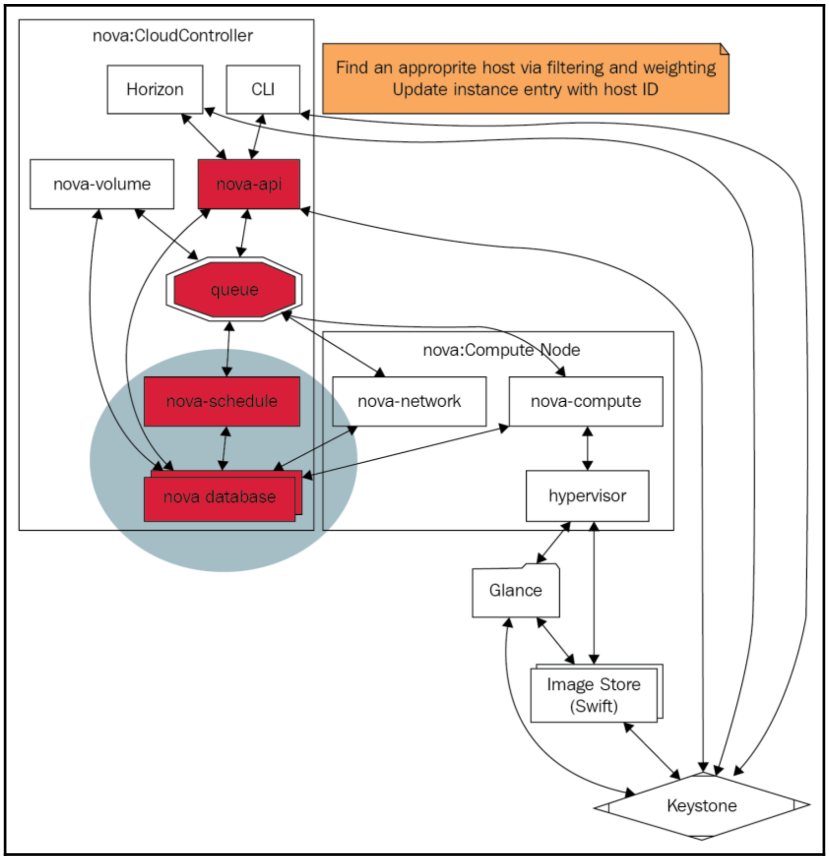
- `rpc.cast`: Don't wait for result
- `rpc.call`: Wait for result (when there is something to return)

However, all the communication between two different services (say, for example, between `nova-api`and`glance-api`) will be carried out using REST API calls.

2. The `nova-scheduler` daemon picks the request from the message queue.

Step 8 - schedule provisioning

The following figure depicts the interaction between the Nova and database services in processing the instance host schedule:

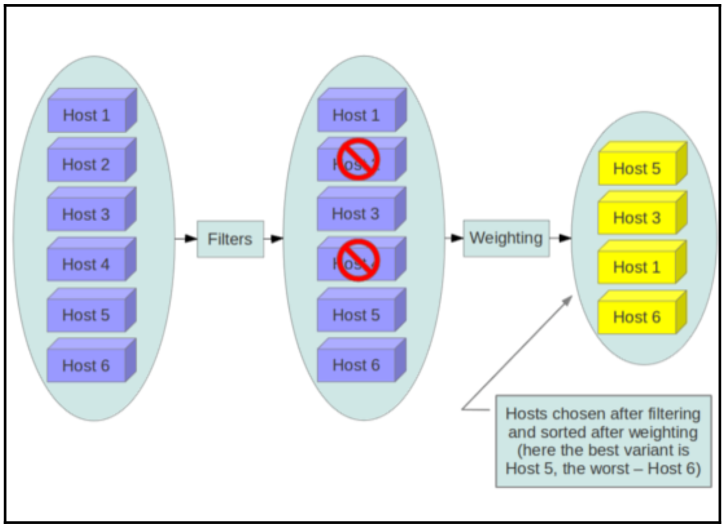


Nova scheduler is a daemon for determining on which compute host the request should run.

As we can see from the preceding figure, `nova-scheduler` interacts with other components through the messaging queue and the central database.

For the scheduling process, the messaging queue is an essential communications hub. All the compute nodes periodically publish the compute host status, resources available, and hardware capabilities to `nova-scheduler` through the messaging queue. Whenever the API request for new VM provisioning comes in, `nova-scheduler` picks up the request from the messaging queue.

Then the `nova-scheduler` daemon brings together all of the collected data and uses it to make decisions about on which compute host the new VM should get provisioned:


Host 1
Host 2
Host 3
Host 4
Host 5
Host 6
Filters
Host 1
Host 3
Host 5
Host 6
Weighting
Host 5
Host 3
Host 1
Host 6
Hosts chosen after filtering and sorted after weighting (here the best variant is Host 5, the worst – Host 6)

The preceding diagram gives us an overview of how `nova-scheduler` does its job. The whole decision-making process divides into two stages:

- Filtering phase
- Weighting phase

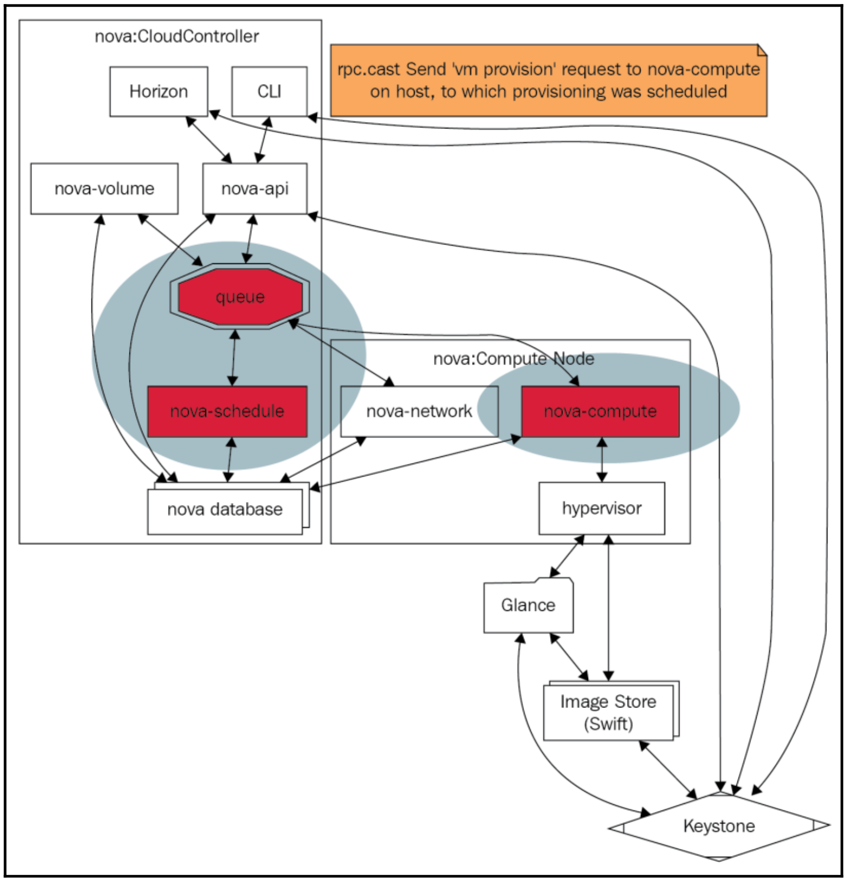
The filtering stage will make a list of suitable hosts by applying filters. Then, the weighting phase will sort the hosts according to their weighted cost scores, which are given by applying some cost functions. As a final step, the top-ranking host based on the weighting process will be selected to provision the user's instance creation request:

1. The `nova-scheduler` daemon interacts with the Nova database to find a suitable host via filtering and weighing.
2. After selecting the appropriate compute host for a new VM provisioning, the `nova-scheduler` daemon will return an updated instance entry in the database with a chosen host ID.
3. Notably, during the initial Nova database entries for the new VM, `nova-api` will set the host ID value to null. Later, the `nova-scheduler` will update the same with the chosen host ID.

To find out more about the filters available, visit:
`https://docs.openstack.org/ocata/config-reference/compute/schedulers.html`.

Step 9 - starting the VM provisioning

The following figure depicts the interaction between Nova services in processing the instance provision:



After the scheduling process, the following process will take place:

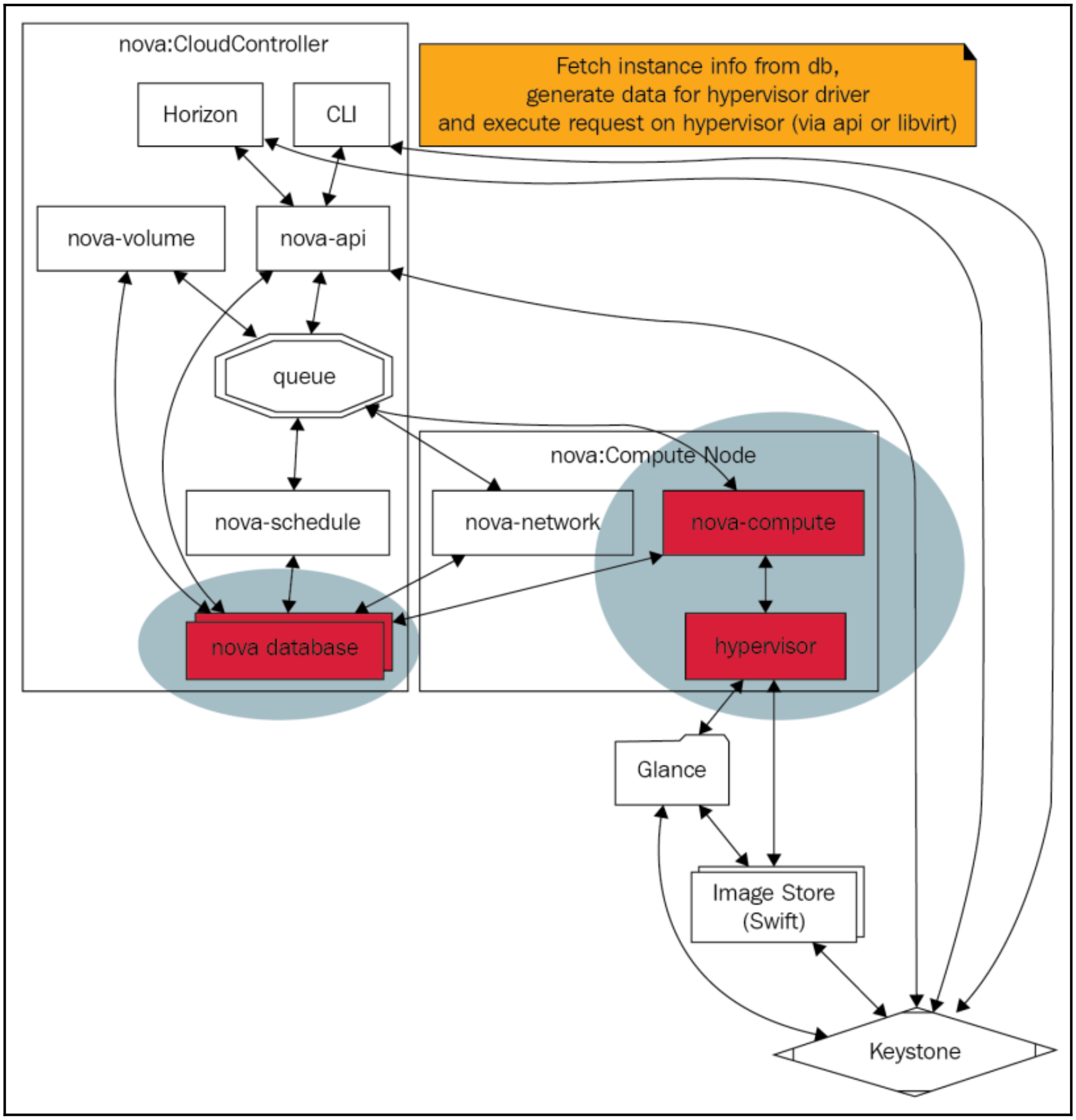
1. The `nova-scheduler` daemon publishes the `rpc.cast` message to the compute queue (based on host ID) to trigger the VM provisioning.
2. The `nova-compute` service picks the request from the messaging queue.
3. Then, the `nova-compute` service sends the `rpc.call` request to the `nova-conductor` to fetch the instance information such as host ID and flavor info.

For the sake of easy understanding, I have shown only the core nova components in the preceding figure. The `nova-conductor` daemon has been ignored.

The `nova-compute` service running on the compute host has no direct access to the database. This is because, when one of your compute nodes is compromised, then the attacker has (almost) full access to the database. With the `nova-conductor` daemon, the compromised node could not access the database directly, and all the communication can only go through the `nova-conductor` daemon.

Step 10 - starting VM rendering via the hypervisor

The following figure depicts the interaction between the Nova and database services in rendering the instance in a compute node:


nova:CloudController
Horizon
CLI
Fetch instance info from db, generate data for hypervisor driver and execute request on hypervisor (via api or libvirt)
nova-volume
nova-api
queue
nova:Compute Node
nova-schedule
nova-network
nova-compute
nova database
hypervisor
Glance
Image Store (Swift)
Keystone

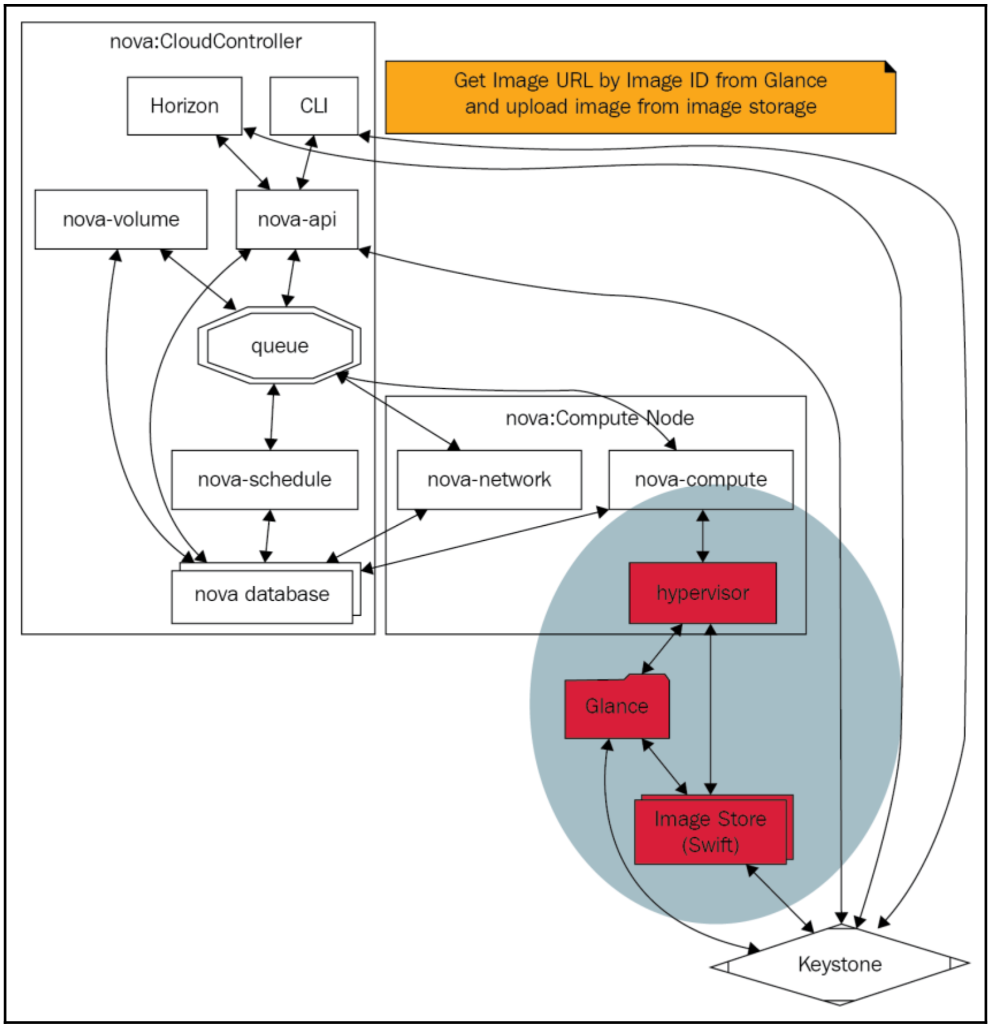
Nova compute is a worker daemon which mainly creates and terminates the VMs using the appropriate hypervisor API.

On processing the VM rendering, the following process will take place:

1. The `nova-conductor` daemon picks the `rpc.call` request from the messaging queue.
2. The `nova-conductor` daemon starts interacting with `nova-database` for the VM information.
3. `nova-conductor` returns the instance information to `nova-compute` via the messaging queue.
4. `nova-compute` picks the instance information from the messaging queue.
5. Now, the `nova-compute` daemon generates the data for the hypervisor driver based on the information collected from the database via `nova-conductor` and then executes the `instance create` request on the hypervisor using the appropriate hypervisor API.

Step 11 - requesting the base image for VM rendering

The following figure shows the interaction between the Nova and Glance image services in rendering the VM:


nova:CloudController
Get Image URL by Image ID from Glance
and upload image from image storage
Horizon
CLI
nova-volume
nova-api
queue
nova:Compute Node
nova-schedule
nova-network
nova-compute
nova database
hypervisor
Glance
Image Store
(Swift)
Keystone

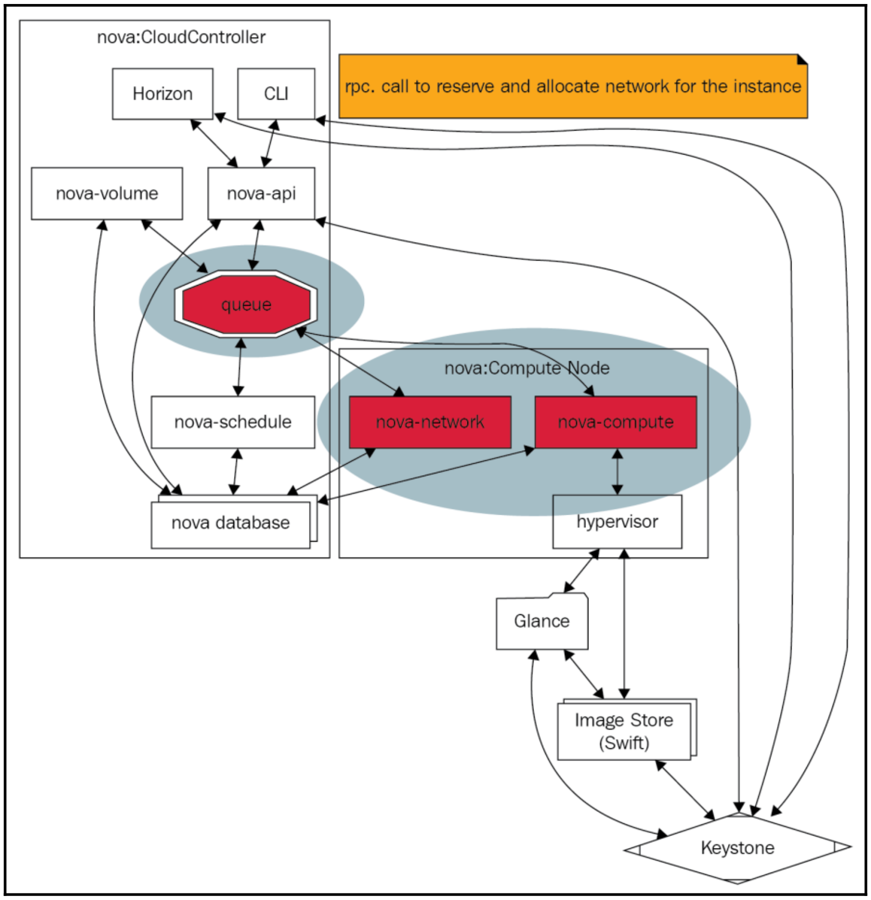
The Glance (image service) project in OpenStack provides the services for discovering, registering, and retrieving the golden images for VM provisioning.

As part of processing the VM rendering, the following process will take place:

1. In a meanwhile, the `nova-compute` daemon sends the REST API call along with the AUTH TOKEN to the `glance-api` service.
2. The REST API request will get the image URI from the Glance service by referring to the image ID submitted with the **instance create** form, and then upload the image to the chosen compute host from the image storage.
3. The `glance-api` service validates the AUTH TOKEN with KeyStone.
4. On successful validation of the AUTH TOKEN, the Glance service will allow the `nova-compute` daemon to download the image using the URI from the Glance store (optionally, the Swift storage service).

Step 12 - configuring the network for an instance

The following figure depicts the interaction between the Nova and networking services in processing the network configuration for an instance:



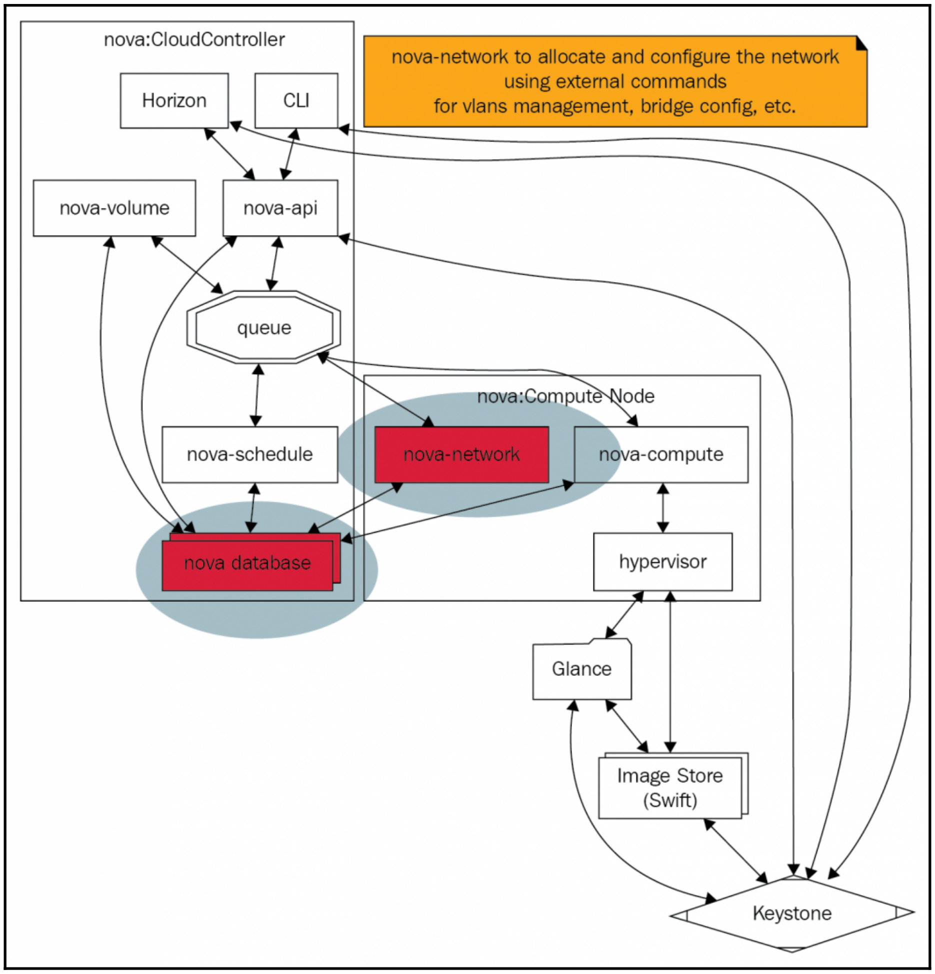
As part of configuring the network for an instance, the following process will take place:

1. Now, the `nova-compute` daemon sends the REST API call along with the AUTH TOKEN to the Neutron API service to allocate and configure the network (IP address) for an instance.

In the preceding figure, I have depicted the legacy `nova-networking` service by replacing the neutron service, for beginners to follow the VM provisioning workflow easily. Moreover, we will discuss the neutron service and its extended functionality in detail in `Chapter 5`, *Day 5 - Networking Strategy*.

2. The `neutron-server` validates the AUTH TOKEN with KeyStone:

Unlike neutron (the dedicated project for OpenStack networking), the `nova-network` service is the part of the Nova project itself. So, the AUTH TOKEN validation is negated in the legacy Nova network.



3. On successful validation of the AUTH TOKEN, the neutron networking service will create a virtual network interface card (vNIC) for the new VM on the compute host using the networking driver.
4. Then neutron configures the IP, gateway, DNS name, and L2 connectivity for the new VM.

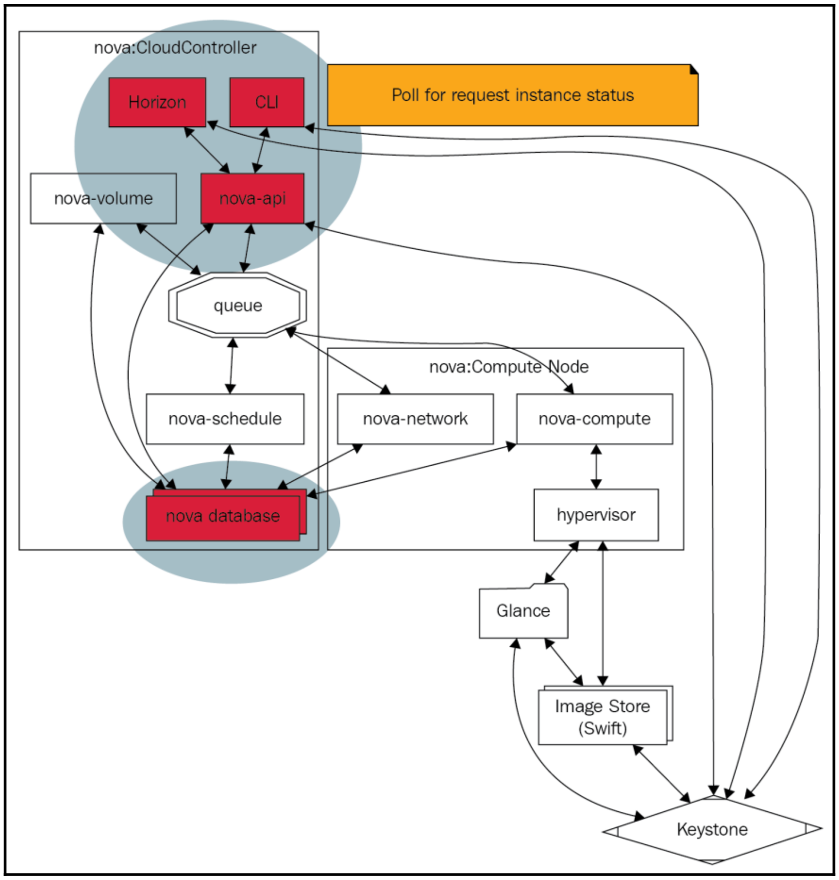
Why are there two different networking methods in OpenStack?

The first is called legacy networking (`nova-network`). Like `nova-scheduler` and `nova-conductor`, `nova-network` is also a subprocess implanted in the `nova-compute` project. The legacy `nova-networking` model has many limitations, such as creating complex network topologies, extending its backend operation to vendor-specific technologies, and providing project-specific networking configurations to each project.

To overcome the limitations in the legacy `nova-network`, the dedicated networking model called neutron was added to OpenStack to provide networking-as-a-service functionality.

Step 13 - VM in running state

The following figure depicts the interaction between the Nova and database services in providing the VM status to the end user:



On successful completion of all the above stages, the following process will take place:

1. `nova-compute` has all of the information, including image, network, and other VM info, to generate the request for the hypervisor driver, and then `nova-compute` will pass all the information about the VM (in a single message) to the hypervisor for creating an instance.
2. Meanwhile, the `nova-api` service endlessly polls the request for the instance status to the database.

The following table shows the instance status in the Horizon dashboard at the various stages of polling requests during the provisioning process:

Status	**Task**	**Power state**	**Steps**
Build	scheduling	None	3-10
Build	networking	None	11
Build	`block_device_mapping`	None	-
Build	spawning	None	12
Active	none	Running	

As I mentioned earlier, the database is the brain of OpenStack as all the information is saved in it. All of the services in OpenStack rely on the database information. `nova-api` will poll the request for the status of all of the services to the database for presenting in the Horizon dashboard.

Sometimes, as a part of the troubleshooting process and for fixing the error state of the instance, we could do a few hacks on the database (strongly not recommended in the production environment) by changing the instance status manually. Since all of the OpenStack service relies on the database information, Horizon will present the value as it is in the database without cross-checking the DB entry.

Summary

In this chapter, we have gone through each step of the request workflow for provisioning an instance in detail. I firmly believe readers will have gained a clear picture of the VM creation life cycle in OpenStack and knowledge of how each component of OpenStack are interconnected and work in bringing up a new VM.

In `Chapter 5`, *Day 5 - Networking Strategy*, we will focus on OpenStack networking in detail and the extended features available in OpenStack neutron.

5
Day 5 - Networking Strategy

In this chapter, you will be introduced to the most interesting, as well as the most complex component in OpenStack, code named neutron. To understand the neutron better and easier, it is a must to have some basic understanding of the networking world. So, let me precisely recap some basic and essential networking concepts followed by:

- Networking basics
- OpenStack networking
- Network types
- Neutron offerings
- Network traffic flow

Networking basics

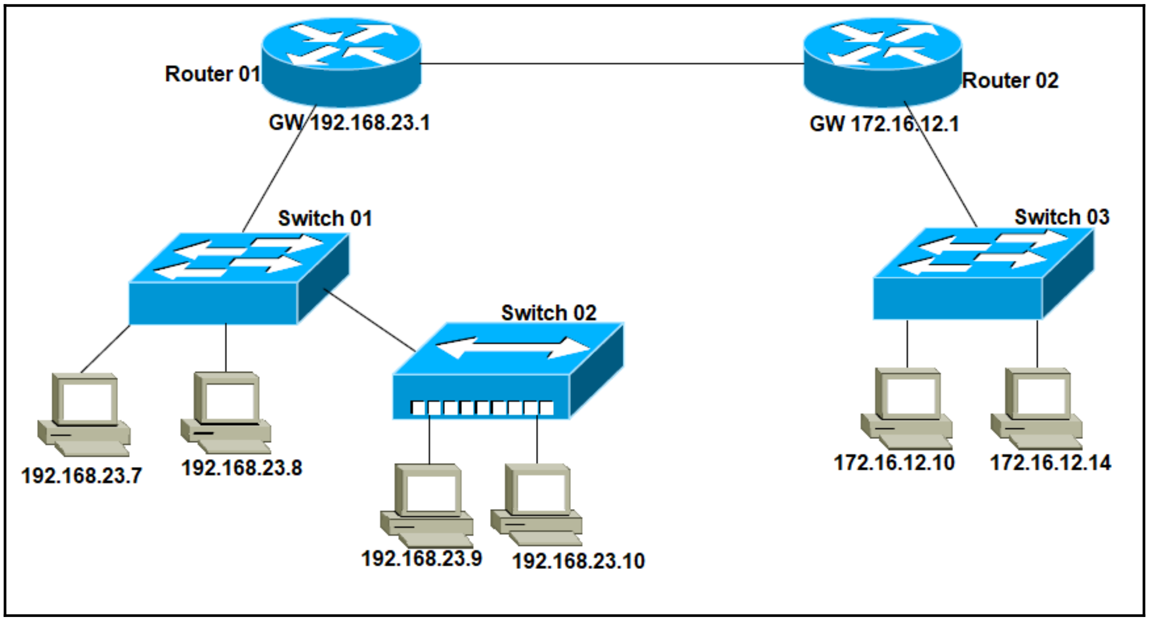
Ethernet is the most widely installed **Local Area Network** (**LAN**) technology. Ethernet is a networking protocol that describes how networked devices can format data for transmission to other network devices on the same network and how to put this data out on the network connection. Every NIC connected to an ethernet network has its unique hardware address, commonly known as a **Media Access Control** (**MAC**) address, which is represented as a hexadecimal string, such as `08:00:37:c9:58:56`.

The **Internet Protocol** (**IP**) is a set of rules by which data is sent from one computer to another on the internet or another network. Each computer (known as a host) on the internet/intranet has at least one IP address that uniquely identifies it from all other computers on it. A typical IP address (IPv4) looks like `192.168.1.23`.

Switches are layer-2 network devices that connect various network devices together through the ethernet to allow communication between these devices. Switches forward the packet received on one port to another port based on the destination MAC address in the packet header, so that the formatted data will reach the desired destination node.

Routers are layer-3 network devices that enable data packets to communicate between two hosts (computer/device) on different networks.

As shown in the following figure, typically, a network layout goes like, a router connects physically via a network cable to a switch and then physically, again via a network cable to the **Network Interface Cards** (**NIC**) in any network devices you may have, such as computer and printers:



VLAN is a network isolation technology in the managed switch that helps a switch to perform as multiple independent switches. By configuring VLAN in the switch settings, we can isolate the data traffic of the two computers that are linked to the same switch.

Dynamic Host Configuration Protocol (**DHCP**) is a client/server protocol that acts as a central management server for the distribution of IP addresses within a network. DHCP is also used to configure the subnet mask, default gateway, and DNS server information on the device.

An **Overlay network** is a logical network that is built on top of a physical network. The nodes that are connected to the overlay network will act like it has a direct link between the two separated nodes. However, the two nodes in the overlay network have many virtual or logical links through many physical links in the underlying network.

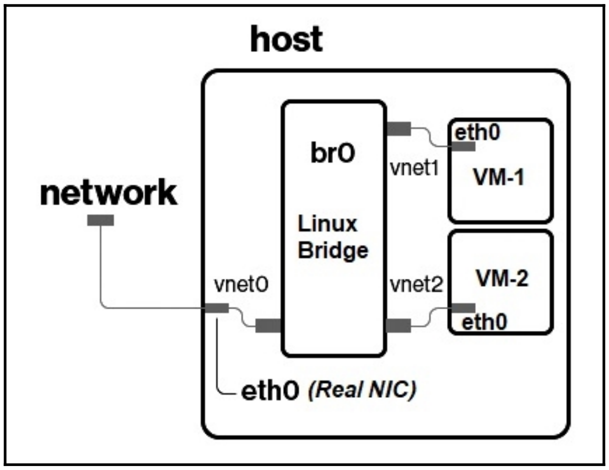
Data encapsulation is a process of formatting the data, where the data is enlarged with successive layers of control information before transmission on a network. The inverse of data encapsulation is decapsulation, which is a process of unpacking the successive layers at the receiving end of a network to process the original data.

Network address translation (**NAT**) is a process of remapping the source or destination IP address in the packet headers while they are in transit across a traffic routing device. Typically, the sender and receiver applications are not aware that the IP packets are being modified.

Network namespaces provide an isolated network stack for all the processes within the namespace. This includes network interfaces, routing tables, and IP tables rules. Using a namespace, you can use the same identifier multiple times in different namespaces.

Software-Defined Networking (**SDN**) is a network architecture approach and not a specific product that enables the network to be intelligently and centrally controlled using software applications. The goal of SDN is to allow network administrators to manage the entire network with ease by programming the network traffic from a centralized controller instead of configuring individual switches and can deliver services to wherever they are needed in the network, irrespective of the underlying network technology.

Linux bridge is a software program that typically performs the layer-2 switch functionality. The Linux bridge is a virtual switch that allows one **Virtual Machine** (**VM**) to interconnect with another. This virtual switch cannot receive or transmit data packets on its own unless you bind one or more real devices to it:



OpenStack networking

Before we dive deep into OpenStack networking, let's recap the overview of OpenStack networking from `Chapter 2`, *Day 2 - Know Your Battalion*.

OpenStack neutron is an SDN networking project in OpenStack, focused on delivering **Networking-as-a-Service** (**NAAS**) in virtual computing environments. Characteristically, the neutron service will act as a networking team in the company that manages all the network-related activities. Moreover, the neutron functionality in OpenStack extends across nadir to the zenith of advanced networking.

Legacy nova-network to OpenStack neutron

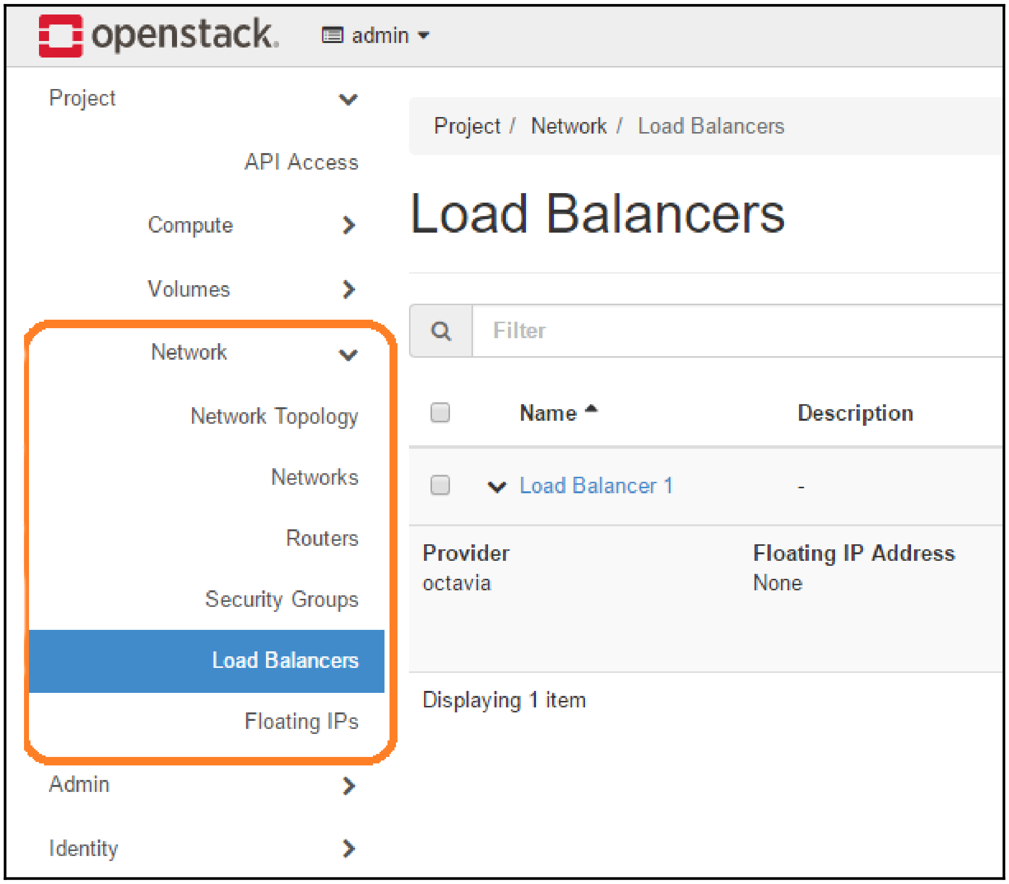
During the earlier release of OpenStack, there was no dedicated project in OpenStack to handle the networking functionality. Instead, the subproject was embedded in the compute project called nova-network (legacy networking) that takes care of managing the basic networking in the virtual compute environment. However, the features of legacy networking are limited, which has no extended functionality to support creating complex network topologies and project-specific networking elements in OpenStack. To address these limitations, a new, dedicated networking project that provides advanced networking functionality to a virtual computing environment was added in OpenStack and code named as OpenStack neutron.

Notably, the legacy nova-network has been said to be depreciated for a while now. However, there are still environments out there that use legacy nova-network in a production environment. It is mainly due to certain use cases that match legacy nova-network better than neutron.

To be precise, using legacy nova-network would be like using Internet Explorer, just because you happen to access the website that only supports IE browser. On the other hand, using neutron would be like using the Chrome browser that supports long listing add-ons to have extended functionality.

OpenStack Neutron

From the Folsom OpenStack release,the dedicated project focused on delivering the networking-as-a-service in a virtual computing environment is added in an OpenStack, code named as a neutron. The OpenStack neutron project provides an API that lets the user define network configuration in the OpenStack cloud environment. The neutron service allows the OpenStack administrator to integrate different networking technologies to power the OpenStack cloud networking. The neutron service also offers extended networking functionality that includes virtual routers, **Network Address Translation** (**NAT**), load balancing, firewall-as-a-service, and virtual private networks:



Load Balancers

Name	Description
Load Balancer 1	-

Provider	Floating IP Address
octavia	None

Displaying 1 item

In `Chapter 3`, *Day 3 -Field Sketch*, you have learned the sub-components of neutron services and the interconnections in detail. Here, I have focused on explaining the most commonly used neutron functionalities, plugins, and the network traffic flows.

Network types

The network types in OpenStack neutron is broadly classified into two types:

- Provider networks
- Self-service networks

Provider networks

Provider networks connect to the existing layer-2 physical networks or VLANs in the datacenter. The *OpenStack* user with a member role cannot create or configure the network topologies in the provider network. The provider network can only be set up by an *OpenStack* admin who has access to manage the actual physical networks in the datacenter. This is because configuring the provider network requires configuration changes in the physical network infrastructure level.

Self-service networks

Self-service networks enable the users with member role to create their own network topology within the user's tenant/project without involving OpenStack administrator's help. The users can create as many virtual networks as they like (not exceeding the project quota limit), interconnected with virtual routers and connected to an external network. Also, self-service networks offer VxLAN/GRE encapsulation protocols; in that way, the user can create a private tunnel within your OpenStack nodes. This will allow the user to set up their own subnets and can be linked to the external or provider networks using an OpenStack router.

By default, the self-service networks created in any specific tenant/project is entirely isolated and are not shared with other tenants in *OpenStack*. With the help of network isolation and overlay technologies, the user can create multiple private(self-service) networks in the same tenant and can also define their own subnets, even if that subnet ranges overlap with the subnet of another tenant network topology.

Types of network isolation and overlay technologies

A **local network** is a network type in OpenStack that can only be used on a single (all-in-one) host OpenStack setup. This local network type is appropriate for the proof-of-concept or development environments.

A **flat network** is a network type in OpenStack that offers every VMs in the entire OpenStack setup to shares the same network segment. To be precise, a typical L2 ethernet network is a *flat* network that allows servers attached to this network to see the same broadcast traffic, and they can contact each other without requiring a router.

For example, let's take, two tenants (tenant A and B) from the OpenStack setup configured in flat network type with the subnet range `192.168.1.0/24`. In this network type, VM1 from tenant 1 may get assigned to IP `192.168.1.5`, VM1 from tenant 2 may get `192.168.1.6`, and so on. This means that the flat network type allows the tenant A to see the traffic from tenant B and has no isolation between projects.

In most cases, the flat networks are preferred when configuring the provider network.

A **VLAN network** type in OpenStack neutron uses VLANs for segmentation. Whenever a user creates a new network, the neutron will assign a unique VLAN ID to each new network from the segmentation range we have configured in the neutron configuration file. To allow the traffic flow outside the host, the network administrator needs to manually configure the physical switches in the data centre with the same VLAN ID tag. Thus, using VLAN network type would require a lot of manual inputs in managing the underlying physical network environment.

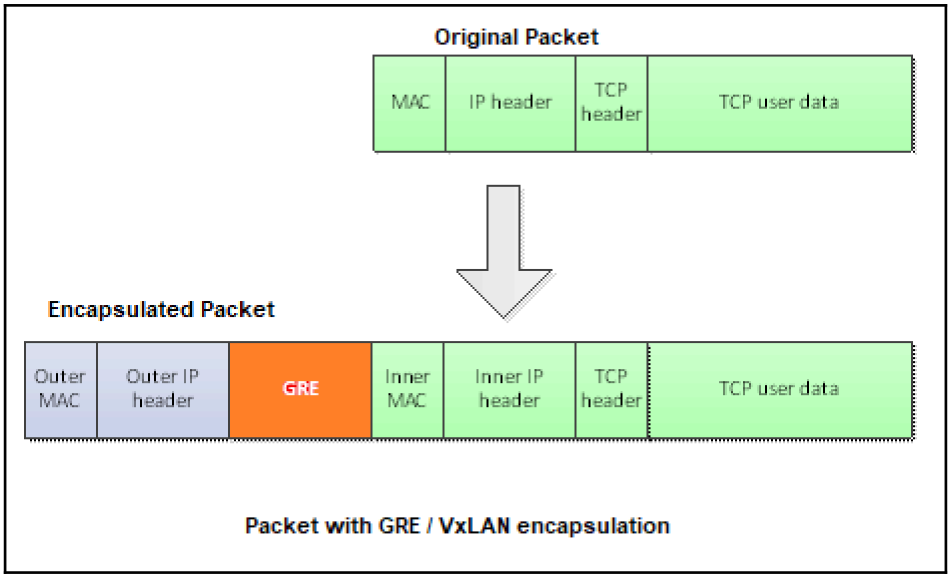
However, unlike flat network type, the VLAN network will provide the traffic isolation between the tenants. With the help of the VLAN isolation techniques, the tenant can specify the same subnet range across different tenants.

For example, VM 01 from tenant A can get IP `192.168.1.5` and VM 01 from tenant B can also get IP `192.168.1.5` without any conflicts. Thus, network administrators need not worry about the users who create the existing subnet range, as the VLANs keep them separate.

GRE and VxLAN networks are the most commonly used network type in the OpenStack neutron.They both work by encapsulating the network traffic to create overlay networks. Like VLAN networks, whenever the user creates a new network, the neutron assigns a unique tunnel ID. However, an overlay network does not require any configuration changes to the underlying physical switch environment that will eliminate the manual interaction.

With the help of the overlay technologies, GRE and VxLAN segmentation provides complete isolation between the tenants and offers overlapping subnets and IP ranges. It does this by encapsulating the VM traffic in tunnels.

For example, let's assume that a tenant (A) has three VMs (VM01, VM02, and VM03) running on the compute node *n1*, *n2*, and *n3* respectively. Using VxLAN / GRE network type in neutron will create a fully connected mesh of tunnels among all of the three compute nodes for communication between the VMs. If a VM 01 on compute node n1 wants to send packets to the VM 02 on compute node n2, then node n1 will encapsulate the IP packets coming out of the VM 01 using a segmentation ID that was generated during the network creation and transmitting on the network pointing the compute node *n2* as the destination address. At the receiving end, the compute node *n2* receives the encapsulated packet and will decapsulate the packets and then route them to the target VM 02:



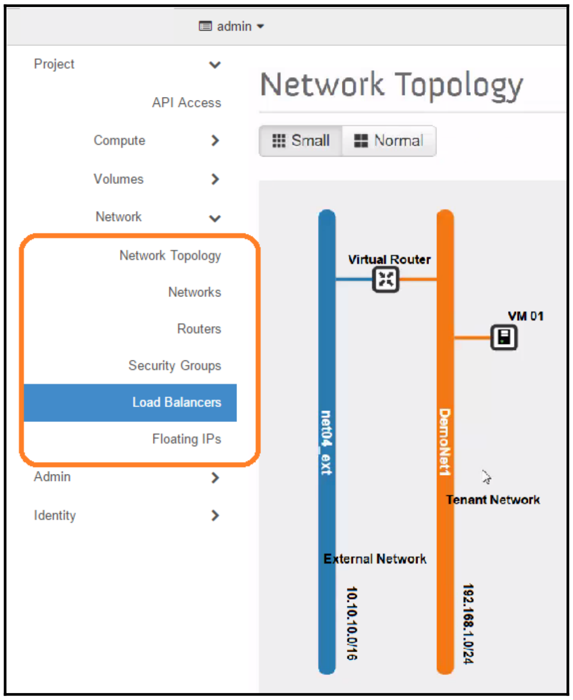
Why VxLAN?

There are two reasons why to use VxLAN:

- It increases scalability in virtualized cloud environments as VxLAN ID enables you to create up to 16 million isolated networks. This overcomes the limitation of VLANs having VLAN ID that allows you to create a maximum of 4094 private networks.
- No configuration changes are required in the physical network devices.

Neutron offerings

As I mentioned earlier, OpenStack neutron covers the nadir to the zenith of advanced networking techniques. The OpenStack neutron, a dedicated project for networking in OpenStack to serve the Networking-as-a-Service feature in the virtual compute environment, has many extended features over the legacy nova-network. Here, I have listed the most commonly available features of the vanilla OpenStack neutron:



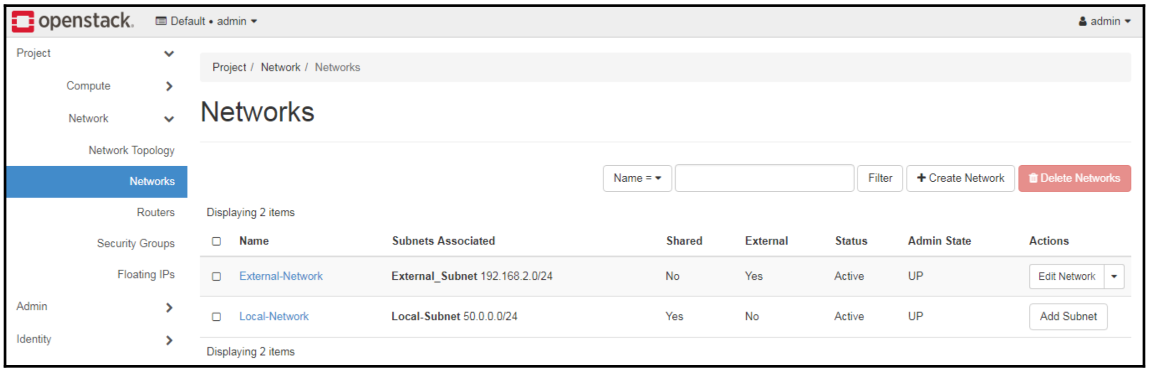
Network topology

From the preceding figure, you can see that the very first option under the **Network** panel (left side) is network topology. Neutron has this topology feature to illustrate the overlay network for the end users to visualize the networking layout for their virtual computing environment.

The network topology diagram in the preceding figure was depreciated in the latest OpenStack release, replacing with a new attractive UX model. However, personally, I feel this depreciated UX for network topology helps the beginner to understand the network layout a lot better than the latest one.

Networks and subnets

Using a VxLAN/GRE network type in the OpenStack neutron will enable the user to create their own subnets that can be connected to the external/provider networks using an OpenStack router. The following figure shows the network administration options available in the **Network** panel:



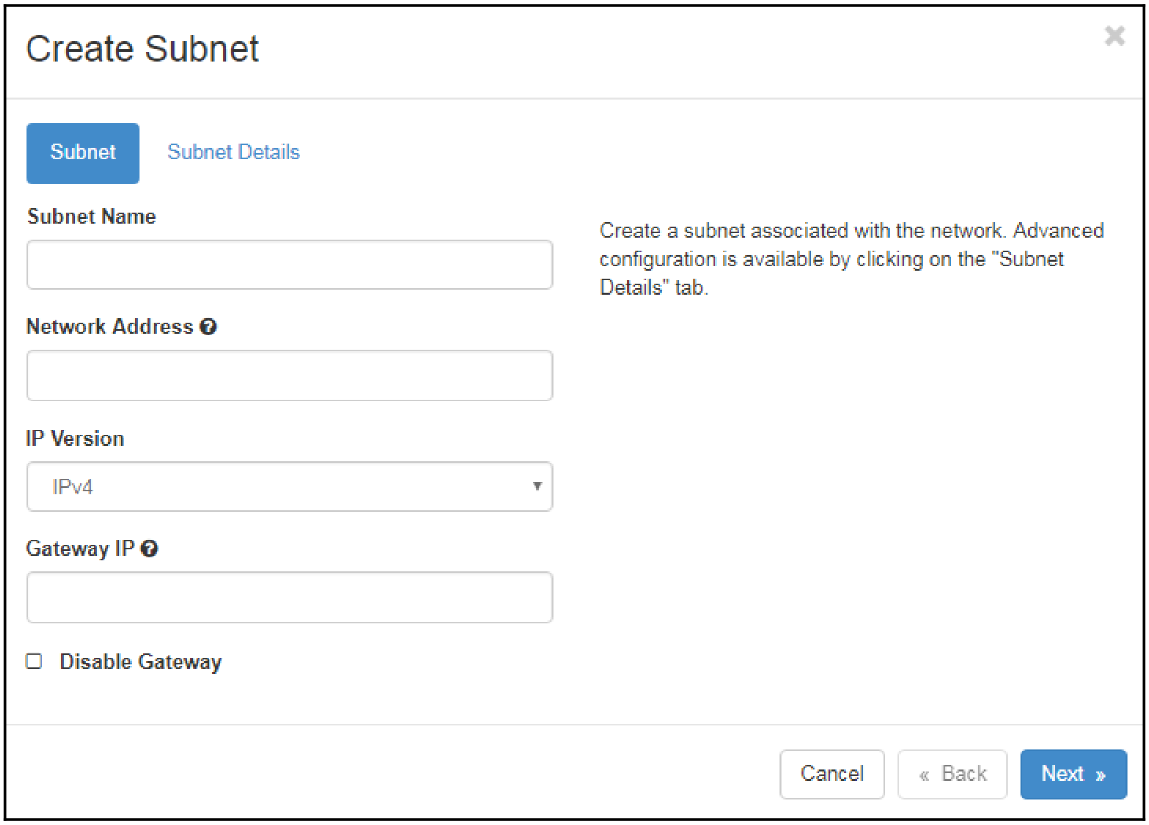
Networks

Displaying 2 items

Name	Subnets Associated	Shared	External	Status	Admin State	Actions
External-Network	**External_Subnet** 192.168.2.0/24	No	Yes	Active	UP	Edit Network
Local-Network	**Local-Subnet** 50.0.0.0/24	Yes	No	Active	UP	Add Subnet

Displaying 2 items

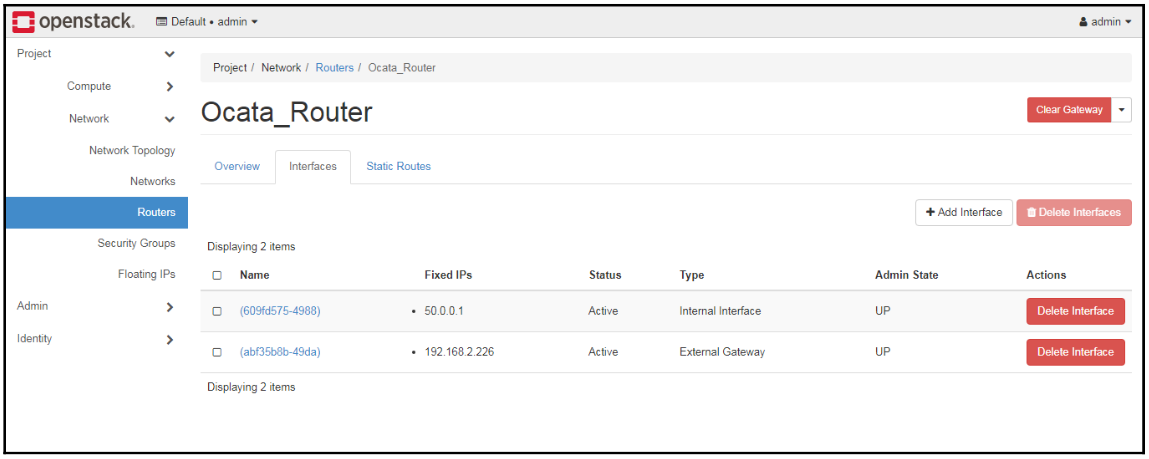
The **Networks** tab will provide the user with an option to create new networks. Typically, creating a new network in an OpenStack neutron is very similar to creating a new switch in a virtual environment. The user can create any number of networks without exceeding the network quota limit for the specific project. The following figure shows the options available for creating a subnet for the network:



Create a subnet associated with the network. Advanced configuration is available by clicking on the "Subnet Details" tab.

Routers

Routers enable the virtual layer-3 functionality, such as routing and **Network Address Translation** (**NAT**) between self-service (tenant network) and provider networks, or even between two different self-service networks belonging to the same project. The following figure provides the information about the router's interface mappings and the router's gateway:

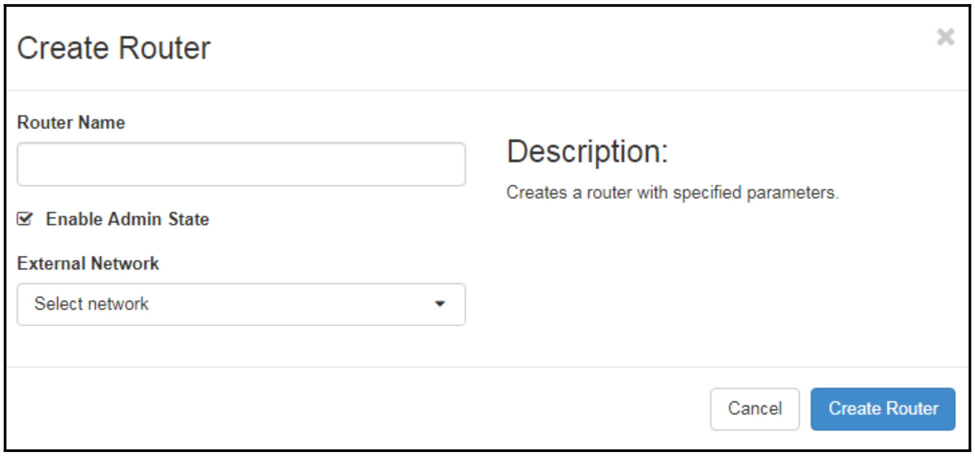


Displaying 2 items

Name	Fixed IPs	Status	Type	Admin State	Actions
(609fd575-4988)	• 50.0.0.1	Active	Internal Interface	UP	Delete Interface
(abf35b8b-49da)	• 192.168.2.226	Active	External Gateway	UP	Delete Interface

Displaying 2 items

Like network creation, the OpenStack user can create his own router within the project without exceeding the router quota limit. As you can see from the following figure, the router creation process does not require many inputs. Beneath, the OpenStack networking service uses a layer-3 agent to manage all routers via namespaces:

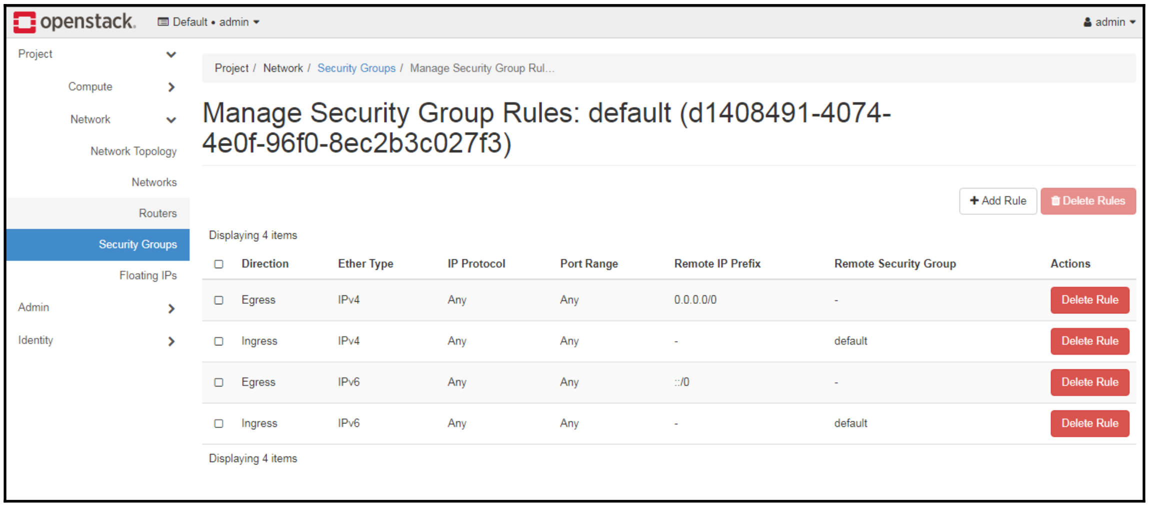


Creates a router with specified parameters.

Security groups

Security groups provide virtual firewall rules that control inbound and outbound network traffic at the port level. Underneath, creating and managing the security group will write a proper IP tables rule in the compute node.

By default, each tenant has a default security group that allows all egress (outbound) traffic and denies all ingress (inbound) traffic. The default security group will be added to all virtual machines as a default setting. So, we need to change the rules in the default security group to allow inbound traffic to the virtual machines. Optionally, we could also create and manage a dedicated security group for each virtual machine based on our use cases:



Manage Security Group Rules: default (d1408491-4074-4e0f-96f0-8ec2b3c027f3)

Displaying 4 items

Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Actions
Egress	IPv4	Any	Any	0.0.0.0/0	-	Delete Rule
Ingress	IPv4	Any	Any	-	default	Delete Rule
Egress	IPv6	Any	Any	::/0	-	Delete Rule
Ingress	IPv6	Any	Any	-	default	Delete Rule

Displaying 4 items

Extended services

Adding to the above listed functionalities, OpenStack neutron also offers the extended features.

VPNaaS

The **Virtual Private Network-as-a-Service** (**VPNaaS**) is a neutron extension that introduces the VPN functionality to the OpenStack networking.

LBaaS

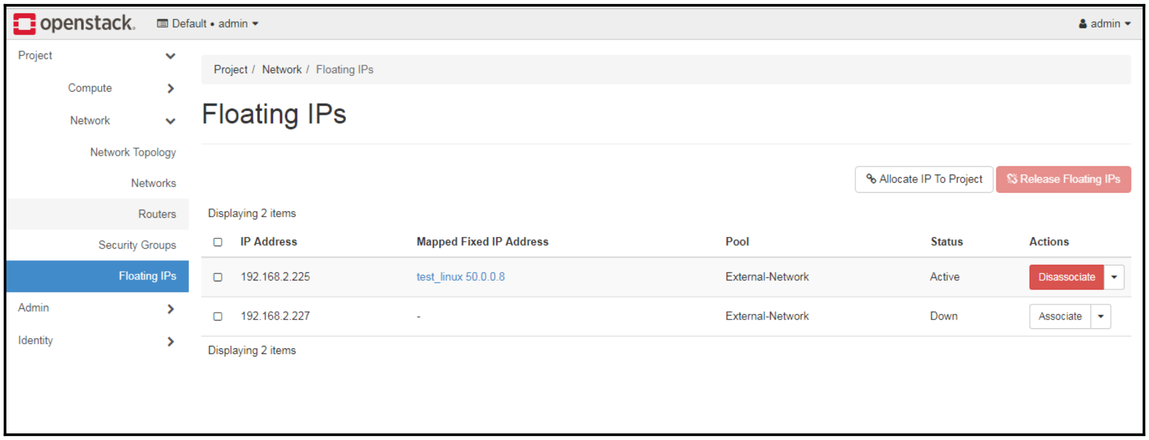
The **Load-Balancer-as-a-Service** (**LBaaS**) is a neutron extended feature to enable load balancing in the virtual computing environment. Beneath, the neutron uses the HA Proxy software for implementing load balancer functionality.

FWaaS

The **Firewall-as-a-Service** (**FWaaS**) functionality in a neutron is in an experimental state that enables users to configure the firewall rules and a firewall policy that contains match conditions and a reactive action to take (allow or deny) on matched traffic.

Floating IP

Floating IP in OpenStack networking is nothing but the IP range from the OpenStack external network to NAT with the virtual machines private IP. When we create a new virtual machine in the private network, the VM will receive the private IP from the DHCP agent of the private network. To access the virtual machine from the external network, we need to SNAT the external network IP with the VM's private IP. The **Floating IPs** tab under the **Network** panel provides the options for NAT mapping:



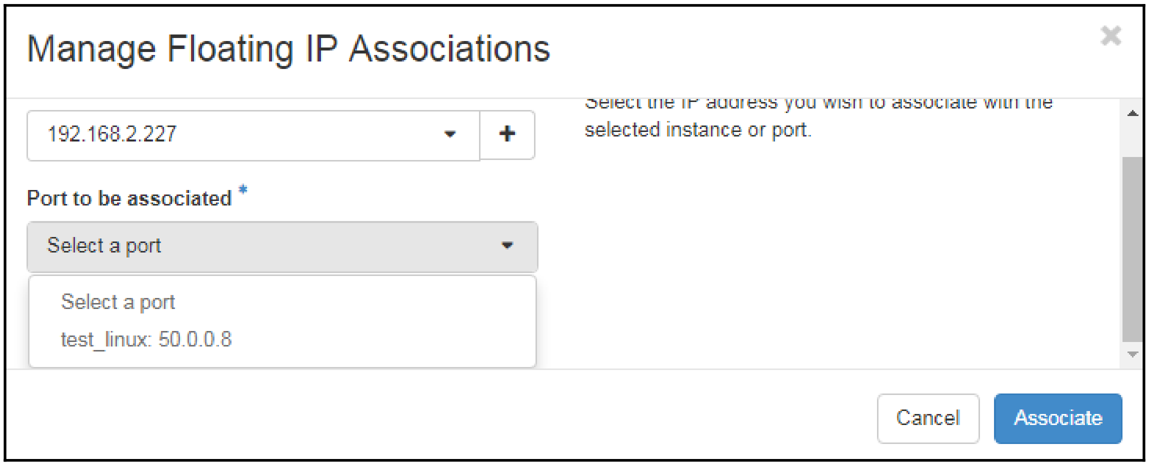
Floating IPs

Displaying 2 items

IP Address	Mapped Fixed IP Address	Pool	Status	Actions
192.168.2.225	test_linux 50.0.0.8	External-Network	Active	Disassociate
192.168.2.227	-	External-Network	Down	Associate

Displaying 2 items

The following figure shows the available options for NAT mapping in the OpenStack neutron. The router enables the functionality to connect the instances directly from an external network using the floating IP addresses. Thus, the floating IP association with private IP is functional only if the external network and the private network are linked with the common router:


Manage Floating IP Associations

192.168.2.227

Port to be associated *

Select a port

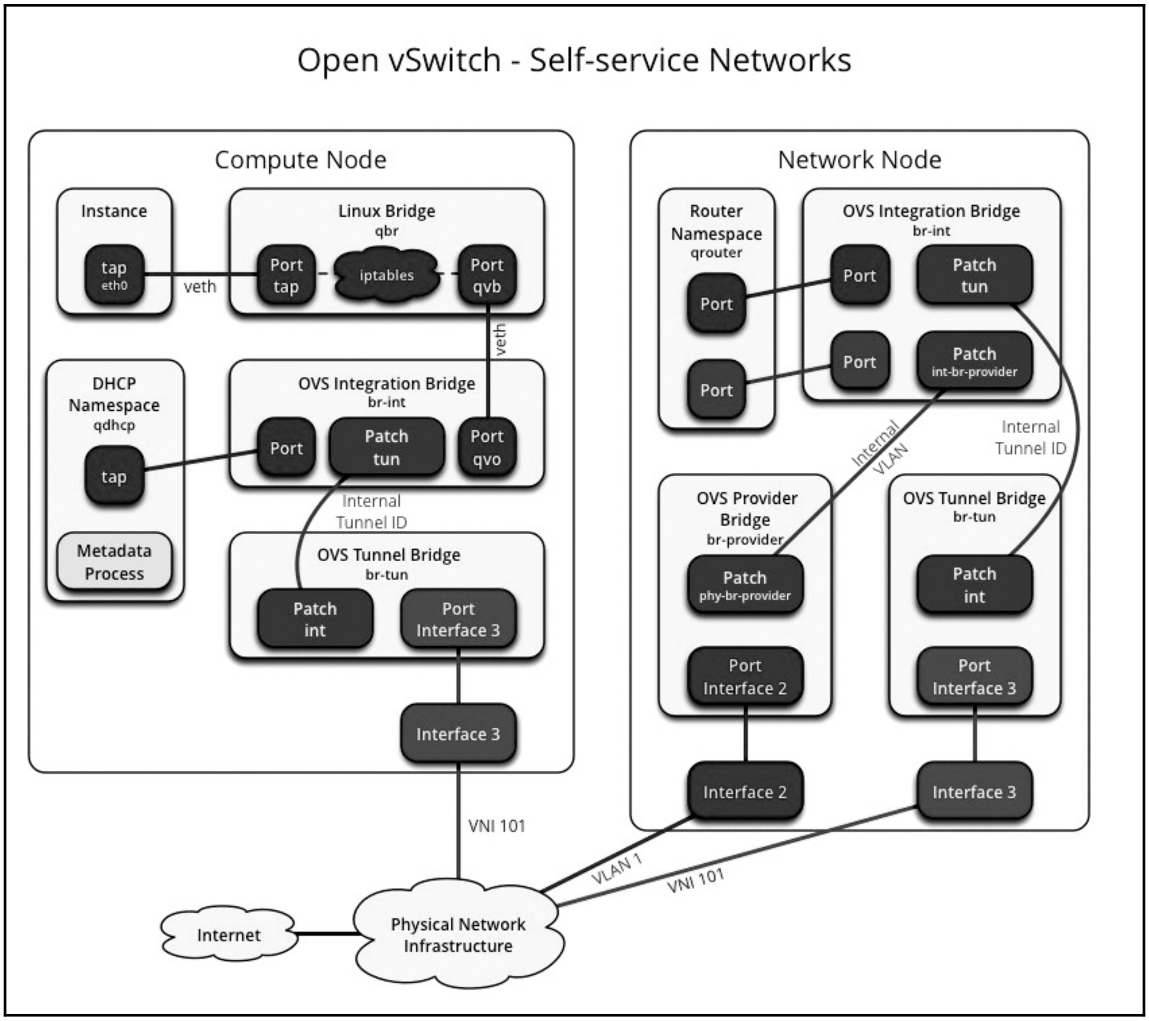
Select a port

test_linux: 50.0.0.8

Select the IP address you wish to associate with the selected instance or port.

Network traffic flow

To understand the neutron clearly, one must know the clear picture of how neutron manages the packets flow in OpenStack. The following figure shows the components and connectivity of self-service network, using the Open vSwitch plugin:



By referring to the preceding figure, we will see the flow of network traffic in the following scenarios:

- **North-south network traffic**: Travels between an instance and external network
- **East-west network traffic**: Travels between instances

Before that, take a note of the following keywords:

TAP device, such as `vnet0` is a virtual device used by the hypervisors to implement a virtual network interface card, most commonly known as VIF or vNIC. The virtual machine will process an ethernet frame received by a TAP device attached.

A **veth pair** is a virtual cable that directly connects the virtual network interfaces. It is more like a LAN cable we use in real life to connect a PC and switch. An ethernet frame which goes in one end will come out on the other end of a veth pair.

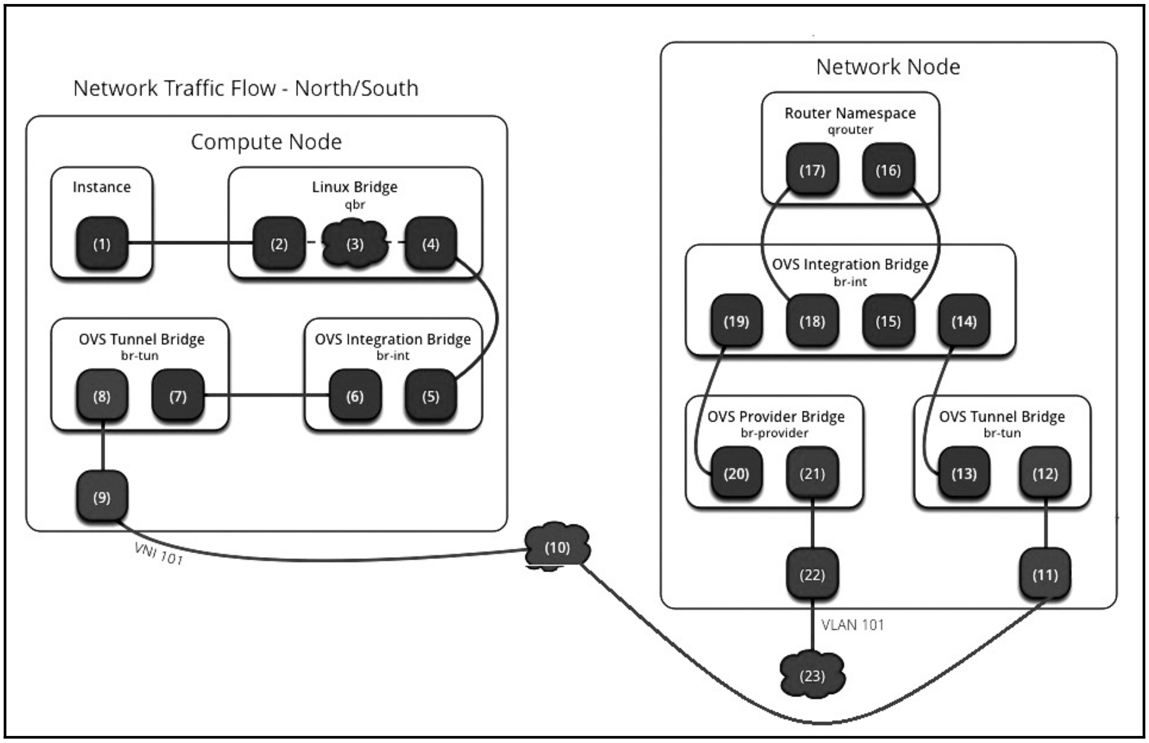
A **Linux bridge** is more like a simple, unmanaged L2 switch. We can connect multiple (physical or virtual) network interfaces devices to a Linux bridge.

An **Open vSwitch bridge** is more like a manageable multi-layer switch. We can attach the network interface devices to an Open vSwitch bridge's ports and the ports can be configured with VLAN. Notably, despite being an open source software, OVS supports most advanced networking technologies such as NetFlow, sFlow, and OpenFlow that can be seen in the latest proprietary switches.

North-south network traffic

Let's assume the following scenario:

- The instance is in compute node 01 and uses the self-service network
- The instance sends a packet to the internet (say, ping `www.hellovinoth.com`)



On compute node:

The following packet flow will take place on the compute node:

1. The virtual machine's virtual network interface (vNIC) **(1)** forwards the packet to the security group bridge (**qbr**) port **(2)** through the veth pair.
2. The security group rules applied **(3)** to the **Linux Bridge (qbr)** that handles the packet filtering using IP tables rules applied to it using the security group mapped to the instance.
3. The security group bridge's port **(4)** forwards the packet to the **OVS Integration Bridge (br-int)** on security group port **(5)** via veth pair.
4. The **OVS Integration Bridge (br-int)** adds an internal VLAN tag to the packet.
5. The OVS integration bridge patch port **(6)** forwards the packet to the **OVS Tunnel Bridge (br-tun)** patch port **(7)**.

6. The **OVS Tunnel Bridge (br-tun)** then modifies the assigned internal VLAN tag with a segmentation tunnel ID and stores the table record for VLAN and the corresponding tunnel ID is mapped.
7. The **OVS Tunnel Bridge (br-tun)** port **(8)** encapsulates the packet with the compute node's ethernet frame, that is, the source MAC address as a compute node MAC and destination MAC as a network node.
8. The underlying physical interface **(9)** for the overlay networks forwards the packet to the network node through the overlay network **(10)**.

On network node:

The following packet flow will take place on the network node:

1. The physical interface **(11)** for the overlay networks sends the packet to the tunnel bridge (br-tun) port **(12)**.
2. The **OVS Tunnel Bridge (br-tun)** decapsulates the packet.
3. The **OVS Tunnel Bridge (br-tun)** modifies the segmentation tunnel ID with an internal VLAN tag by fetching the table record for VLAN ID and the respective tunnel ID mapped.
4. The **OVS Tunnel Bridge (br-tun)** patch port **(13)** forwards the packet to the **OVS Tunnel Bridge (br-tun)** patch port **(14)**.
5. The **OVS Integration Bridge (br-int)** port for the self-service (tenant) network **(15)** removes the internal VLAN tag and forwards the packet to the tenant network interface **(16)** in the router namespace.
6. Then, the router performs NAT on the packet, which changes the source IP address of the packet to the router IP address and sends it to the gateway of the provider network through the router's gateway **(17)** port.
7. The router redirects the packet to the **OVS Integration Bridge (br-tun)** port for the provider network **(18)**.
8. The OVS integration bridge's patch port (int-br-provider) **(19)** forwards the packet to the OVS provider bridge's patch port (phy-br-provider) **(20)**.
9. The OVS provider bridge (br-provider) encapsulates the packet with network node's ethernet frame.
10. The OVS provider bridge (br-provider) port **(21)** forwards the packet to the physical network interface **(22)** as an ordinary packet (Jumbo frame).
11. The physical network interface redirects the packet to the internet through the physical network infrastructure **(23)**.

East-west network traffic

Now, let's see, the traffic flow between two virtual machines on two different networks.

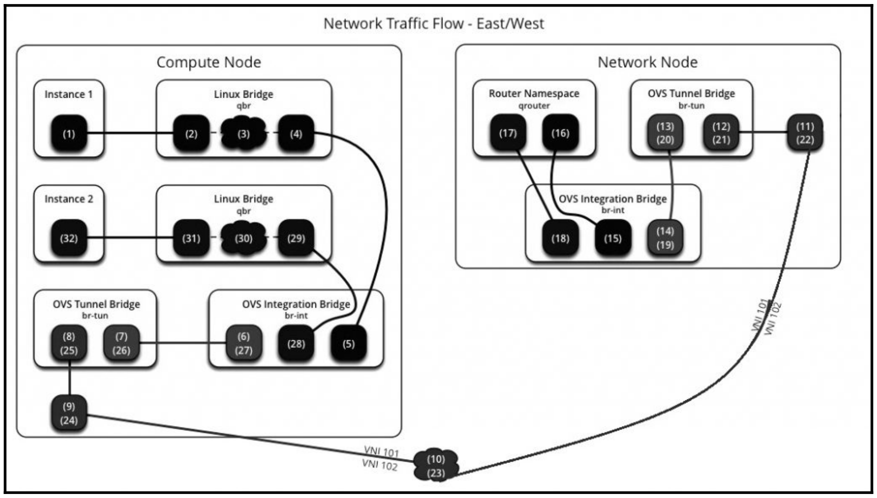
Scenario:

- Instance 01 is in compute node 01 and uses self-service network 01
- Instance 02 is located in compute node 01 and uses self-service network 02
- VM 1 on compute node 01 sends a packet to VM 2, which resides on the same compute node, 01

On compute node:

The following packet flow will take place on the compute node:

1. The virtual machine's virtual network interface (vNIC) **(1)** forwards the packet to the security group **Linux Bridge (qbr)** port **(2)** through the veth pair.
2. The security group rules applied (3) to the **Linux Bridge (qbr)** that handles the packet filtering using IP tables rules applied using the security group mapped to the instance.
3. The security group bridge's port **(4)** forwards the packet to the **OVS Integration Bridge (br-int)** on security group port **(5)** via veth pair.
4. The **OVS Integration Bridge (br-int)** adds an internal VLAN tag to the packet.
5. The **OVS Integration Bridge (br-int)** patch port **(6)** forwards the packet to the **OVS Tunnel Bridge (br-tun)** patch **port (7)**:



6. The **OVS Tunnel Bridge (br-tun)** then modifies the assigned internal VLAN tag with a segmentation tunnel ID and stores the table record for VLAN and the corresponding tunnel ID is mapped.
7. The **OVS Tunnel Bridge (br-tun)** port **(8)** encapsulates the packet with the compute node's ethernet frame, that is, the source MAC address as compute node MAC and the destination MAC as a network node.
8. The underlying physical interface **(9)** for overlay networks directs the packet to the network node through the overlay network link **(10)**.

On network node:

The following packet flow will take place on the network node:

1. The physical interface **(11)** of a network node for overlay networks forwards the packet to the **OVS Tunnel Bridge (br-tun)** port **(12)**.
2. The **OVS Tunnel Bridge (br-tun)** decapsulates the packet.
3. The **OVS Tunnel Bridge (br-tun)** modifies the VxLAN segmentation tunnel ID with an internal VLAN tag by fetching the table record for VLAN ID and the respective tunnel ID is mapped.
4. The **OVS Tunnel Bridge (br-tun)** patch-int patch port **(13)** forwards the packet to the **OVS Tunnel Bridge (br-tun)** patch-tun patch port **(14)**.
5. The OVS integration bridge port for tenant network 1 (15) removes the internal VLAN tag and redirects the packet to the tenant network 01 interfaces **(16)** in the router namespace.
6. The router sends the packet to the next-hop IP address, typically the gateway IP address of tenant network 2, through the tenant network 2 interfaces **(17)**.
7. The router forwards the packet to the **OVS Integration Bridge (br-int)** port for tenant network 2 **(18)**.
8. The **OVS Integration Bridge (br-int)** adds the internal VLAN tag to the packet.
9. The OVS Integration Bridge interchanges the inner VLAN tag for a VxLAN segmentation tunnel ID.
10. The **OVS Integration Bridge (br-int)** patch-tun patch port **(19)** forwards the packet to the **OVS Tunnel Bridge (br-tun)** patch-int patch port **(20)**.
11. The **OVS Tunnel Bridge (br-tun)** port **(21)** encapsulates the packet with network node's ethernet frame.
12. The underlying physical interface **(22)** for overlay networks forwards the packet to the compute node through the overlay network **(23)**.

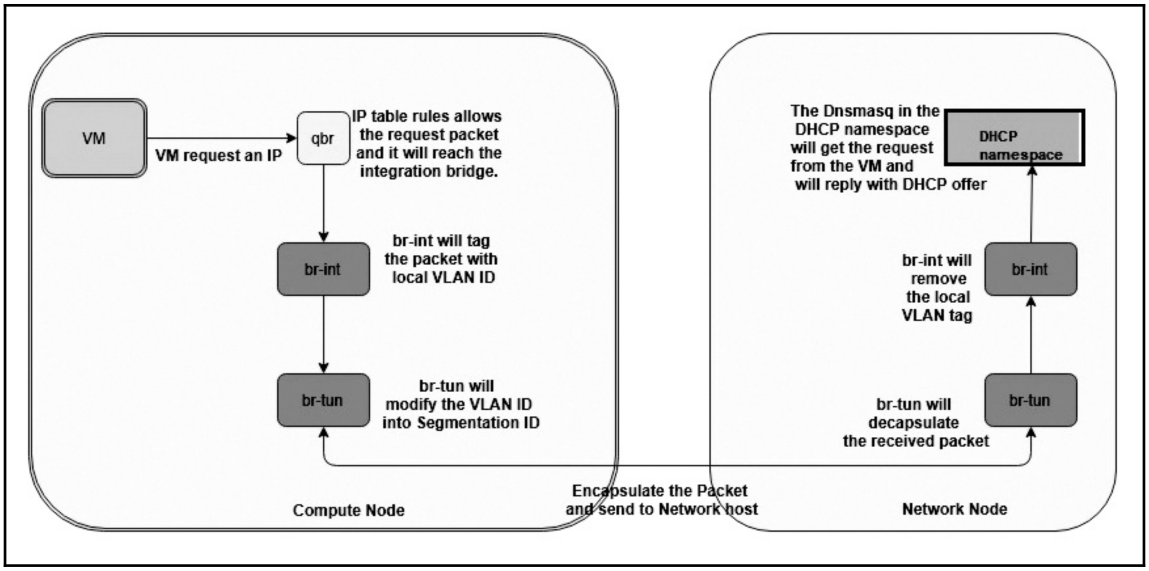
On computing node:

The following packet flow will take place on the compute node:

1. The packet forwarded from network node will receive at the compute node. Then, the original physical interface card **(24)** for overlay networks redirects the packet to the **OVS Tunnel Bridge (br-tun) (25)**.
2. The **OVS Tunnel Bridge (br-tun)** decapsulatesthe packet.
3. The OVS tunnel bridge interchanges the VxLAN segmentation tunnel ID for an internal VLAN tag.
4. The **OVS Tunnel Bridge (br-tun)** patch-int patch port **(26)** forwards the packet to the **OVS Integration Bridge (br-int)** patch-tun patch port **(27)**.
5. The **OVS Integration Bridge (br-int)** removes the internal VLAN tag from the packet.
6. The **OVS Integration Bridge (br-int)** security group port **(28)** forwards the packet to the security group bridge (qbr) OVS port **(29)** through the veth pair.
7. Security group rules **(30)** on the **Linux Bridge (qbr)** that handles the packet filtering using IP tables rules applied to it using the security group mapped to the instance.
8. The security group bridge (qbr) instance port **(31)** forwards the packet to the virtual machine's interface **(32)** through the veth pair.

How does a VM get an IP?

The DHCP agent interconnects with the neutron-server over RPC. Every single network in OpenStack has its own DHCP namespace. With the help of namespaces, the DHCP agent ensures the complete network isolation between other networks. Each DHCP namespace has the `dnsmasq` process running and it takes cares of serving the DHCP parameters, such as IP address and netmask:


VM
VM request an IP
qbr
IP table rules allows the request packet and it will reach the integration bridge.
br-int
br-int will tag the packet with local VLAN ID
br-tun
br-tun will modify the VLAN ID into Segmentation ID
Compute Node
Encapsulate the Packet and send to Network host
The Dnsmasq in the DHCP namespace will get the request from the VM and will reply with DHCP offer
DHCP namespace
br-int will remove the local VLAN tag
br-int
br-tun will decapsulate the received packet
br-tun
Network Node

The preceding figure is self-explanatory, which will explain - how does a virtual machine receive the network information from the DHCP agent? It is important to understand such traffic flows in OpenStack that could greatly help us during the troubleshooting process.

Summary

In this chapter, we have seen the OpenStack networking in detail. I believe that the reader will have gained in-depth knowledge of OpenStack neutron and its interesting functionality. The step-by-step walkthrough on *How the networking traffic flow works* will help the readers to understand the neutron better and will contribute to troubleshooting in OpenStack.

In the next chapter, we will have the hands-on training exercise on how to use the OpenStack horizon for using all of the OpenStack core components.

6
Day 6 - Field Training Exercise

The wait is over! So far, we have come across more of the theoretical content. Now, it is time to unleash the power of learning by doing. In this chapter, we will have the hands-on lab exercises on OpenStack, which covers all of the critical skills needed to administrate the OpenStack cloud.

In the upcoming training session, you will be executing a series of hands-on labs exercises for each component of OpenStack that will help you in understanding the OpenStack architecture by doing them.

Before getting our hands dirty in OpenStack, let's ensure the prerequisites for the field training exercise:

- The all-in-one Openstack cluster, which we built in `Chapter 1`, *Day 1 - Build Your Camp*
- To access the OpenStack dashboard (Horizon), you need to have Google Chrome installed
- To access the OpenStack via CLI, you need to have the SSH client software installed:
    - For Windows OS use the PuTTy SSH client
    - For Linux/OS X use the command-line Terminal

Understanding the OpenStack lab environment

As a first step, let's explore and understand how the lab environment is configured, which will help you with the upcoming hands-on exercise:

Goal	Getting ready for the lab environment both command-line and UI
Activities	• Explore Lab through CLI • Explore Lab through Web UI

Exploring lab through the command line

In this section, we will walk through how to navigate and explore our lab environment:

1. From your workstation using PuTTy or Terminal, SSH to the virtual machine or server where we have installed all-in-one OpenStack setup using DevStack on Day 1. Please go back to `Chapter 1`, *Day 1 - Build Your Camp* and recap the lab environment setup.
2. Log in using the stack/stack username and password:

```
ssh stack@192.168.56.101
#Command will prompt for password
```

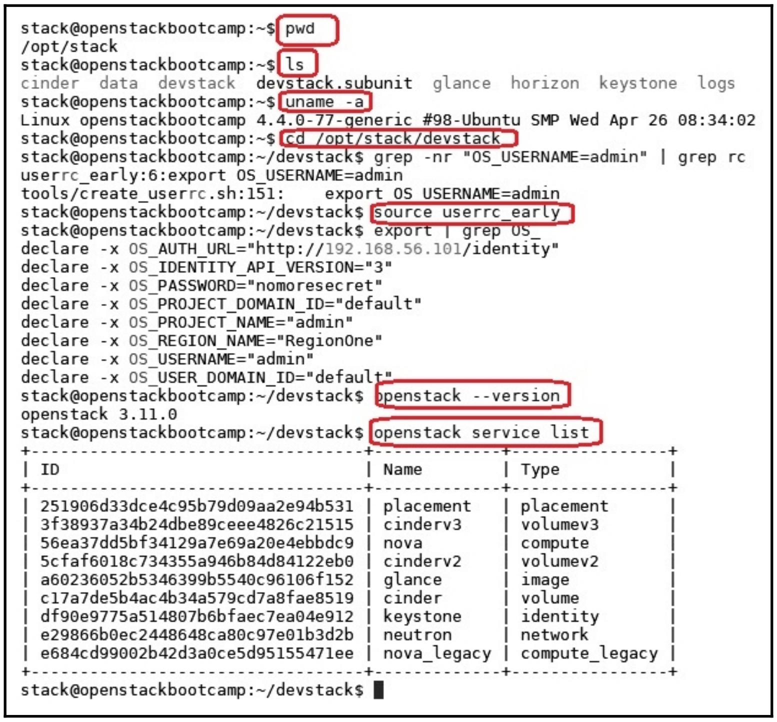
3. Let's execute the following commands to verify your OpenStack server's basic info:

```
hostname
#The output will display your hostname. In my case, the hostname of
my server is "openstackbootcamp".
pwd
#The output will write the full pathname of the current working
directory.
ls
#This command writes the list of files and directories in your
working directory.
uname -a
#This command writes the Linux build we are currently running on
our system.
cd /opt/stack/devstack
#This command will take you to the devstack directory from your
current home directory.
source userrc_early
#This step will Source the environment file to set environment
variables for your current shell to execute Openstack commands.
grep -nr "OS_USERNAME=admin" | grep rc
# Note that the environment file name may vary sometime. The above
command would help you in finding the right one.
export | grep OS_
#You can verify if the environment variables are set by executing
the above command.
```

Whenever you open the new command-line window in your workstation, you need to reconnect the SSH session and source the environment file again.

```
openstack --version
#Returns the version number for Openstack command-line client.
openstack service list
# Will return the list of servers configured as part of the
Openstack cluster.
openstack --help
#The above command will output the long listing available Openstack
CLI.
```

You may also refer to the following screenshot for the output of the commands mentioned previously:



```
stack@openstackbootcamp:~$ pwd
/opt/stack
stack@openstackbootcamp:~$ ls
cinder  data  devstack  devstack.subunit  glance  horizon  keystone  logs
stack@openstackbootcamp:~$ uname -a
Linux openstackbootcamp 4.4.0-77-generic #98-Ubuntu SMP Wed Apr 26 08:34:02
stack@openstackbootcamp:~$ cd /opt/stack/devstack
stack@openstackbootcamp:~/devstack$ grep -nr "OS_USERNAME=admin" | grep rc
userrc_early:6:export OS_USERNAME=admin
tools/create_userrc.sh:151:    export OS_USERNAME=admin
stack@openstackbootcamp:~/devstack$ source userrc_early
stack@openstackbootcamp:~/devstack$ export | grep OS_
declare -x OS_AUTH_URL="http://192.168.56.101/identity"
declare -x OS_IDENTITY_API_VERSION="3"
declare -x OS_PASSWORD="nomoresecret"
declare -x OS_PROJECT_DOMAIN_ID="default"
declare -x OS_PROJECT_NAME="admin"
declare -x OS_REGION_NAME="RegionOne"
declare -x OS_USERNAME="admin"
declare -x OS_USER_DOMAIN_ID="default"
stack@openstackbootcamp:~/devstack$ openstack --version
openstack 3.11.0
stack@openstackbootcamp:~/devstack$ openstack service list
+----------------------------------+-------------+----------------+
| ID                               | Name        | Type           |
+----------------------------------+-------------+----------------+
| 251906d33dce4c95b79d09aa2e94b531 | placement   | placement      |
| 3f38937a34b24dbe89ceee4826c21515 | cinderv3    | volumev3       |
| 56ea37dd5bf34129a7e69a20e4ebbdc9 | nova        | compute        |
| 5cfaf6018c734355a946b84d84122eb0 | cinderv2    | volumev2       |
| a60236052b5346399b5540c96106f152 | glance      | image          |
| c17a7de5b4ac4b34a579cd7a8fae8519 | cinder      | volume         |
| df90e9775a514807b6bfaec7ea04e912 | keystone    | identity       |
| e29866b0ec2448648ca80c97e01b3d2b | neutron     | network        |
| e684cd99002b42d3a0ce5d95155471ee | nova_legacy | compute_legacy |
+----------------------------------+-------------+----------------+
stack@openstackbootcamp:~/devstack$ 
```

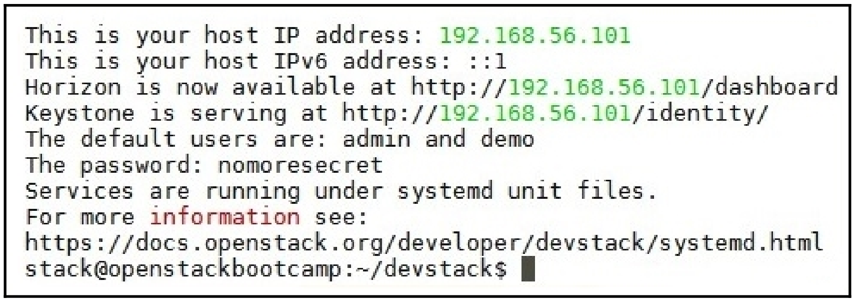
If you could execute all of the preceding commands in your Terminal successfully, congratulate yourself. Now, you are all set to take off.

If you got stuck with any error messages while executing the preceding commands, don't panic. Just jump to `Chapter 9`, *Day 9 - Repair OpenStack*, where I have listed the most common errors and the solutions to fix it.

Exploring lab through the Horizon dashboard

In this section, we will study how to log in to the OpenStack dashboard (Horizon) and how to navigate through the OpenStack services available:

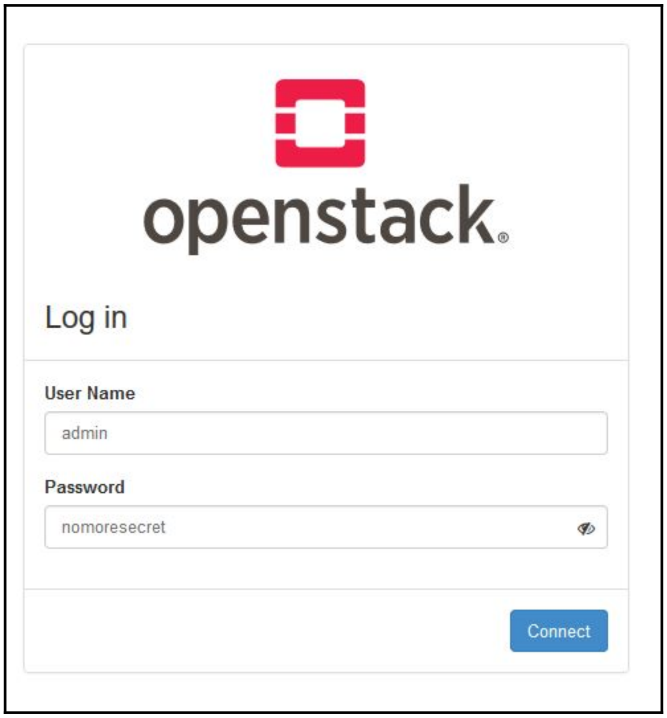
1. Access the OpenStack Horizon via your host machine's browser with the URL and password, which was displayed during your DevStack installation completion output:



```
This is your host IP address: 192.168.56.101
This is your host IPv6 address: ::1
Horizon is now available at http://192.168.56.101/dashboard
Keystone is serving at http://192.168.56.101/identity/
The default users are: admin and demo
The password: nomoresecret
Services are running under systemd unit files.
For more information see:
https://docs.openstack.org/developer/devstack/systemd.html
stack@openstackbootcamp:~/devstack$
```

In my case, I access my Openstack UI at `http://192.168.56.101/dashboard/`, and the password for the **admin** user is `nomoresecret`.

2. Log in to the OpenStack dashboard, using your user credentials:



Log in

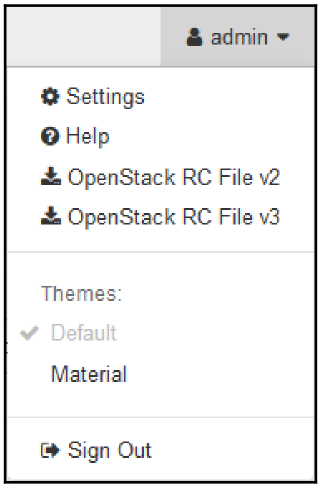
User Name

admin

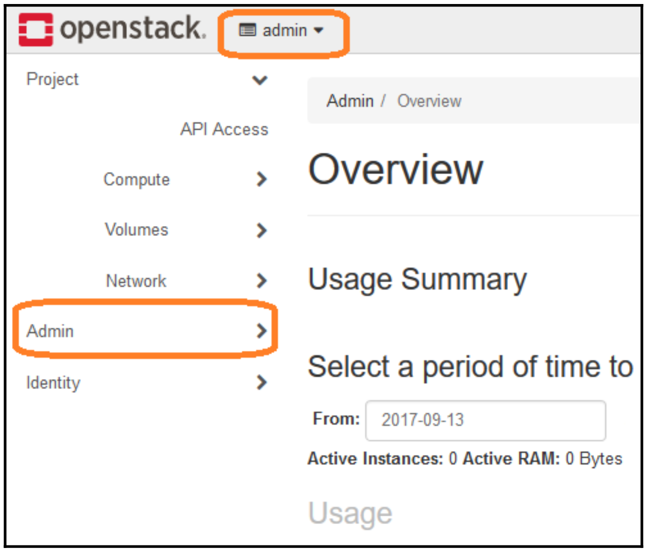
Password

nomoresecret

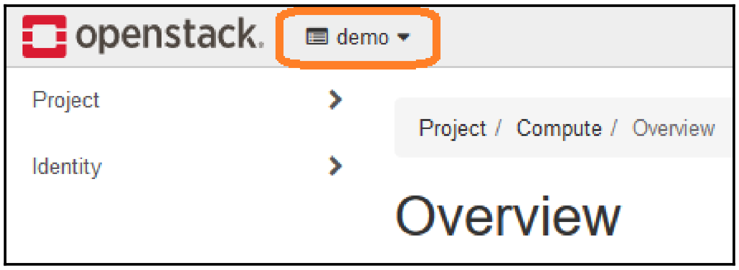
The top-level row shows the username. You could also navigate to the settings menu to find available options on the dashboard:



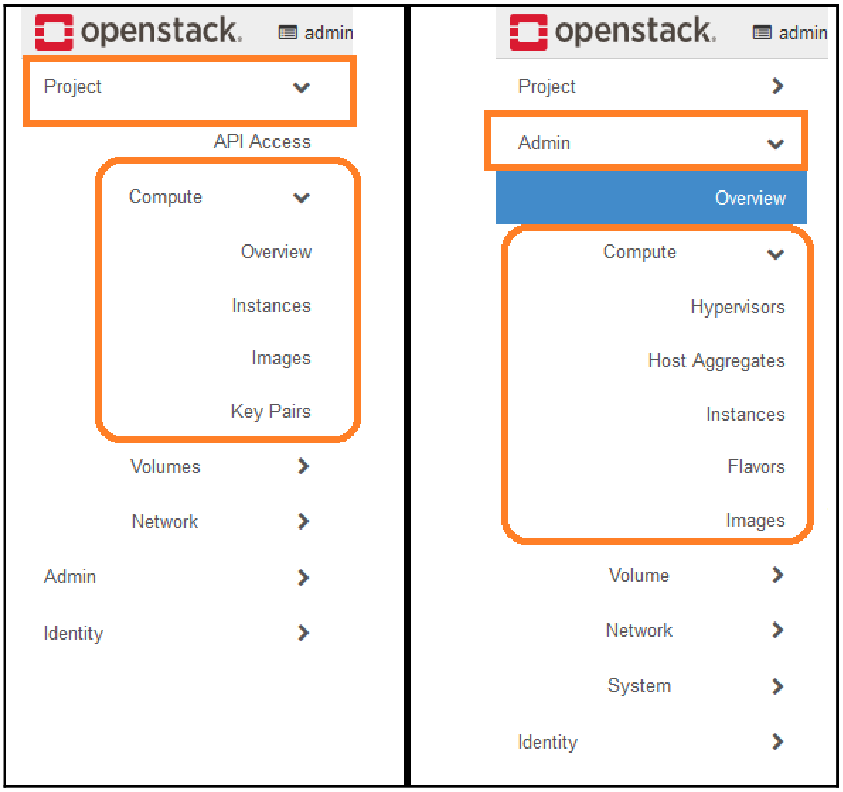
3. The visible view, tabs, and functions on the OpenStack Horizon depends on the roles of the logged in user. In our case, we have logged as an **admin** user with admin role privileges so that you can see all of the available tabs and functions for both the admin and the member role access:



4. If you logged in with the **demo** user account, you would have logged in with end user privileges, so the screen shows only the **Project** and **Identity** tab. It has no role mapped to access the admin functionality:



5. Click each menu link on the left side of the **Project** tab navigation panel to see the different tabs displayed. In the following screenshot, I have compared the difference between the admin role and end-user role functionality:

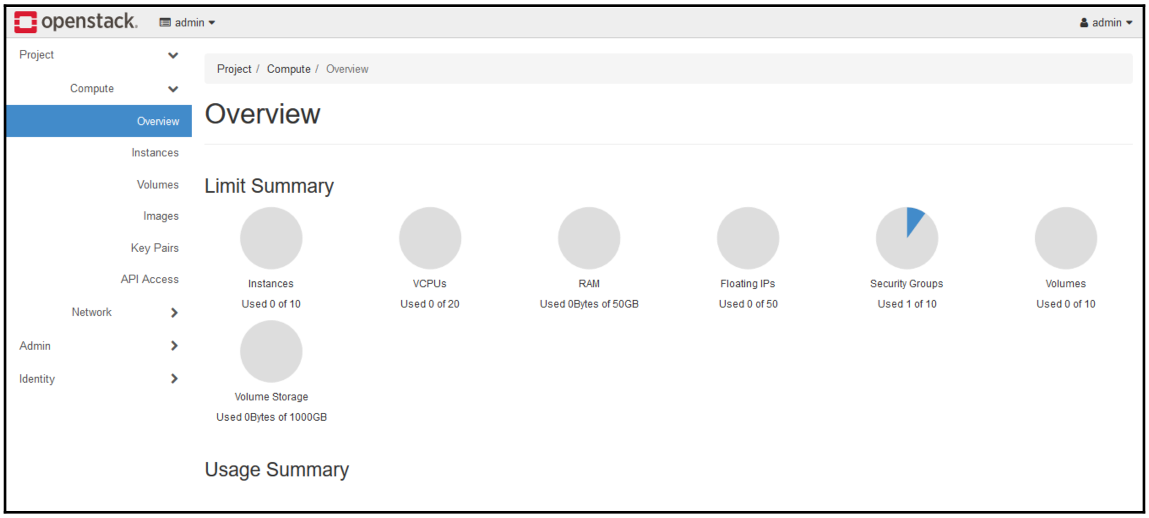


The left panel of the preceding figure displays the available options under the **Project** panel, which is accessible to all the end users. Typically, the non-privileged users will only have member role mapping. Well, on the other side, the **Admin** panel will have an available option for admin functionality.

I have also highlighted the difference in the options available under the **Compute** tab of both of the **Project** and **Admin** panels.

OpenStack Horizon - Project tab

From the **Project** tab, we could view and manage the OpenStack resources in a selected project. Each user should be mapped with at least one project. We can select the project from the drop-down menu at the top left:



The Compute tab

The **Compute** tab has the following options associated with it:

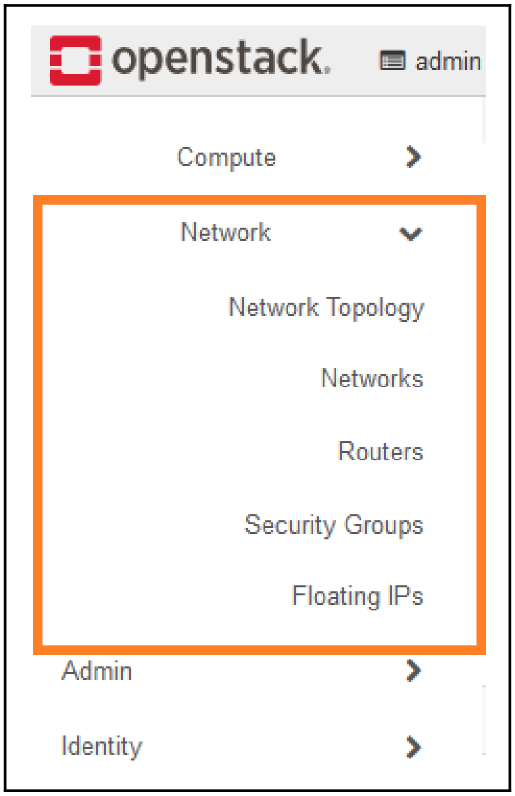
- **Overview**: To view project report that includes resource quota usage and summary.
- **Instances**: To view, launch, create a snapshot, stop, pause, or reboot instances, or connect to them through VNC.
- **Volumes**: To view, create, edit, and delete volumes.

- **Images**: To view images, instance snapshots, and volume snapshots created by project users, images that are publicly available. Also, it can create, edit, and delete images, and launch instances from images and snapshots.
- **Key Pairs**: To view, create, edit, and import SSH key-pairs, and delete keypairs.
- **API Access**: To view and download service API endpoint information.

The Network tab

The **Network** tab consists of the following options:

- **Network Topology** : To view the network topology of the project
- **Networks**: To create and manage the private/tenant networks



- **Routers**: To create and manage routers for your tenant/project
- **Security Groups**: View and manage the security groups and security group rules
- **Floating IPs**: Allocate an IP address to the virtual machine port or release it from a project

OpenStack Horizon - the Admin tab

The user with the admin role mapped could view the **Admin** tab on their Horizon dashboard. Administrative users can use the **Admin** tab to view the usage of entire OpenStack resources and to manage instances, volumes, flavors, images, public networks, and so on.

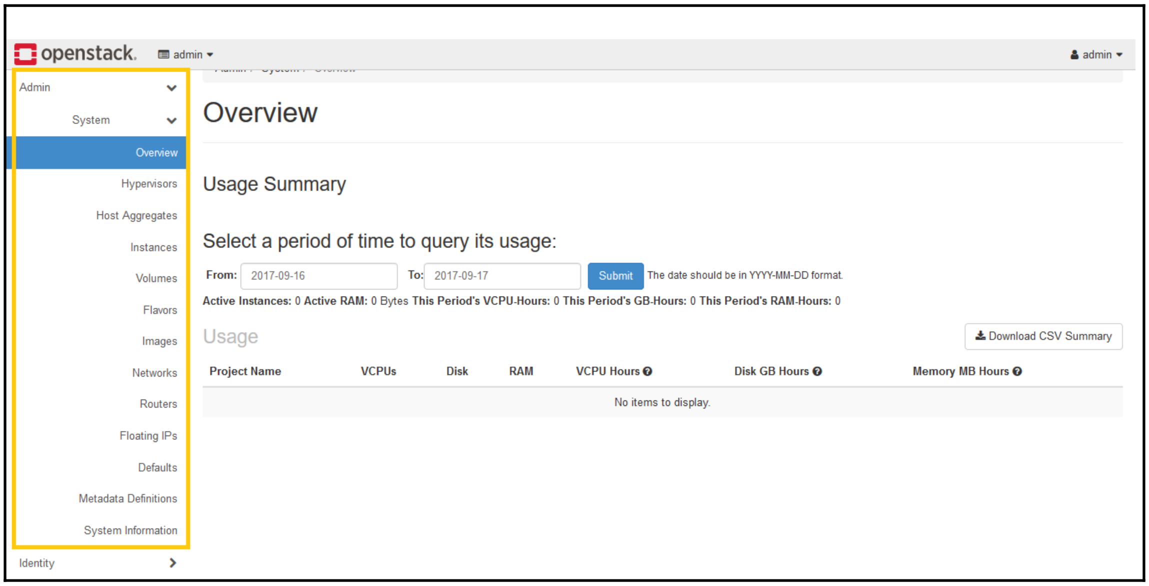
As I mentioned earlier, from the **Admin** tab, the user could view and manage the resources like virtual machines, volumes, images, and network details of any project/tenant.

The System tab

The **System** tab consists of the following options:

- **Overview**: To view basic reports.
- **Hypervisors**: To view the hypervisor summary.
- **Host Aggregates**: To view, create, and edit host aggregates. View the list of availability zones.
- **Instances**: To view, pause, resume, suspend, migrate, soft or hard reboots, and delete running instances that belong to users of any project. Also, view the log or access any instance through VNC.
- **Volumes**: To view, create, manage, and delete volumes and snapshots that belong to users of any projects.
- **Flavors**: To view, create, edit, view other specifications for, and delete flavors. A flavor defines the size template of an instance.
- **Images**: To view, create, edit properties for, and delete custom images and manage public images.

The following screenshot shows the **System** tab with an available panel under it:



Overview

Usage Summary

Select a period of time to query its usage:

From: 2017-09-16 To: 2017-09-17 The date should be in YYYY-MM-DD format.

Active Instances: 0 Active RAM: 0 Bytes This Period's VCPU-Hours: 0 This Period's GB-Hours: 0 This Period's RAM-Hours: 0

Usage

Project Name	VCPUs	Disk	RAM	VCPU Hours	Disk GB Hours	Memory MB Hours
No items to display.						

- **Networks**: To view, create, edit properties for, and delete private and public networks. Notably, only the **admin** user can manage the public network.
- **Routers**: To view, edit properties for, and delete routers that belong to any project.
- **Floating IPs**: To allocate an IP address to port or release it from a project.
- **Defaults**: To view default quota values. Also, manage the default quotas for each project in OpenStack by defining the maximum allowable size and number of resources.
- **Metadata Definitions**: To import namespace and view the metadata information.

Well done! Now, you have a better understanding of our lab environment. Let's start exploring the OpenStack.

Checkpoint

- Connect to the lab environment using CLI and Horizon
- Understand the lab environment setup

Compute Service - Horizon

In this session, we will use the Horizon dashboard to execute some basic Nova compute operations:

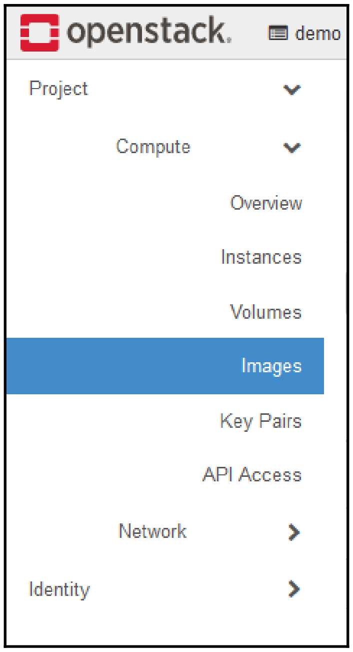
Goal	Use the Horizon dashboard to perform basic Nova compute operations
Activities	• Create an instance in Horizon • Connect to the instance console • Terminate newly created instance

Launching a new instance using Horizon

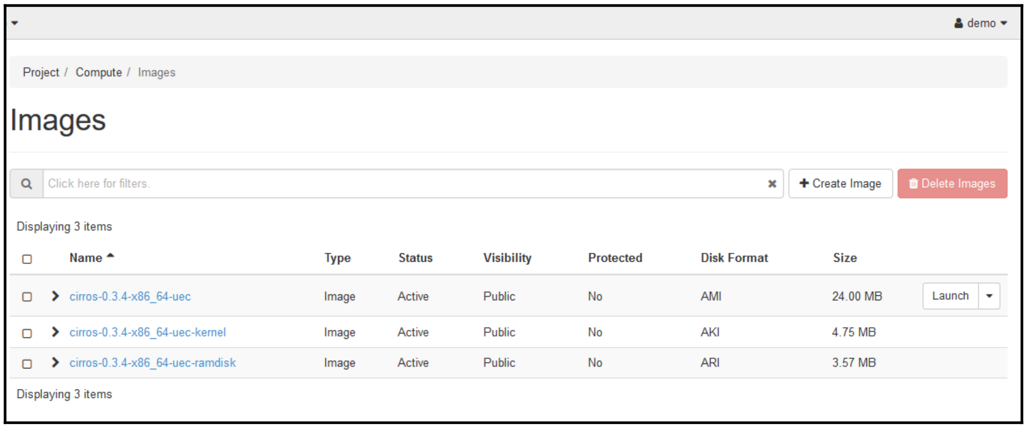
We refer to virtual machines as instances that run inside the cloud. You could provision a new instance from a pre-bundled OS image that is stored in an OpenStack image service. As I said, an image is a copy of the entire contents of a storage device, which includes all the hard drive partition information, the file allocation table, boot sectors, and the operating system installation.

The OpenStack Image (Glance) service provides a pool of images that are available to members of the different projects. Let's look at what images are available for the project **demo**:

1. Log in to the Horizon Dashboard as a **demo** user. You could find the password details of the **demo** user in the DevStack installation completion output. In most DevStack setup cases, the password for **admin** and **demo** users remains the same.
2. Click the **Images** panel in the **Project** tab on the Horizon dashboard:



In response, the Horizon will show the images that have been uploaded to the Glance repository and are available for this project:



Displaying 3 items

	Name	Type	Status	Visibility	Protected	Disk Format	Size
☐	cirros-0.3.4-x86_64-uec	Image	Active	Public	No	AMI	24.00 MB
☐	cirros-0.3.4-x86_64-uec-kernel	Image	Active	Public	No	AKI	4.75 MB
☐	cirros-0.3.4-x86_64-uec-ramdisk	Image	Active	Public	No	ARI	3.57 MB

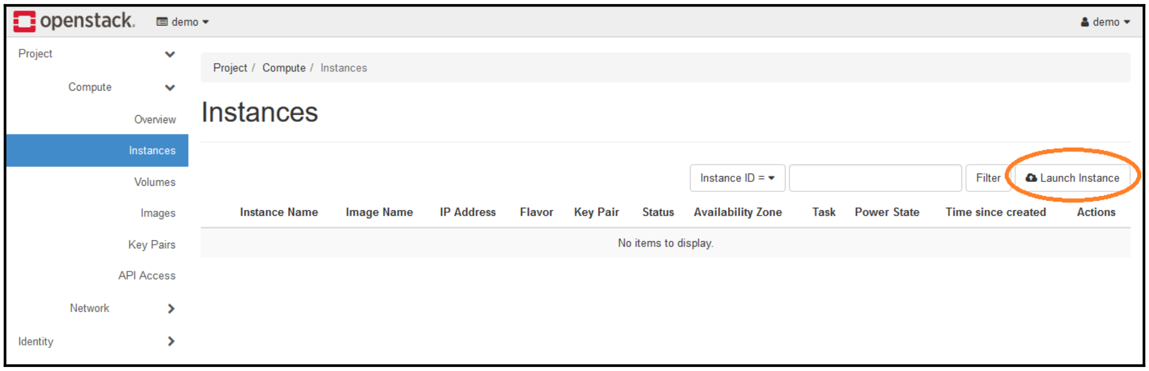
Displaying 3 items

Images visibility in Glance can be:

- **Private**: Available only for the selected project in which they were created
- **Shared**: Available for the project in which they were created and to other projects the image has been explicitly shared with
- **Public**: Available for all of the projects within the OpenStack cluster

We could launch a new instance from the Glance panel by clicking on the **Launch** button displayed next to the image list. However, let me walk through the straightforward method for creating a new virtual machine using the instance panel.

1. Click the **Instances** menu in the **Project** tab on the Horizon dashboard:

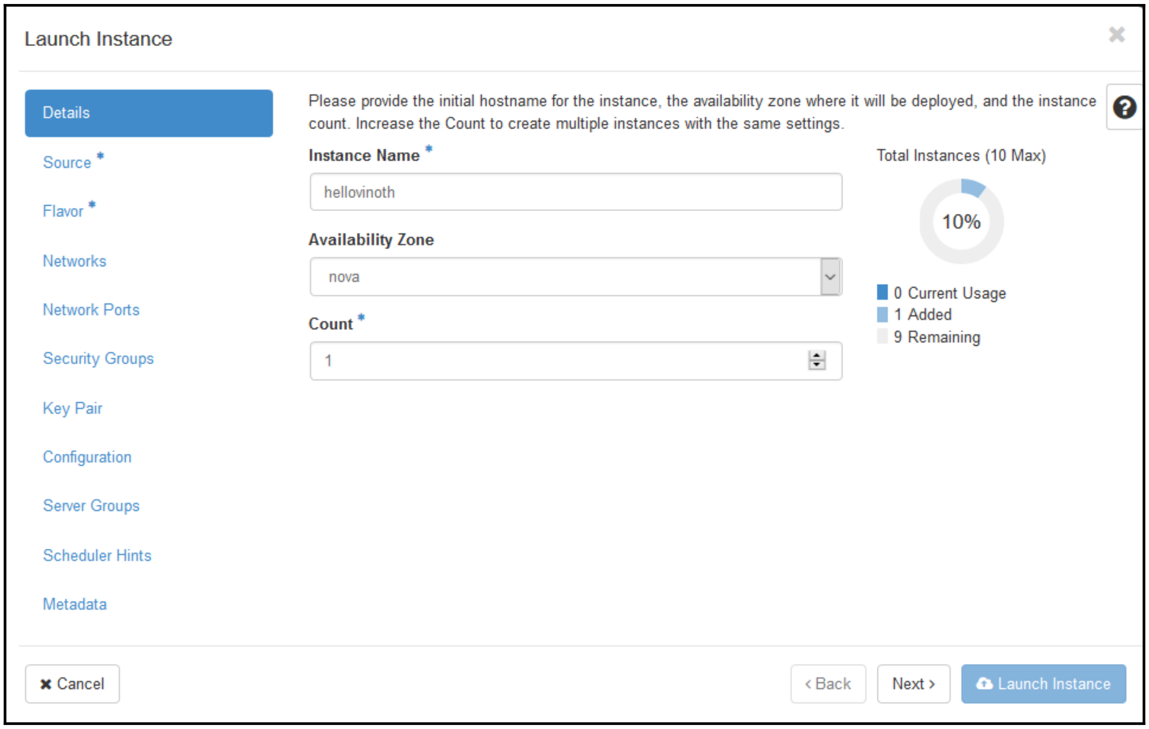


Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
No items to display.										

As you can see, there are no running VMs for now.

2. Click the **Launch Instance** button in the top right corner on the Horizon.

In response, the **Launch Instance** window will get a pop-up. We need to fill the inputs corresponding to the new virtual machine:



Launch Instance

Details
Source *
Flavor *
Networks
Network Ports
Security Groups
Key Pair
Configuration
Server Groups
Scheduler Hints
Metadata

Please provide the initial hostname for the instance, the availability zone where it will be deployed, and the instance count. Increase the Count to create multiple instances with the same settings.

Instance Name *: hellovinoth

Availability Zone: nova

Count *: 1

Total Instances (10 Max)

10%

0 Current Usage
1 Added
9 Remaining

To begin with, you may fill only the mandatory fields on the **Launch Instance** pop-up tab.

Mandatory fields

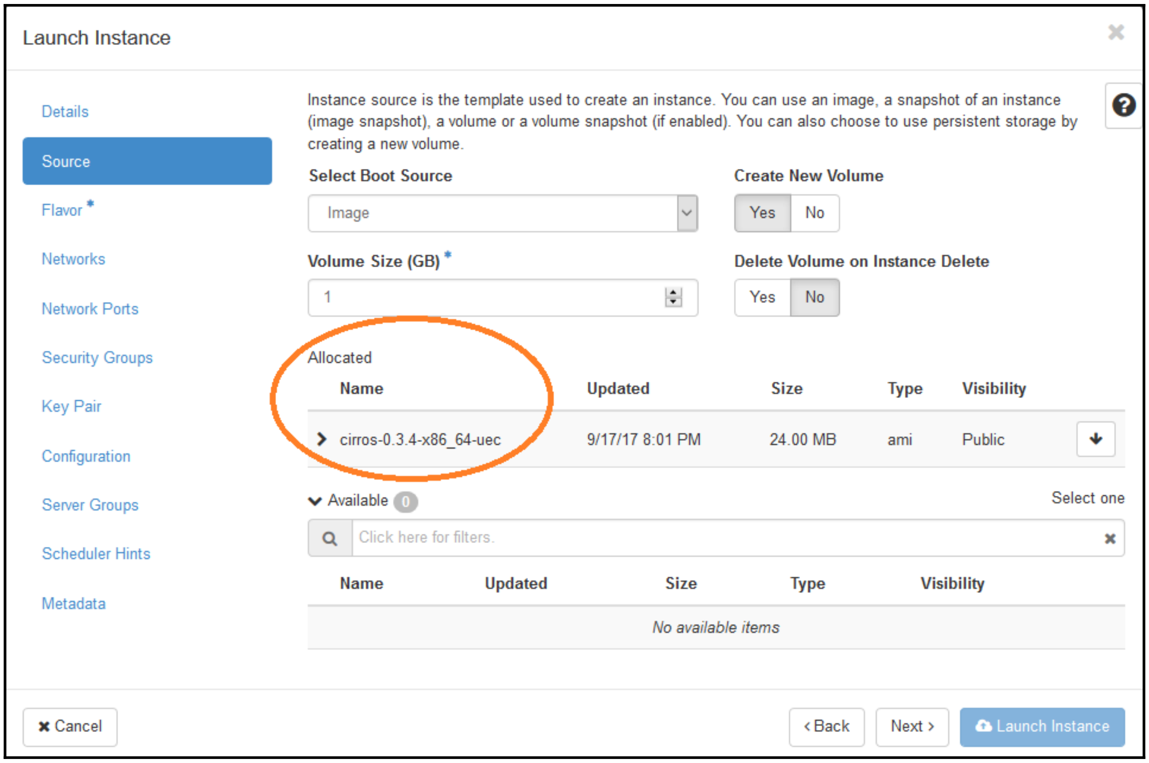
The following listed fields are the mandatory items when submitting the **Launch Instance** request:

- **Instance Name***: It's a name of the virtual machine. Also, it will be reflected as the hostname.
- **Count***: It counts the number of virtual machines that need to be provisioned with the same configuration.

- **Allocated Image**: We need to click (choose) any one image from the available image list to provision a new instance.
- **Allocated Flavor**: A flavor is an available hardware configuration for a server. OpenStack comes with a list of default flavors to be used by all users. Notably, only an administrator can modify the existing flavors and create new ones. In my case, due to the lack of hardware resource, I have chosen a minimal flavor, `m1.nano`.
- **Allocated Network**: Choose any one **Private** network from the available network list to provision a new virtual machine in that subnet.

In the **Launch Instance** window, specify the following values:

Details tab	
Instance Name*	`hellovinoth`
Availability Zone	`nova`
Count*	`1`
Source tab	
Select Boot Source	`Image`
Create New Volume	`Yes`
Volume Size (GB)*	`1`
Delete Volume on Instance Delete	`No`
Image Name	`cirros-0.3.4-x86_64-uec`



Instance source is the template used to create an instance. You can use an image, a snapshot of an instance (image snapshot), a volume or a volume snapshot (if enabled). You can also choose to use persistent storage by creating a new volume.

Select Boot Source: Image

Create New Volume: [✓ Yes] No

Volume Size (GB) *: 1

Delete Volume on Instance Delete: Yes [✓ No]

Allocated

Name	Updated	Size	Type	Visibility
cirros-0.3.4-x86_64-uec	9/17/17 8:01 PM	24.00 MB	ami	Public

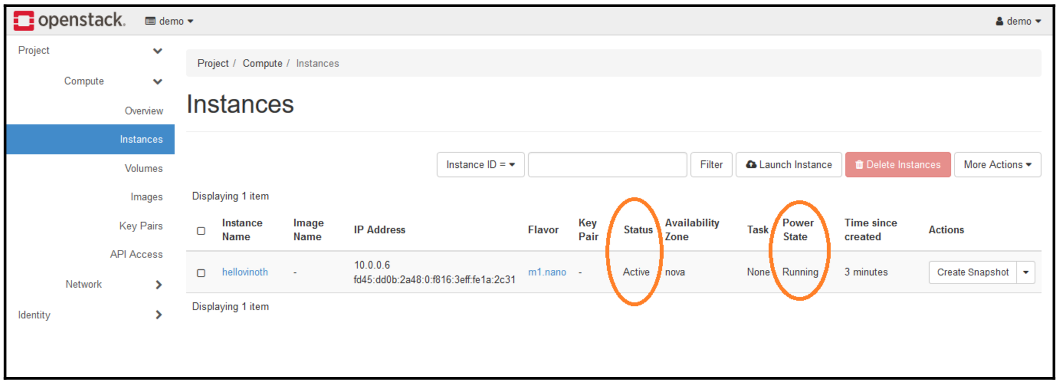
Available 0 Select one

Click here for filters.

Name	Updated	Size	Type	Visibility
No available items				

Flavor tab	
Allocated Flavor Name	`m1.nano`
Networks tab	
Allocated Network Name	`private`
Security Groups tab	
Allocated Security Group Name	`Default`

After filling all of the mandatory fields, click the **Launch** button to submit the instance parameter to the Nova services to process:


openstack. demo ▾ demo ▾

Project / Compute / Instances

Instances

Instance ID = ▾ Filter Launch Instance Delete Instances More Actions ▾

Displaying 1 item

	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
☐	hellovinoth	-	10.0.0.6 fd45:dd0b:2a48:0:f816:3eff:fe1a:2c31	m1.nano	-	Active	nova	None	Running	3 minutes	Create Snapshot ▾

Displaying 1 item

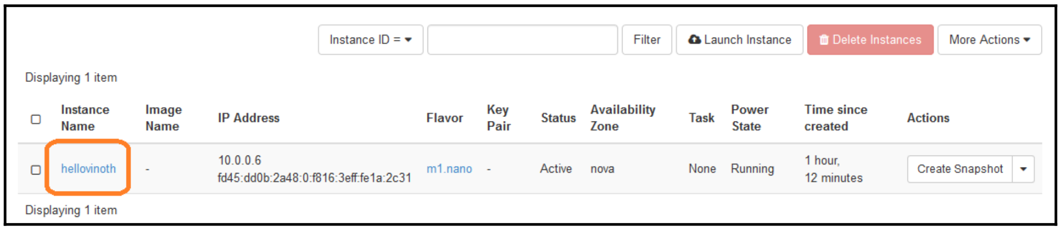
Then, pay close attention on how **Status**, **Task**, and **Power State** fields on the dashboard changes during our new instance creation. Wait until the status and **Power State** of the instance changes to **Active** and **Running** status, respectively.

Congratulate yourself! You have successfully created a new instance in OpenStack via the Horizon dashboard.

Connecting to the instance using the VNC console

In computing, **Virtual Network Computing** (**VNC**) is a graphical remote desktop sharing system that uses the **Remote Frame Buffer** (**RFB**) protocol to control another end computer remotely. In an IaaS system like OpenStack, VNC is a very convenient tool for the end user to connect to the VMs through web UI:

1. On the **Instance** screen, click the **Instance Name**: `hellovinoth`:

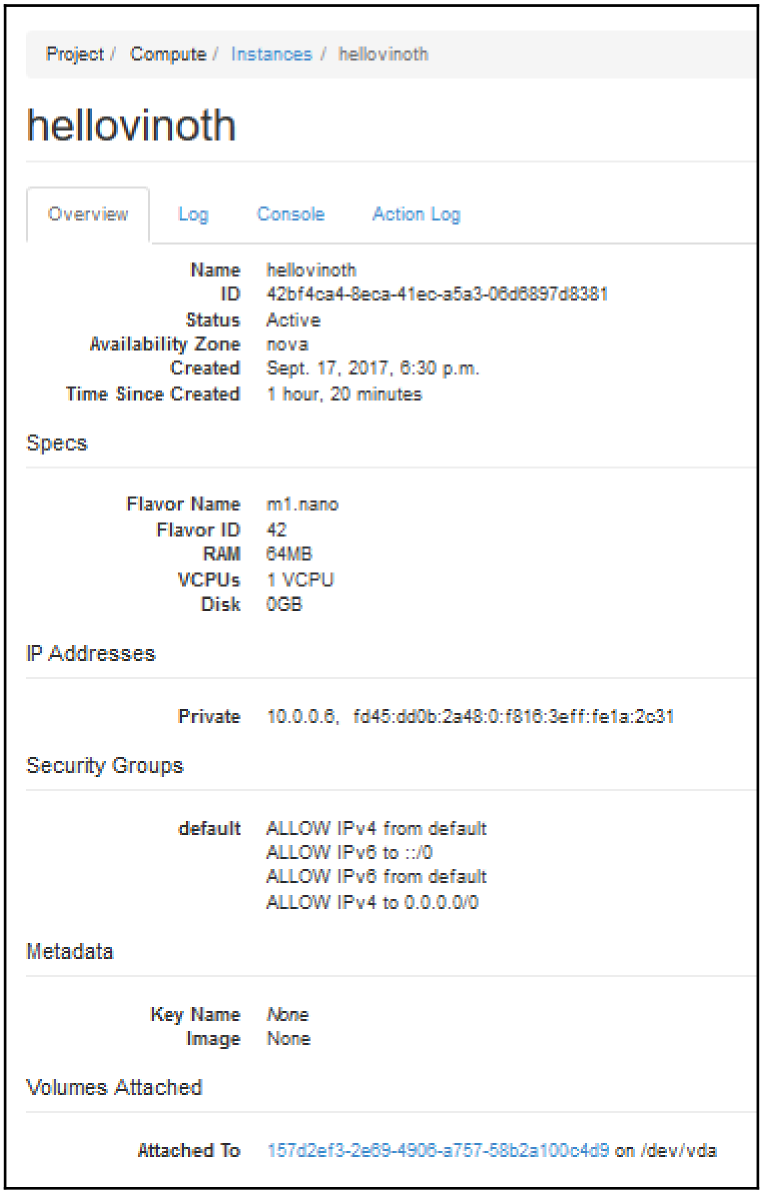

Instance ID = ▾ Filter Launch Instance Delete Instances More Actions ▾

Displaying 1 item

	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
☐	hellovinoth	-	10.0.0.6 fd45:dd0b:2a48:0:f816:3eff:fe1a:2c31	m1.nano	-	Active	nova	None	Running	1 hour, 12 minutes	Create Snapshot ▾

Displaying 1 item

2. In response, the **Instance Detail** window will get displayed with the **Overview** tab opened by default. In there, you can see the more detailed information on VM:



hellovinoth

Name	hellovinoth
ID	42bf4ca4-8eca-41ec-a5a3-08d6897d8381
Status	Active
Availability Zone	nova
Created	Sept. 17, 2017, 6:30 p.m.
Time Since Created	1 hour, 20 minutes

Specs

Flavor Name	m1.nano
Flavor ID	42
RAM	64MB
VCPUs	1 VCPU
Disk	0GB

IP Addresses

Private	10.0.0.6, fd45:dd0b:2a48:0:f816:3eff:fe1a:2c31

Security Groups

default	ALLOW IPv4 from default ALLOW IPv6 to ::/0 ALLOW IPv6 from default ALLOW IPv4 to 0.0.0.0/0

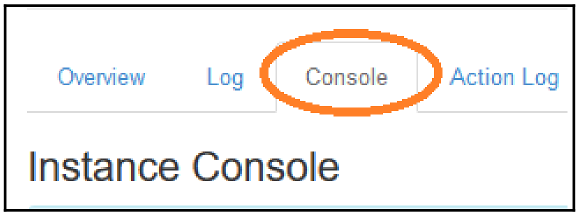
Metadata

Key Name	None
Image	None

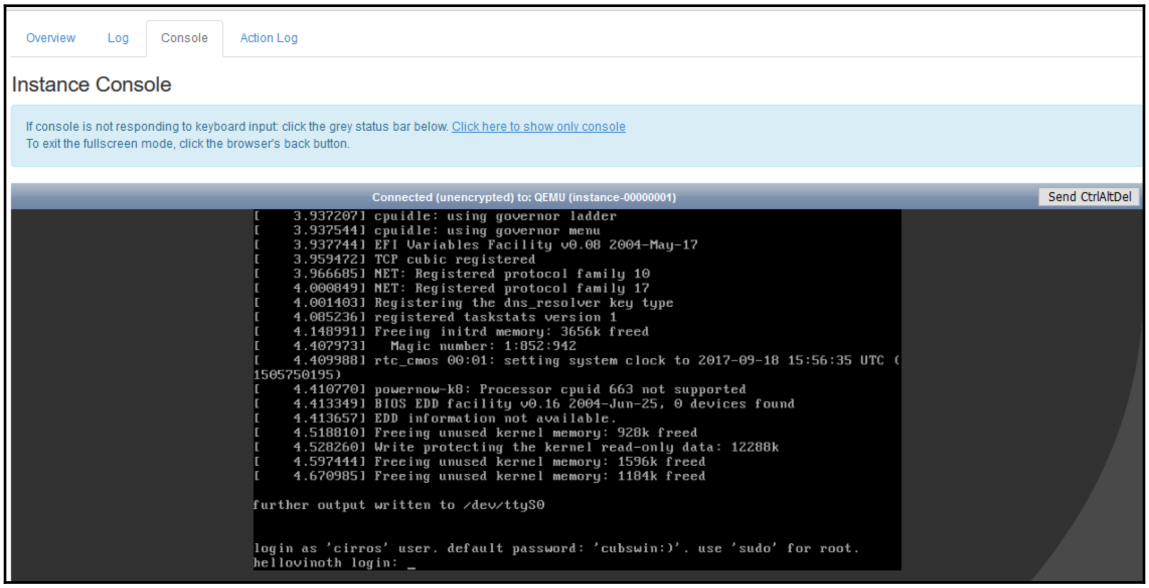
Volumes Attached

Attached To	157d2ef3-2e69-4906-a757-58b2a100c4d9 on /dev/vda

3. Now, click the **Console** tab next to the **Overview** tab on the **Instances** details window. The **Console** tab will navigate to the **Instance Console** screen. From there, you can see the VNC console to access the remote virtual machine:



4. Log in to the VM using the default credential, Username: `cirros` and Password: `cubswin:`



If console is not responding to keyboard input: click the grey status bar below. Click here to show only console
To exit the fullscreen mode, click the browser's back button.

Connected (unencrypted) to: QEMU (instance-00000001)

```
[    3.937207] cpuidle: using governor ladder
[    3.937544] cpuidle: using governor menu
[    3.937744] EFI Variables Facility v0.08 2004-May-17
[    3.959472] TCP cubic registered
[    3.966685] NET: Registered protocol family 10
[    4.000849] NET: Registered protocol family 17
[    4.001403] Registering the dns_resolver key type
[    4.085236] registered taskstats version 1
[    4.148991] Freeing initrd memory: 3656k freed
[    4.407973]   Magic number: 1:852:942
[    4.409988] rtc_cmos 00:01: setting system clock to 2017-09-18 15:56:35 UTC (
1505750195)
[    4.410770] powernow-k8: Processor cpuid 663 not supported
[    4.413349] BIOS EDD facility v0.16 2004-Jun-25, 0 devices found
[    4.413657] EDD information not available.
[    4.518810] Freeing unused kernel memory: 928k freed
[    4.528260] Write protecting the kernel read-only data: 12288k
[    4.597444] Freeing unused kernel memory: 1596k freed
[    4.670985] Freeing unused kernel memory: 1184k freed

further output written to /dev/ttyS0


login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.
hellovinoth login: _
```

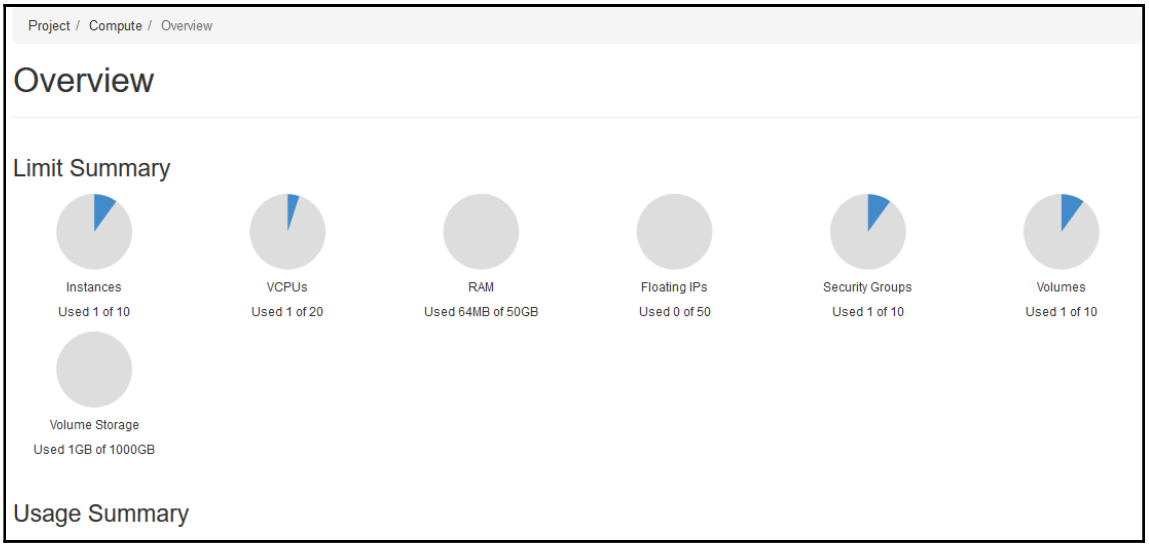
If the VNC console is not responding to keyboard input: click on the grey area and then try typing at the prompt. If you see a Command Prompt, you have successfully connected to the VM using the VNC console.

Tracking usage for instances

OpenStack Horizon enables the user to track the resource utilization for each project via UI-based metrics. Using the UI data under the **Overview** tab, users of the selected project could track the resource usage such as **Instances** count, **vCPUs**, disk, **RAM**, **Floating IPs**, **Security Group** count, **Volumes** count and **Volumes Storage**:

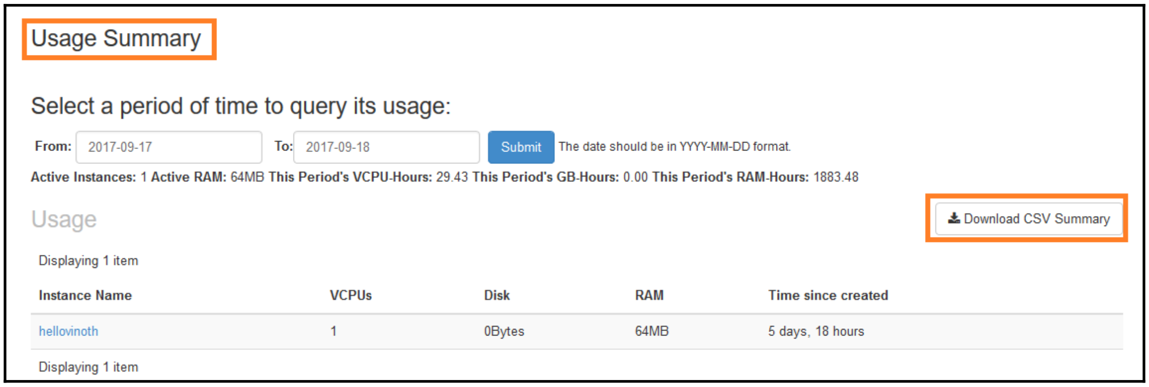
1. Click the **Overview** tab in the left navigation panel on the **Project** tab in Horizon.
2. The **Overview** screen displays the summary of resource utilization related to the selected project and its quotas.
3. The **Overview** screen has two sub-divisions in it to serve its purpose:
    - **Limit Summary**
    - *** Usage Summary**

The following screenshot shows the **Overview** screen displaying the resource quota summary:



- **Limit Summary**: It is the usage summary of the currently running instances, more like a live report
- **Usage Summary**: It enables the user to fetch the summary report between the selected days

The following screenshot shows the latter half of the **Overview** screen displaying the **Usage Summary**:


Usage Summary

Select a period of time to query its usage:

From: 2017-09-17 To: 2017-09-18 Submit The date should be in YYYY-MM-DD format.

Active Instances: 1 **Active RAM:** 64MB **This Period's VCPU-Hours:** 29.43 **This Period's GB-Hours:** 0.00 **This Period's RAM-Hours:** 1883.48

Usage

Download CSV Summary

Displaying 1 item

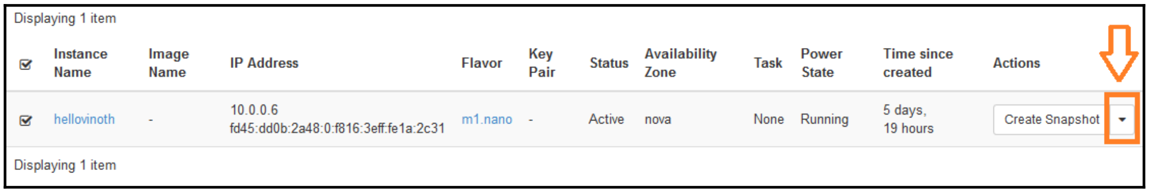
Instance Name	VCPUs	Disk	RAM	Time since created
hellovinoth	1	0Bytes	64MB	5 days, 18 hours

Displaying 1 item

Optionally, users could also download the summary report in CSV file format. Add to that; the **Active Instance** provides detailed information on the usage for each VM in the **Usage Summary** section. As of now, we have only one virtual machine `hellovinoth` running in our lab environment. So, the report on the preceding figure listed only one virtual machine and its corresponding resource summary.

Managing an instance

1. Click the **Instances** menu in the left navigation panel on the **Project** tab in Horizon.
2. Click the down arrow button (highlighted in the following screenshot) next to the **Create Snapshot** button to expand the list of actions available for the instance:


Displaying 1 item

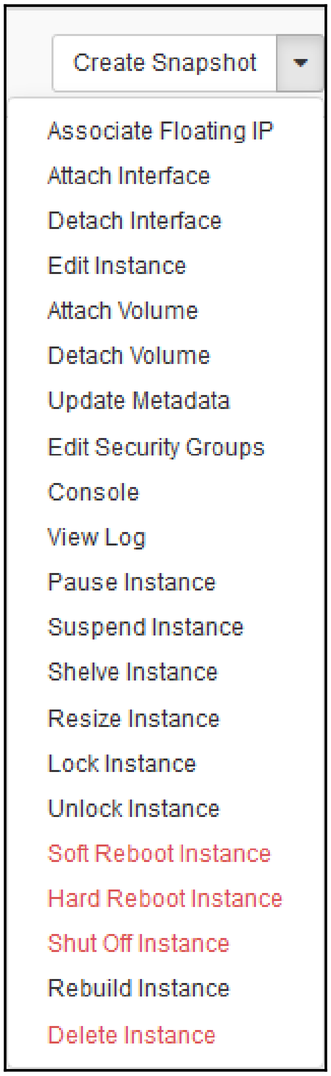
☑	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
☑	hellovinoth	-	10.0.0.6 fd45:dd0b:2a48:0:f816:3eff:fe1a:2c31	m1.nano	-	Active	nova	None	Running	5 days, 19 hours	Create Snapshot ▾

Displaying 1 item

There are numerous actions available for an instance in the **Active** state; let's learn about the purpose of the actions available for an active instance:

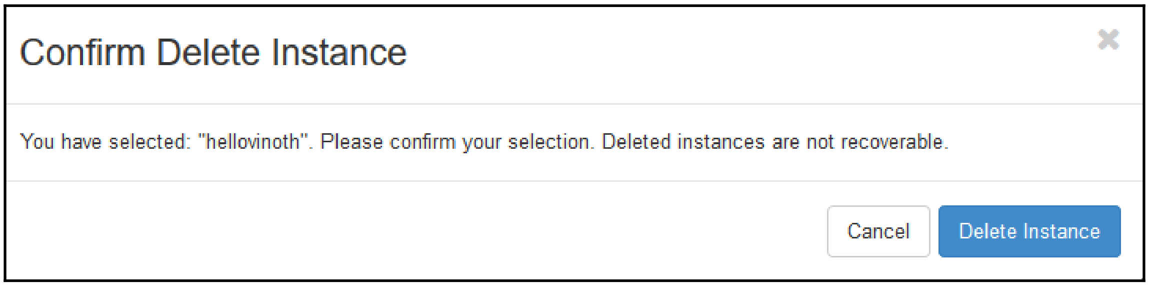
- **Associate Floating IP**: Map (NAT) dedicated public IP from the external network. We use this so that we can access this VM from the external network
- **Attach Interface**: To add more NIC to the VM. By default, you could provision a VM with one NIC attached
- **Detach Interface**: To remove the added interface from the VM
- **Edit Instance**: To rename the VM display name in the Horizon for your easy reference
- **Attach Volume**: To map the secondary disk to the VM from the Cinder service
- **Detach Volume**: To remove the mapped disk from the VM
- **Update Metadata**: To edit the instance metadata information
- **Edit Security Groups**: To add/change the security groups mapping to the VM
- **Console**: An alternative way to navigate to the page with VNC console for the VM
- **View Log**: Navigate to the **Instance Console Log** screen and display the instance live `dmesg` logs
- **Pause Instance**: To store the state of the VM in RAM. Just like putting the PC in sleep mode
- **Suspend Instance**: To store the VM state on disk. All RAM will be written to the hard disk, and the VM will set to stopped state. Just like putting the PC in hibernate mode
- **Shelve Instance**: This will create a snapshot (backup) for the VM and delete all the underlying resource of VM. Just like deleting the actual content from the book without removing the index entry
- **Unshelve Instance**: This will restore the shelved VM by restoring the latest snapshot
- **Resize Instance**: To resize the virtual machines flavor. This process will also reboot the virtual machine
- **Lock Instance**: To lock actions for the VM, so that a non-admin user will not be able to execute actions
- **Unlock Instance**: To unlock the actions for the VM
- **Soft Reboot Instance**: Soft reboot attempts a graceful shutdown and restarts the virtual machine
- **Hard Reboot Instance**: Power cycles the instance

The following screenshot shows the list of available options under the action drop-down list:



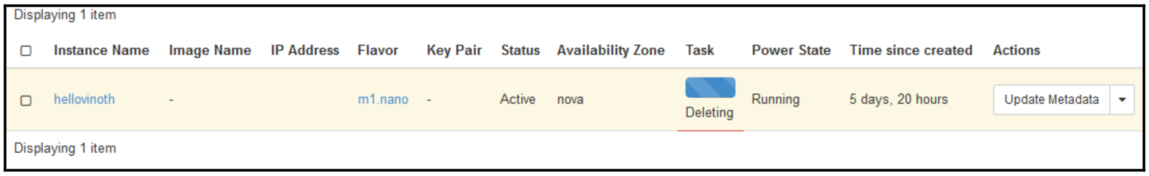
- **Shut Off Instance**: To put the VM in shut off state
- **Rebuild Instance**: To restore the VM with the same configuration using the image/snapshot available
- **Delete Instance**: To delete an instance when you no longer need it

Choose the **Delete Instance** option from the action list to delete the instance. In response, you will get the pop-up asking for the confirmation to delete the virtual machine:


Confirm Delete Instance

You have selected: "hellovinoth". Please confirm your selection. Deleted instances are not recoverable.

After confirmation, in a few seconds, the VM will be deleted, and the instances window becomes empty:


Displaying 1 item

☐	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
☐	hellovinoth	-		m1.nano	-	Active	nova	Deleting	Running	5 days, 20 hours	Update Metadata

Displaying 1 item

Congratulate yourself! You have completed the compute service exploration via Horizon in the lab exercise.

Checkpoint

- Get familiar with OpenStack Horizon
- Provision a new instance via the OpenStack Dashboard
- Connect to the newly provisioned VM using VNC web console
- Understand the project usage summary
- Look at actions available for the VM in an active state
- Terminate the newly provisioned VM

Compute service - CLI

In this session, we will use the OpenStack **command-line interface** (**CLI**) to perform some basic Nova compute operations:

Goal	Use the OpenStack CLI to perform basic Nova compute operations
Activities	• Launch an **Instance Using OpenStack** CLI • Connect to the **Instance Using SSH** • Terminate the created instance

OpenStack CLI clients (OSC)

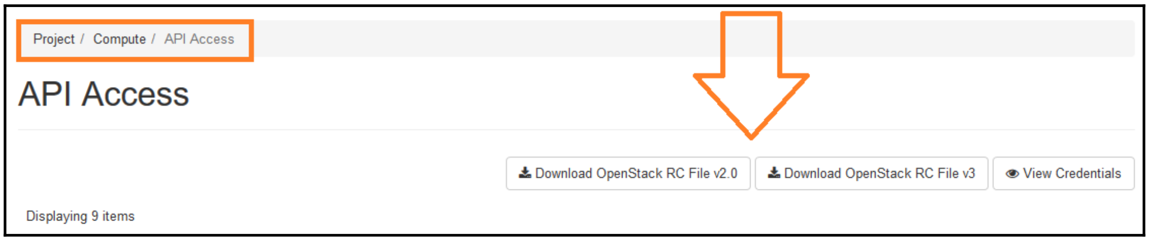
OpenStack offers the flexibility to manage its resources through web UI as well as using CLI. In fact, OpenStack command-line clients support more operations and command parameters than the Horizon UI.

During earlier OpenStack release, each OpenStack service had its own command-line client such as nova-CLI, neutron-CLI, Cinder-CLI, and so on. However, in the recent OpenStack releases, all those distinct commands set for compute, identity, image, object storage, and block storage APIs are packed in a single shell with a uniform CLI structure called **OpenStack Client** (**OSC**).

All of those distinct CLI are now deprecated in the latest OpenStack release and will be removed in the upcoming release cycle. The command mapping for old CLI to new OSC is available at:
`https://docs.openstack.org/python-openstackclient/latest/cli/decoder.html`.

Before we start using the OpenStack command tool, we need to provide OSC with information on your OpenStack username, password, and project, as well as an endpoint URL for KeyStone to contact for authentication and getting the list of endpoint URLs for the other OpenStack service. We could either specify that information directly in each command or alternatively, we could set an environment file and use it for the whole command-line session.

We could create that environment file on our own, or alternatively, we could even download it directly from the OpenStack dashboard under the **API Access** tab:


Project / Compute / API Access
API Access
Download OpenStack RC File v2.0
Download OpenStack RC File v3
View Credentials
Displaying 9 items

To create an environment file, follow these given instructions:

1. From your workstation using PuTTy or Terminal, SSH to the virtual machine or server where we have installed all-in-one OpenStack setup using DevStack on Day 1. Please go back to `Chapter 1`, *Day 1 - Build Your Camp* and recap the lab environment setup.
2. Log in using the stack/stack username and password:

```
ssh stack@192.168.56.101
#Command will prompt for password
```

3. Create a new environment file for your lab environment with the following details:

```
#!/usr/bin/env bash
export OS_AUTH_URL=http://192.168.1.6/identity/v3
export OS_PROJECT_NAME="demo"
export OS_USER_DOMAIN_NAME="Default"
export OS_USERNAME="demo"
export OS_PASSWORD="password"     --- ##your PASSWORD HERE.
export OS_REGION_NAME="RegionOne"
export OS_IDENTITY_API_VERSION=3
```

4. Alternatively, you could download the OpenStack RC file `v3` from the Horizon dashboard and move the file to the OpenStack cluster.
5. Source the environment file to set environment variables for your current shell using the following command:

```
source /opt/stack/devstack/demorc
```

6. The preceding command has no output in response. To verify if the environment variables are set correctly, execute the following command:

```
export | grep OS_
```

7. In response, you should get an output similar to the one shown here:

```
declare -x OS_AUTH_URL="http://192.168.1.6/identity/v3"
declare -x OS_IDENTITY_API_VERSION="3"
declare -x OS_PASSWORD="password"
declare -x OS_PROJECT_NAME="demo"
declare -x OS_REGION_NAME="RegionOne"
declare -x OS_USERNAME="demo"
declare -x OS_USER_DOMAIN_NAME="Default"
```

Whenever you open the new command-line window in your workstation, you need to reconnect to the SSH session and source the environment file again.

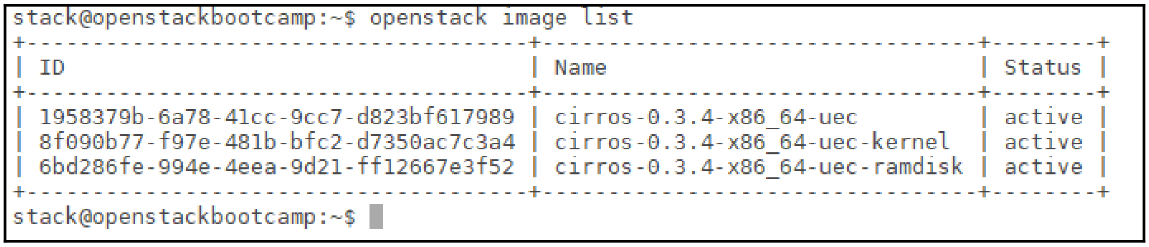
8. To start with, let's try some simple commands in OSC:

```
openstack --version
#Returns the version number for Openstack command-line client.
openstack service list
# Will return the ERROR response, as the demo user (non-admin) is
not authorised to perform the admin action
openstack network list
#Displays the list of networks in the project Demo.
```

Launching an instance using OpenStack CLI

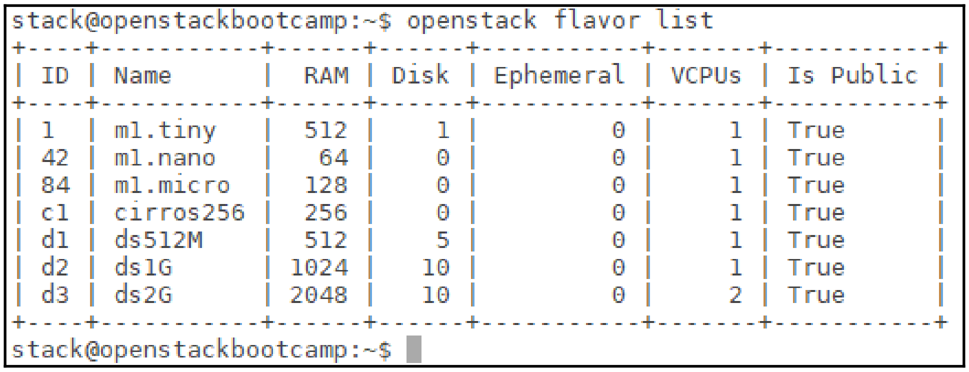
Before launching an instance, we need to gather some additional information that needs to be passed as an input parameter to the OpenStack CLI, such as image ID, flavor name, network ID, and so on:

1. List the available images for the project **demo**:



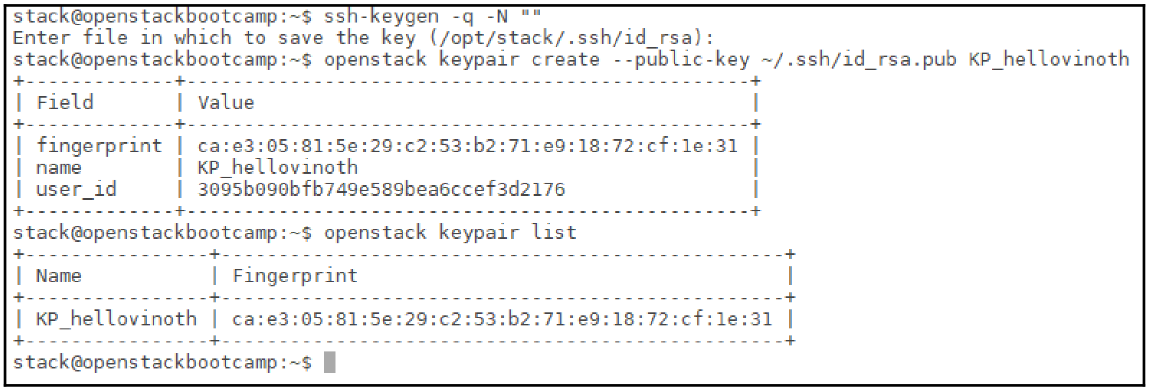
```
stack@openstackbootcamp:~$ openstack image list
+--------------------------------------+---------------------------------+--------+
| ID                                   | Name                            | Status |
+--------------------------------------+---------------------------------+--------+
| 1958379b-6a78-41cc-9cc7-d823bf617989 | cirros-0.3.4-x86_64-uec         | active |
| 8f090b77-f97e-481b-bfc2-d7350ac7c3a4 | cirros-0.3.4-x86_64-uec-kernel  | active |
| 6bd286fe-994e-4eea-9d21-ff12667e3f52 | cirros-0.3.4-x86_64-uec-ramdisk | active |
+--------------------------------------+---------------------------------+--------+
stack@openstackbootcamp:~$
```

2. List the available flavors for the project **demo**:



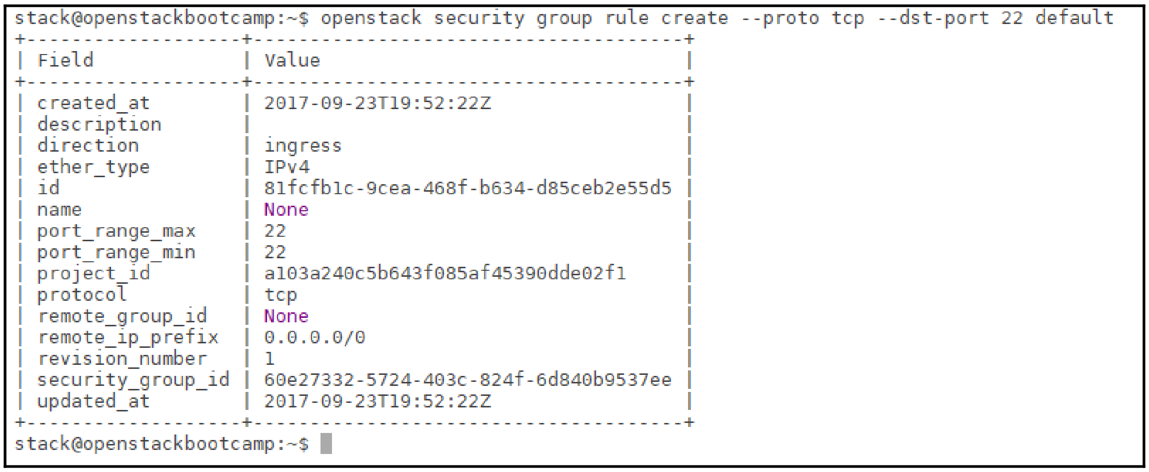
```
stack@openstackbootcamp:~$ openstack flavor list
+----+-----------+------+------+-----------+-------+-----------+
| ID | Name      |  RAM | Disk | Ephemeral | VCPUs | Is Public |
+----+-----------+------+------+-----------+-------+-----------+
| 1  | m1.tiny   |  512 |    1 |         0 |     1 | True      |
| 42 | m1.nano   |   64 |    0 |         0 |     1 | True      |
| 84 | m1.micro  |  128 |    0 |         0 |     1 | True      |
| c1 | cirros256 |  256 |    0 |         0 |     1 | True      |
| d1 | ds512M    |  512 |    5 |         0 |     1 | True      |
| d2 | ds1G      | 1024 |   10 |         0 |     1 | True      |
| d3 | ds2G      | 2048 |   10 |         0 |     2 | True      |
+----+-----------+------+------+-----------+-------+-----------+
stack@openstackbootcamp:~$
```

3. Generate a new key pair:



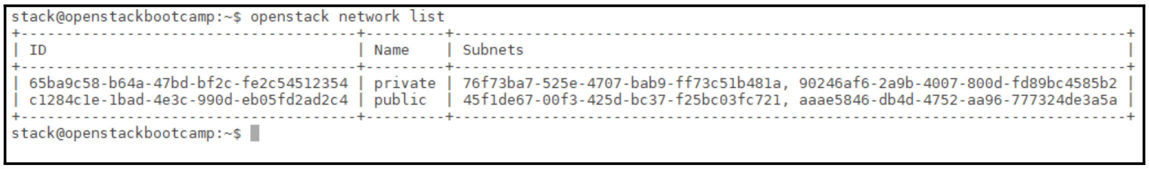
```
stack@openstackbootcamp:~$ ssh-keygen -q -N ""
Enter file in which to save the key (/opt/stack/.ssh/id_rsa):
stack@openstackbootcamp:~$ openstack keypair create --public-key ~/.ssh/id_rsa.pub KP_hellovinoth
+-------------+-------------------------------------------------+
| Field       | Value                                           |
+-------------+-------------------------------------------------+
| fingerprint | ca:e3:05:81:5e:29:c2:53:b2:71:e9:18:72:cf:1e:31 |
| name        | KP_hellovinoth                                  |
| user_id     | 3095b090bfb749e589bea6ccef3d2176                |
+-------------+-------------------------------------------------+
stack@openstackbootcamp:~$ openstack keypair list
+----------------+-------------------------------------------------+
| Name           | Fingerprint                                     |
+----------------+-------------------------------------------------+
| KP_hellovinoth | ca:e3:05:81:5e:29:c2:53:b2:71:e9:18:72:cf:1e:31 |
+----------------+-------------------------------------------------+
stack@openstackbootcamp:~$
```

4. By default, the security group named `default` applies to all instances that have firewall rules that deny remote access to the instances. So, we need to append the new rule to the security group to allow SSH connection to the VM:



```
stack@openstackbootcamp:~$ openstack security group rule create --proto tcp --dst-port 22 default
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| created_at        | 2017-09-23T19:52:22Z                 |
| description       |                                      |
| direction         | ingress                              |
| ether_type        | IPv4                                 |
| id                | 81fcfb1c-9cea-468f-b634-d85ceb2e55d5 |
| name              | None                                 |
| port_range_max    | 22                                   |
| port_range_min    | 22                                   |
| project_id        | a103a240c5b643f085af45390dde02f1     |
| protocol          | tcp                                  |
| remote_group_id   | None                                 |
| remote_ip_prefix  | 0.0.0.0/0                            |
| revision_number   | 1                                    |
| security_group_id | 60e27332-5724-403c-824f-6d840b9537ee |
| updated_at        | 2017-09-23T19:52:22Z                 |
+-------------------+--------------------------------------+
stack@openstackbootcamp:~$
```

5. List available networks for the project **demo**:



```
stack@openstackbootcamp:~$ openstack network list
+--------------------------------------+---------+----------------------------------------------------------------------------+
| ID                                   | Name    | Subnets                                                                    |
+--------------------------------------+---------+----------------------------------------------------------------------------+
| 65ba9c58-b64a-47bd-bf2c-fe2c54512354 | private | 76f73ba7-525e-4707-bab9-ff73c51b481a, 90246af6-2a9b-4007-800d-fd89bc4585b2 |
| c1284c1e-1bad-4e3c-990d-eb05fd2ad2c4 | public  | 45f1de67-00f3-425d-bc37-f25bc03fc721, aaae5846-db4d-4752-aa96-777324de3a5a |
+--------------------------------------+---------+----------------------------------------------------------------------------+
stack@openstackbootcamp:~$
```

6. Now, we have all of the required information to launch an instance from the OSC:

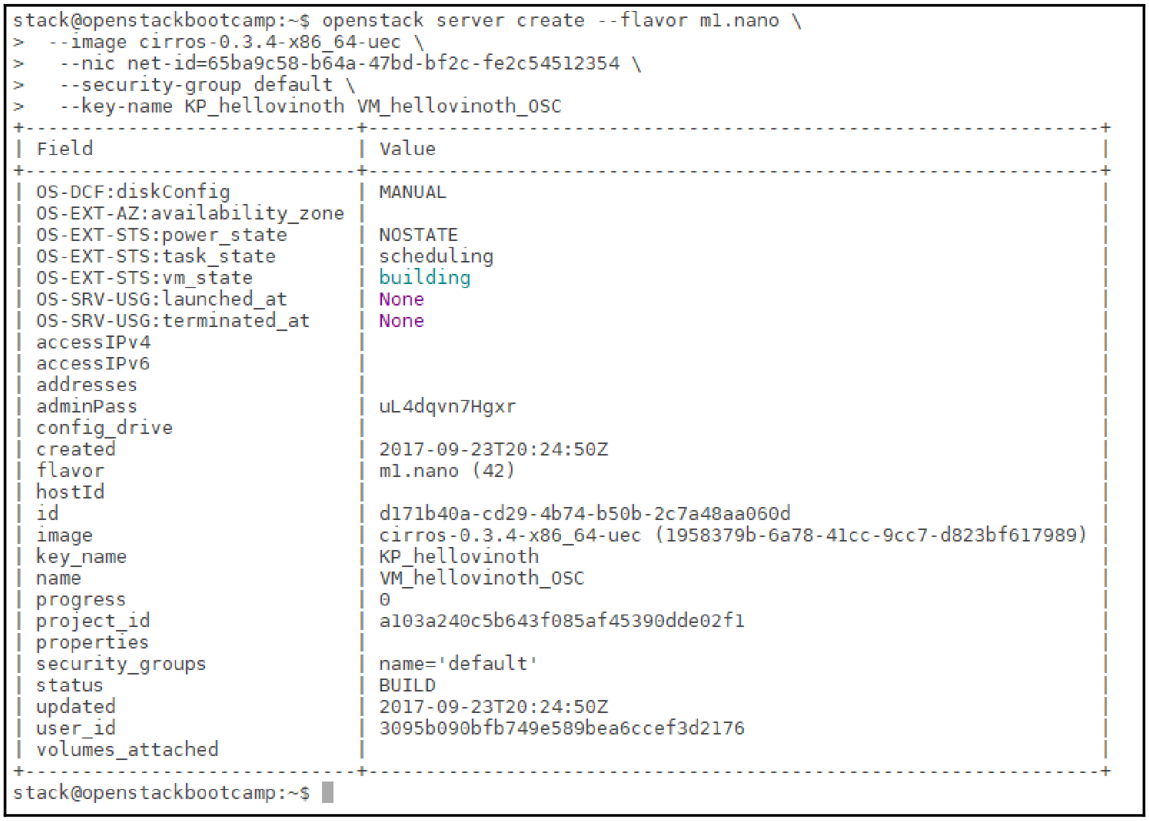
Parameter	**Value**
Image	`cirros-0.3.4-x86_64-uec`
Flavor	`m1.nano`
Keypair	`KP_hellovinoth`
Security Group	`default`
Network ID	`65ba9c58-b64a-47bd-bf2c-fe2c54512354`
Instance Name	`VM_hellovinoth_OSC`

7. Launch an instance using the following OpenStack CLI:

```
openstack server create --flavor m1.nano \
  --image cirros-0.3.4-x86_64-uec \
  --nic net-id=65ba9c58-b64a-47bd-bf2c-fe2c54512354 \
  --security-group default \
  --key-name KP_hellovinoth VM_hellovinoth_OSC
```

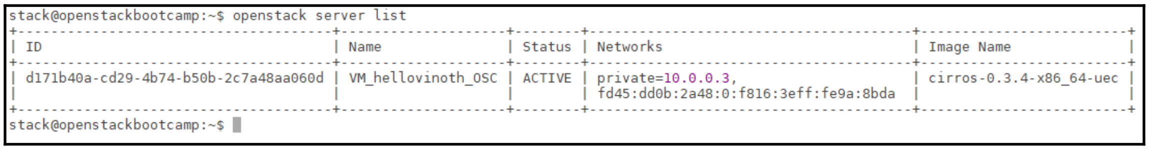
TIP: Replace `net-id` with the ID of your private network from the command output `openstack network list`.

8. You may refer to the following screenshot for the preceding command and its output:



```
stack@openstackbootcamp:~$ openstack server create --flavor m1.nano \
>  --image cirros-0.3.4-x86_64-uec \
>   --nic net-id=65ba9c58-b64a-47bd-bf2c-fe2c54512354 \
>   --security-group default \
>   --key-name KP_hellovinoth VM_hellovinoth_OSC
+-----------------------------+------------------------------------------------------------------+
| Field                       | Value                                                            |
+-----------------------------+------------------------------------------------------------------+
| OS-DCF:diskConfig           | MANUAL                                                           |
| OS-EXT-AZ:availability_zone |                                                                  |
| OS-EXT-STS:power_state      | NOSTATE                                                          |
| OS-EXT-STS:task_state       | scheduling                                                       |
| OS-EXT-STS:vm_state         | building                                                         |
| OS-SRV-USG:launched_at      | None                                                             |
| OS-SRV-USG:terminated_at    | None                                                             |
| accessIPv4                  |                                                                  |
| accessIPv6                  |                                                                  |
| addresses                   |                                                                  |
| adminPass                   | uL4dqvn7Hgxr                                                     |
| config_drive                |                                                                  |
| created                     | 2017-09-23T20:24:50Z                                             |
| flavor                      | m1.nano (42)                                                     |
| hostId                      |                                                                  |
| id                          | d171b40a-cd29-4b74-b50b-2c7a48aa060d                             |
| image                       | cirros-0.3.4-x86_64-uec (1958379b-6a78-41cc-9cc7-d823bf617989)   |
| key_name                    | KP_hellovinoth                                                   |
| name                        | VM_hellovinoth_OSC                                               |
| progress                    | 0                                                                |
| project_id                  | a103a240c5b643f085af45390dde02f1                                 |
| properties                  |                                                                  |
| security_groups             | name='default'                                                   |
| status                      | BUILD                                                            |
| updated                     | 2017-09-23T20:24:50Z                                             |
| user_id                     | 3095b090bfb749e589bea6ccef3d2176                                 |
| volumes_attached            |                                                                  |
+-----------------------------+------------------------------------------------------------------+
stack@openstackbootcamp:~$
```

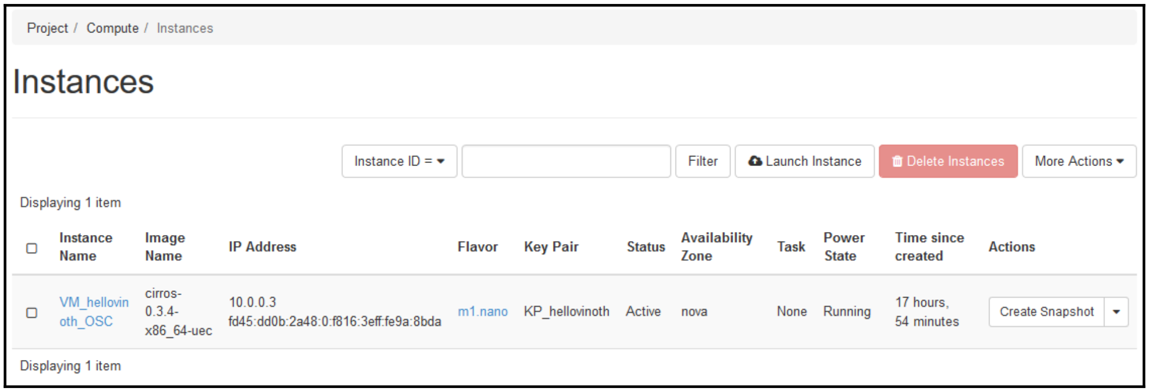
9. Wait for the VM to transition to `Active` status:



```
stack@openstackbootcamp:~$ openstack server list
+--------------------------------------+------------------+--------+-------------------------------------------+---------------------------+
| ID                                   | Name             | Status | Networks                                  | Image Name                |
+--------------------------------------+------------------+--------+-------------------------------------------+---------------------------+
| d171b40a-cd29-4b74-b50b-2c7a48aa060d | VM_hellovinoth_OSC | ACTIVE | private=10.0.0.3,                       | cirros-0.3.4-x86_64-uec |
|                                      |                  |        | fd45:dd0b:2a48:0:f816:3eff:fe9a:8bda      |                           |
+--------------------------------------+------------------+--------+-------------------------------------------+---------------------------+
stack@openstackbootcamp:~$
```

Connecting to the instance using SSH

We could also see the newly provisioned instance and its details from the Horizon dashboard:



Project / Compute / Instances

Instances

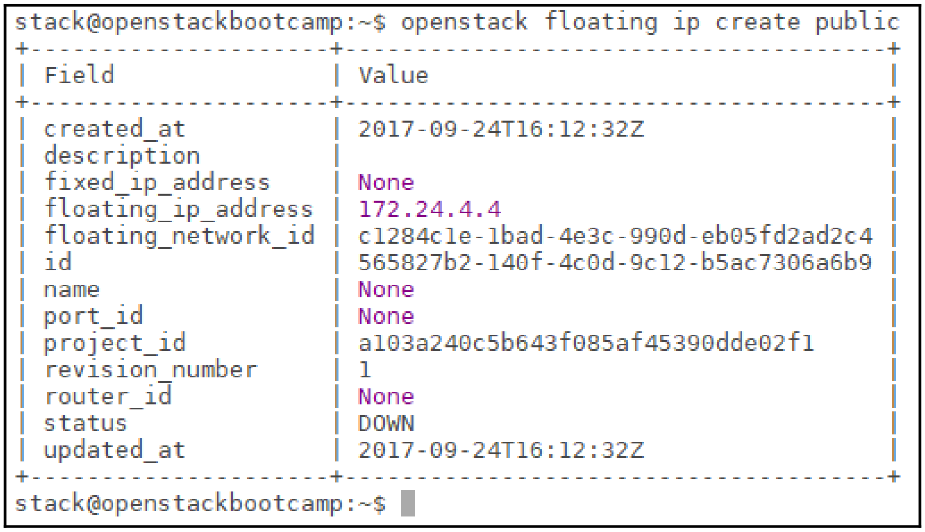
Displaying 1 item

	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
☐	VM_hellovinoth_OSC	cirros-0.3.4-x86_64-uec	10.0.0.3 fd45:dd0b:2a48:0:f816:3eff:fe9a:8bda	m1.nano	KP_hellovinoth	Active	nova	None	Running	17 hours, 54 minutes	Create Snapshot ▾

Displaying 1 item

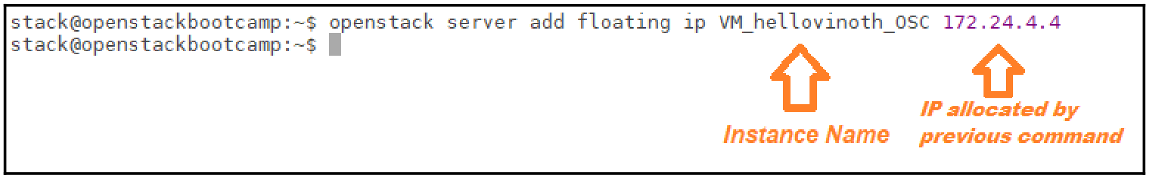
To connect the instance from the external network, we need to associate the floating IP from the external network:

1. Create a floating IP address on the virtual provider network using the following command:



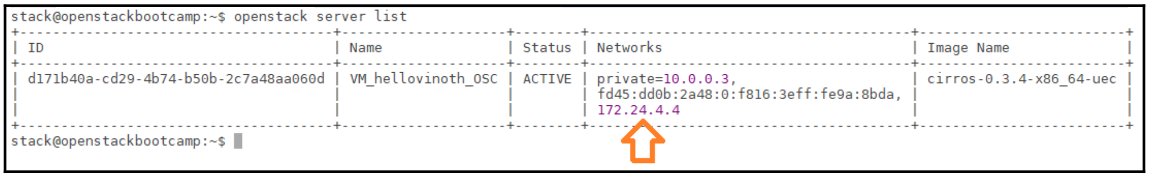
```
stack@openstackbootcamp:~$ openstack floating ip create public
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| created_at          | 2017-09-24T16:12:32Z                 |
| description         |                                      |
| fixed_ip_address    | None                                 |
| floating_ip_address | 172.24.4.4                           |
| floating_network_id | c1284c1e-1bad-4e3c-990d-eb05fd2ad2c4 |
| id                  | 565827b2-140f-4c0d-9c12-b5ac7306a6b9 |
| name                | None                                 |
| port_id             | None                                 |
| project_id          | a103a240c5b643f085af45390dde02f1     |
| revision_number     | 1                                    |
| router_id           | None                                 |
| status              | DOWN                                 |
| updated_at          | 2017-09-24T16:12:32Z                 |
+---------------------+--------------------------------------+
stack@openstackbootcamp:~$
```

2. Then, associate the newly allocated floating IP address to the instance you want to access remotely:



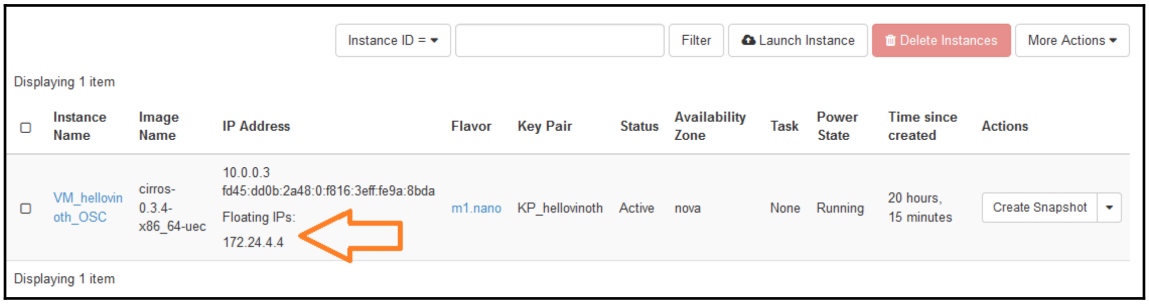
```
stack@openstackbootcamp:~$ openstack server add floating ip VM_hellovinoth_OSC 172.24.4.4
stack@openstackbootcamp:~$
```

3. Check the status of the associated floating IP address:



```
stack@openstackbootcamp:~$ openstack server list
+--------------------------------------+--------------------+--------+---------------------------------------+-------------------------+
| ID                                   | Name               | Status | Networks                              | Image Name              |
+--------------------------------------+--------------------+--------+---------------------------------------+-------------------------+
| d171b40a-cd29-4b74-b50b-2c7a48aa060d | VM_hellovinoth_OSC | ACTIVE | private=10.0.0.3,                     | cirros-0.3.4-x86_64-uec |
|                                      |                    |        | fd45:dd0b:2a48:0:f816:3eff:fe9a:8bda, |                         |
|                                      |                    |        | 172.24.4.4                            |                         |
+--------------------------------------+--------------------+--------+---------------------------------------+-------------------------+
stack@openstackbootcamp:~$
```

4. We could also see the same changes reflecting in the Horizon dashboard under the **Instance** tab:

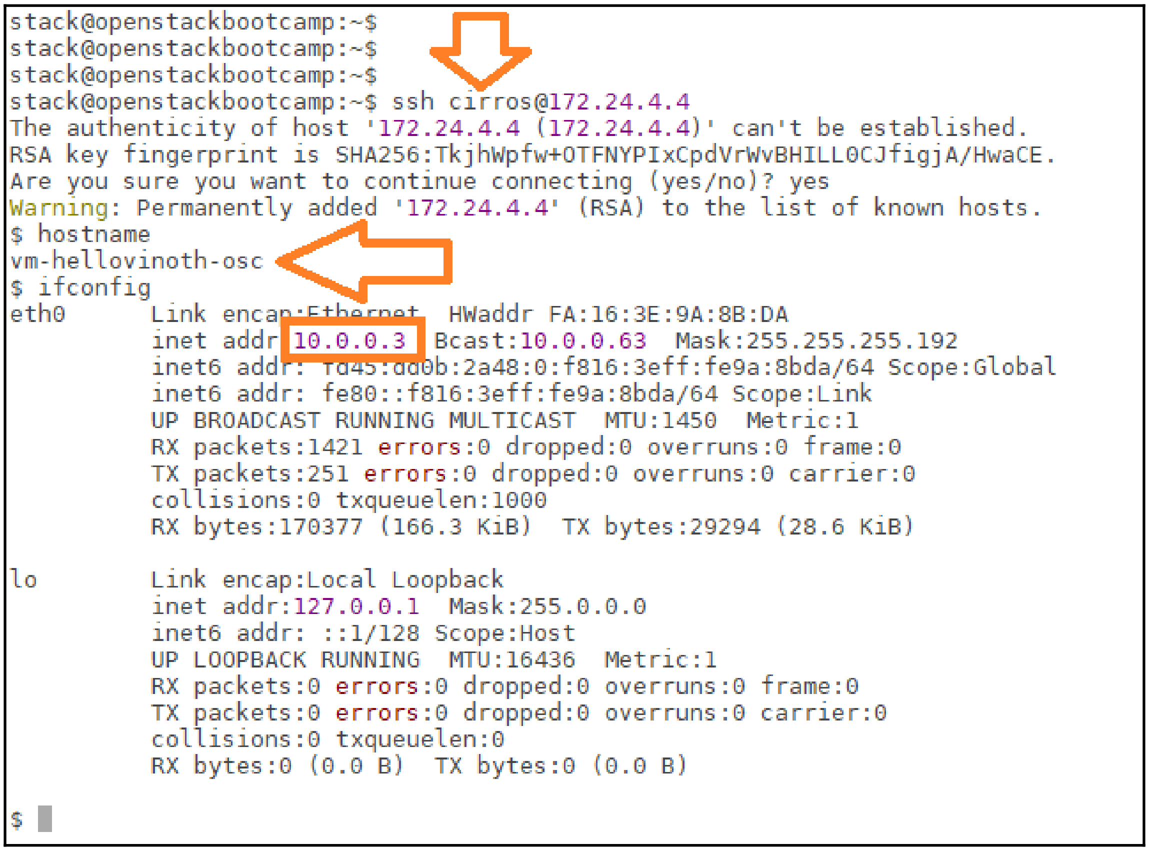


Displaying 1 item

☐	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
☐	VM_hellovinoth_OSC	cirros-0.3.4-x86_64-uec	10.0.0.3 fd45:dd0b:2a48:0:f816:3eff:fe9a:8bda Floating IPs: 172.24.4.4	m1.nano	KP_hellovinoth	Active	nova	None	Running	20 hours, 15 minutes	Create Snapshot ▾

Displaying 1 item

5. Now, try to access your instance using SSH from the controller node:



```
stack@openstackbootcamp:~$
stack@openstackbootcamp:~$
stack@openstackbootcamp:~$
stack@openstackbootcamp:~$ ssh cirros@172.24.4.4
The authenticity of host '172.24.4.4 (172.24.4.4)' can't be established.
RSA key fingerprint is SHA256:TkjhWpfw+OTFNYPIxCpdVrWvBHILL0CJfigjA/HwaCE.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.24.4.4' (RSA) to the list of known hosts.
$ hostname
vm-hellovinoth-osc
$ ifconfig
eth0      Link encap:Ethernet  HWaddr FA:16:3E:9A:8B:DA
          inet addr:10.0.0.3  Bcast:10.0.0.63  Mask:255.255.255.192
          inet6 addr: fd45:dd0b:2a48:0:f816:3eff:fe9a:8bda/64 Scope:Global
          inet6 addr: fe80::f816:3eff:fe9a:8bda/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:1421 errors:0 dropped:0 overruns:0 frame:0
          TX packets:251 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:170377 (166.3 KiB)  TX bytes:29294 (28.6 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

$
```

Usually, we should be able to access the instance via SSH through the floating IP address from any host on the physical provider network. However, the DevStack installation has some limitations that allows the users to connect the virtual machines only from the controller node.

Once you get into the instance via SSH, you could verify the hostname of the instance and the private IP address. As you can notice from the preceding screenshot, the floating IP address will not reflect anywhere inside the instance. This is because the floating IP is associated with the router (NAT).

To exit the `cirros` instance shell and get back to the controller node, execute the following command:

```
exit
```

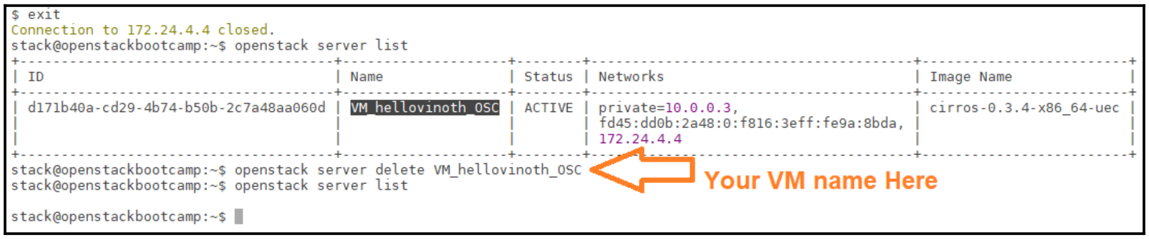
If your virtual machine does not launch or is unable to access as you expect, then jump to `Chapter 9`, *Day 9 - Repair OpenStack* to find the fix.

Terminating the created instance

Like most cases, destroying something is an easy process on comparing the creating process. The same theory applies here, too. We could delete the instance using the following command:

```
openstack server delete VM_hellovinoth_OSC
```

The following screenshot shows the reference output for the preceding server deletion command:



```
$ exit
Connection to 172.24.4.4 closed.
stack@openstackbootcamp:~$ openstack server list
+--------------------------------------+--------------------+--------+------------------------------------------+--------------------------+
| ID                                   | Name               | Status | Networks                                 | Image Name               |
+--------------------------------------+--------------------+--------+------------------------------------------+--------------------------+
| d171b40a-cd29-4b74-b50b-2c7a48aa060d | VM_hellovinoth_OSC | ACTIVE | private=10.0.0.3,                        | cirros-0.3.4-x86_64-uec  |
|                                      |                    |        | fd45:dd0b:2a48:0:f816:3eff:fe9a:8bda,    |                          |
|                                      |                    |        | 172.24.4.4                               |                          |
+--------------------------------------+--------------------+--------+------------------------------------------+--------------------------+
stack@openstackbootcamp:~$ openstack server delete VM_hellovinoth_OSC
stack@openstackbootcamp:~$ openstack server list

stack@openstackbootcamp:~$
```

Note that the preceding command to delete the instance provides no output. So, we could verify the server status using the OpenStack server list command or check the **Instance** tab in the Horizon dashboard.

Congratulate yourself! You have completed the next level in managing the compute service through OpenStack CLI.

Checkpoint

- Get familiar with OpenStack CLI
- Provision a new instance via OpenStack CLI
- Access the newly provisioned VM using SSH login
- Terminate the newly provisioned VM

Image service (Glance)

In this session, we are going to accomplish operations around OpenStack Image Service (Glance) using both the Horizon dashboard and **OpenStack Client** (**OSC**):

Goal	Use the OpenStack Glance image service to add images
Activities	• Use Horizon dashboard to add a CentOS image • Use the command-line to add an Ubuntu image • Launch an instance from the new image

Adding an image using Horizon

Like I mentioned in the previous chapter, the images stored in the Glance service have a pre-installed operating system and the necessary software to support, cloud environment such as a cloud-init tool. The creation of an image from scratch is not in the scope of this chapter. Alternatively, we could download any operating system in OpenStack supported image format as all the major OS distributions maintain their own public repository for their cloud images.

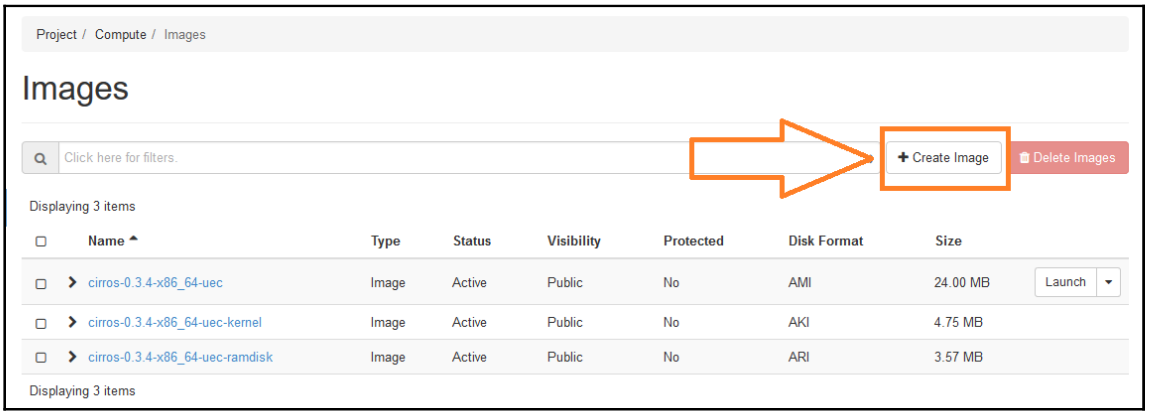
More information on manually building the cloud image and downloading the cloud image is available at: `https://docs.openstack.org/image-guide/create-images-manually.html`.

Before we begin, use the following link to download the CentOS-7 cloud image from the CentOS repository:
`https://cloud.centos.org/centos/7/images/CentOS-7-x86_64-GenericCloud.qcow2`.

When adding an image to Glance, you must specify some mandatory information about the image. The provided information about the Glance image, such as disk format and architecture type, helps the Nova scheduler to choose the appropriate hypervisor to provision the virtual machines.

Follow these instructions to upload an image into the Glance repository using OpenStack Horizon UI:

1. Log in to Horizon using **demo** or any user with `_member_` role credentials.
2. Click the **Images** menu in the left navigation panel on the **Project** tab, which will list the available images for the **demo** project:



Displaying 3 items

Name	Type	Status	Visibility	Protected	Disk Format	Size
cirros-0.3.4-x86_64-uec	Image	Active	Public	No	AMI	24.00 MB
cirros-0.3.4-x86_64-uec-kernel	Image	Active	Public	No	AKI	4.75 MB
cirros-0.3.4-x86_64-uec-ramdisk	Image	Active	Public	No	ARI	3.57 MB

Displaying 3 items

3. Click the **Create Image** button from the top-right corner of the images window. In response, you will get the pop-up window for submitting the image details:

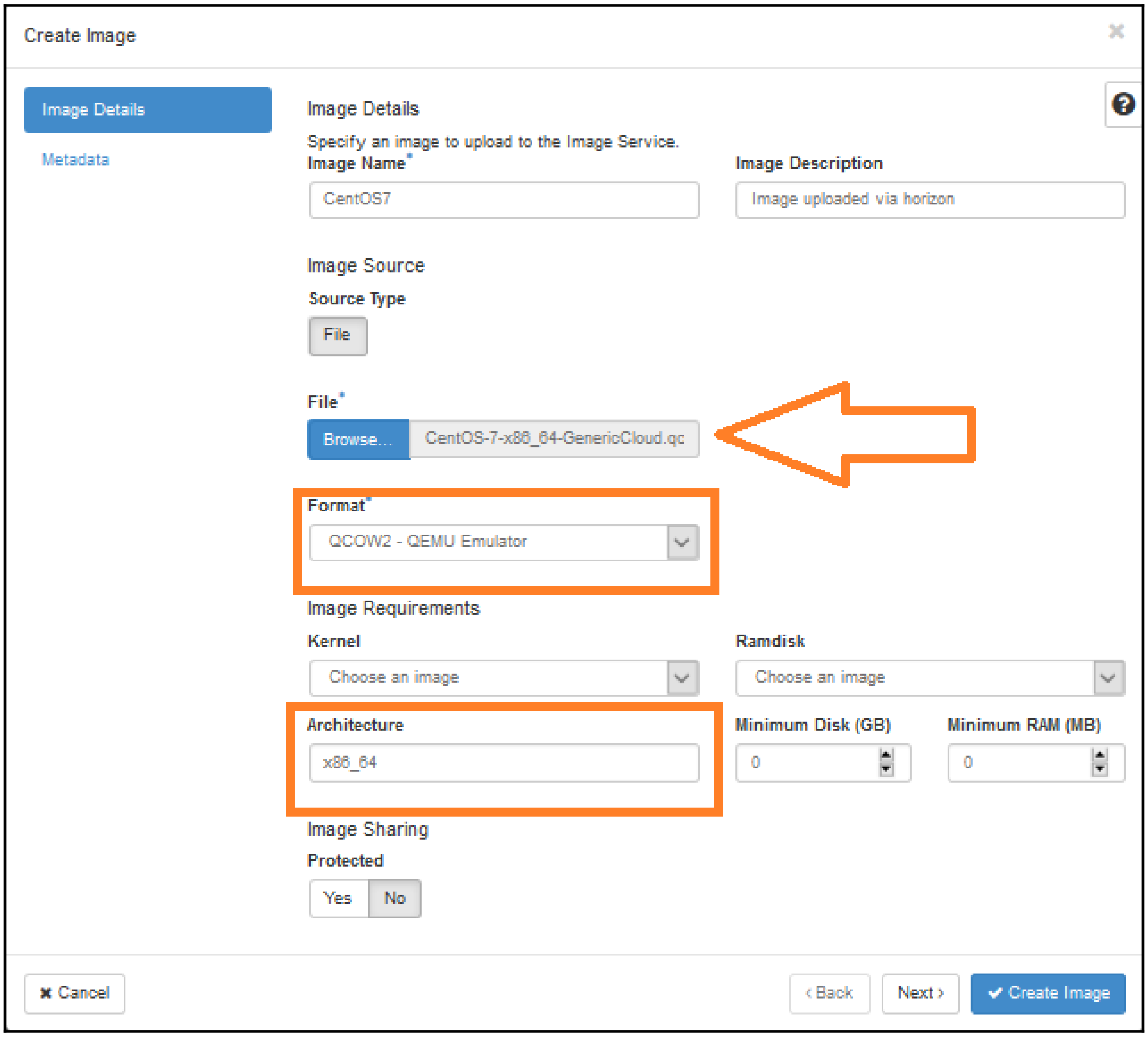

Create Image

Image Details

Specify an image to upload to the Image Service.

Image Name*: CentOS7

Image Description: Image uploaded via horizon

Image Source

Source Type: File

File*: Browse... CentOS-7-x86_64-GenericCloud.qc

Format*: QCOW2 - QEMU Emulator

Image Requirements

Kernel: Choose an image

Ramdisk: Choose an image

Architecture: x86_64

Minimum Disk (GB): 0

Minimum RAM (MB): 0

Image Sharing

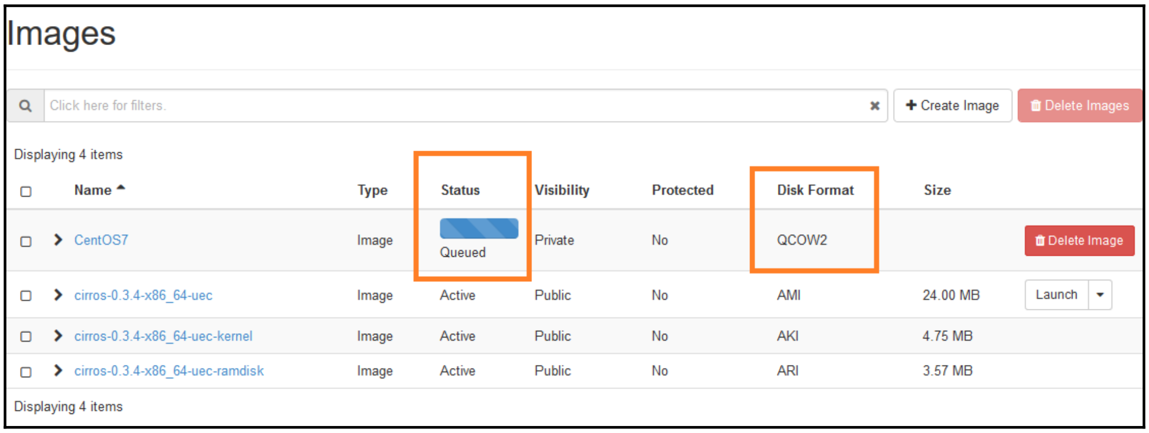
Protected: Yes [✓ No]

4. Mandatory fields are given here:
    - **Image Name***: Name of the image.
    - **File***: Location of the image file from your local host operating system. By default, you could find the CentOS 7 file from the `Downloads` folder of the host machine.
    - **Format***: The disk format of the image. In general, virtual appliance vendors have different formats for laying out the information contained in a virtual machine disk image. The two most popular for KVM hypervisor is `qcow2` and raw disk format.

5. In the **Create Image** window, specify the following values and click the **Create Image** button at the bottom of the form:

Parameter	**Value**
Image Name*	`CentOS 7`
File*	Browse and choose the downloaded CentOS image.
Format*	`QCOW2 – QEMU Emulator`
Leave the rest empty	

6. The image uploading and saving process may take some time depending on the image size and your system performance:


Images

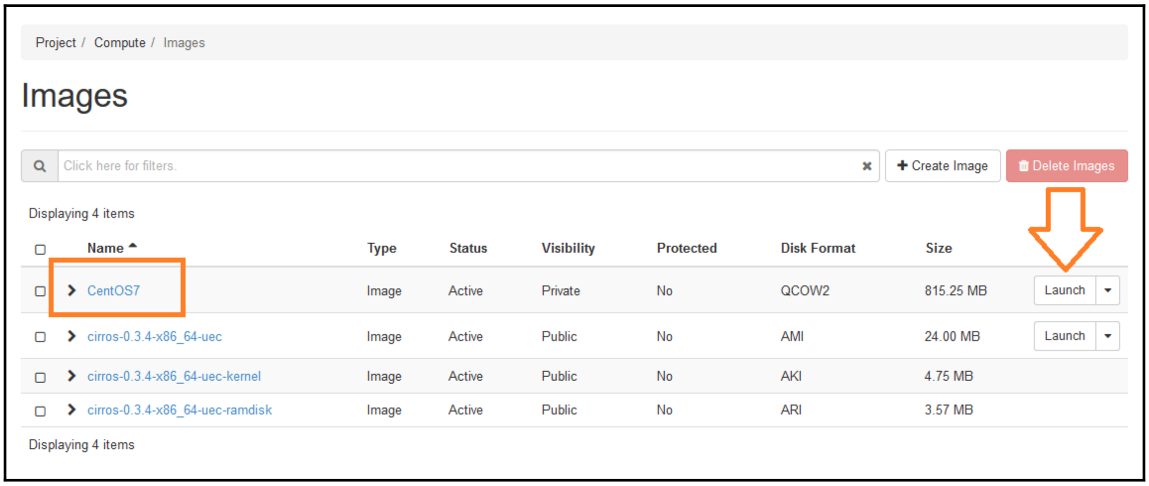
Click here for filters. | + Create Image | Delete Images

Displaying 4 items

Name	Type	Status	Visibility	Protected	Disk Format	Size	
CentOS7	Image	Queued	Private	No	QCOW2		Delete Image
cirros-0.3.4-x86_64-uec	Image	Active	Public	No	AMI	24.00 MB	Launch
cirros-0.3.4-x86_64-uec-kernel	Image	Active	Public	No	AKI	4.75 MB	
cirros-0.3.4-x86_64-uec-ramdisk	Image	Active	Public	No	ARI	3.57 MB	

Displaying 4 items

7. Wait until the status of the image turns from the **Queued** state to the **Active** status. Then, click the **Launch** button from the options panel of the image to launch a CentOS 7 instance to test the image:


Project / Compute / Images

Images

Click here for filters.

Displaying 4 items

	Name	Type	Status	Visibility	Protected	Disk Format	Size	
☐	CentOS7	Image	Active	Private	No	QCOW2	815.25 MB	Launch
☐	cirros-0.3.4-x86_64-uec	Image	Active	Public	No	AMI	24.00 MB	Launch
☐	cirros-0.3.4-x86_64-uec-kernel	Image	Active	Public	No	AKI	4.75 MB	
☐	cirros-0.3.4-x86_64-uec-ramdisk	Image	Active	Public	No	ARI	3.57 MB	

Displaying 4 items

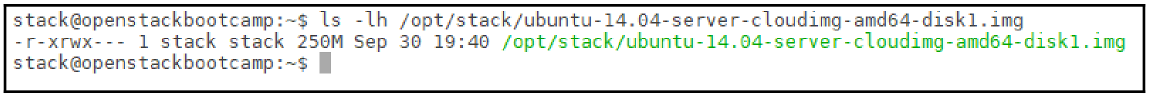
Adding an image using Glance CLI

In this section, we will use **OpenStack Client CLI** (**OSC**) to add the Ubuntu image from a file on the local filesystem:

1. Start an SSH session to your lab environment using SSH shell login credentials.
2. Source the environment file to set environment variables for your current shell. You may go back to the *OpenStack CLI Clients (OSC)* section to get more details on how to perform the particular steps.
3. Execute the following command in lab SSH shell to download the `Ubuntu14_04LTS` cloud image from the Ubuntu repository:

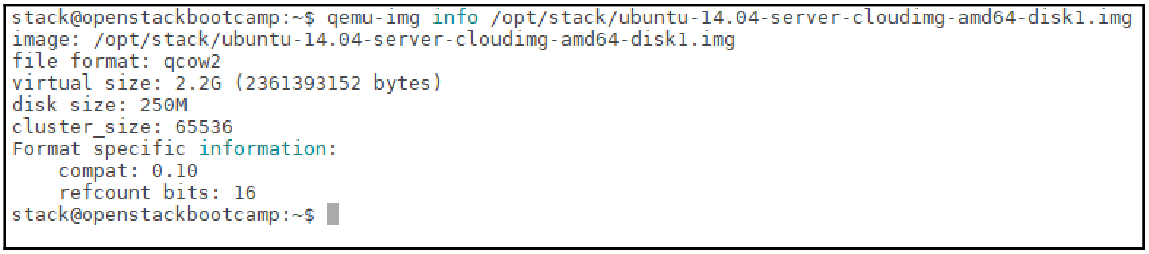
```
wget https://cloud-images.ubuntu.com/releases/14.04.4/release-
20170831/ubuntu-14.04-server-cloudimg-amd64-disk1.img
```

4. Take a close look at the downloaded Ubuntu disk image file for the file permission and disk size using standard Linux commands:



```
stack@openstackbootcamp:~$ ls -lh /opt/stack/ubuntu-14.04-server-cloudimg-amd64-disk1.img
-r-xrwx--- 1 stack stack 250M Sep 30 19:40 /opt/stack/ubuntu-14.04-server-cloudimg-amd64-disk1.img
stack@openstackbootcamp:~$
```

5. Now, look deep into the file properties using the QEMU disk image utility:



```
stack@openstackbootcamp:~$ qemu-img info /opt/stack/ubuntu-14.04-server-cloudimg-amd64-disk1.img
image: /opt/stack/ubuntu-14.04-server-cloudimg-amd64-disk1.img
file format: qcow2
virtual size: 2.2G (2361393152 bytes)
disk size: 250M
cluster_size: 65536
Format specific information:
    compat: 0.10
    refcount bits: 16
stack@openstackbootcamp:~$
```

The attributes you could find in the file properties are:

- `file format`: Is the disk image format
- `virtual_size`: The minimum drive size the VM should have when provisioning the VM using this image
- `disk_size`: The actual file size of the image
- `cluster_size`: With `qcow`, the contents of the image are stored in clusters (blocks)

You should have noticed that the actual file size of an image is smaller than the virtual size of an image, as the copy-on-write is one of the properties of the `qcow2` format.

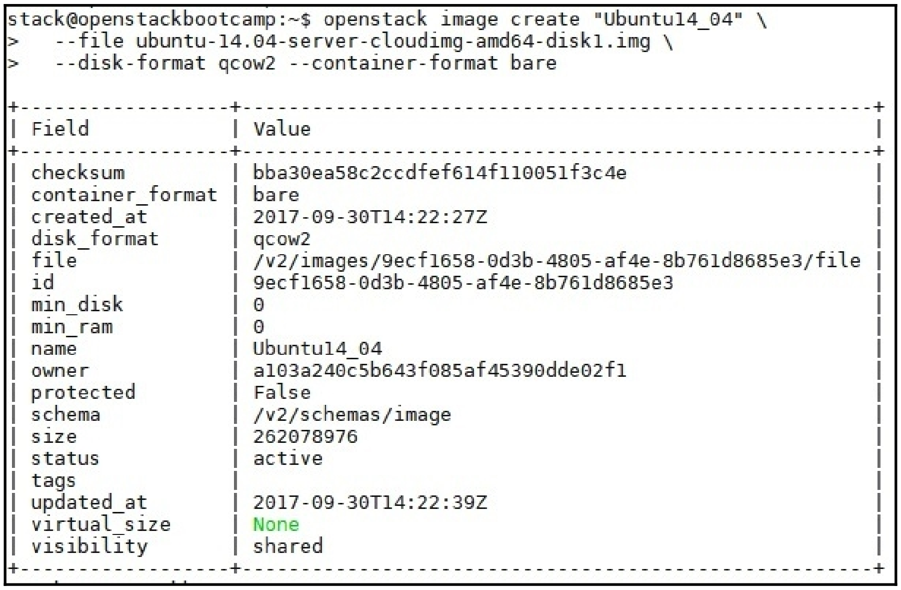
1. Check the available OSC commands related to the Glance image service to see what the available options are that will create an image using the following commands:

   ```
   openstack image --help
   ```

2. Upload the downloaded image to the image service using the following command:

   ```
   openstack image create "Ubuntu14_04" \
     --file ubuntu-14.04-server-cloudimg-amd64-disk1.img \
     --disk-format qcow2 --container-format bare
   ```

3. You may refer to the following screenshot for the reference output of the preceding command:

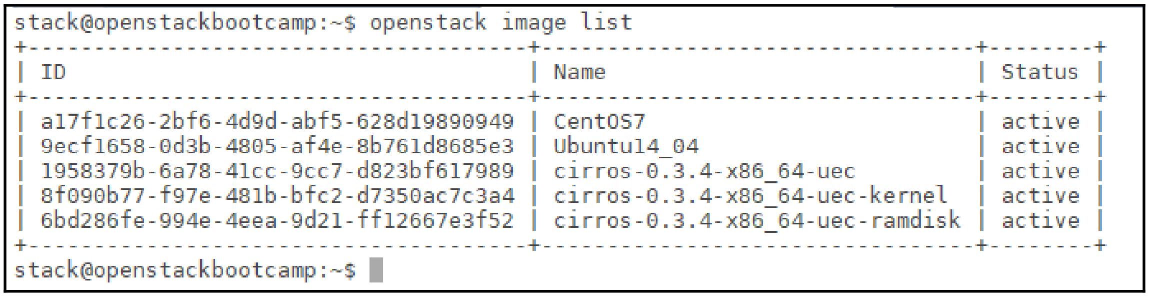


```
stack@openstackbootcamp:~$ openstack image create "Ubuntu14_04" \
>   --file ubuntu-14.04-server-cloudimg-amd64-disk1.img \
>   --disk-format qcow2 --container-format bare

+------------------+------------------------------------------------------+
| Field            | Value                                                |
+------------------+------------------------------------------------------+
| checksum         | bba30ea58c2ccdfef614f110051f3c4e                     |
| container_format | bare                                                 |
| created_at       | 2017-09-30T14:22:27Z                                 |
| disk_format      | qcow2                                                |
| file             | /v2/images/9ecf1658-0d3b-4805-af4e-8b761d8685e3/file |
| id               | 9ecf1658-0d3b-4805-af4e-8b761d8685e3                 |
| min_disk         | 0                                                    |
| min_ram          | 0                                                    |
| name             | Ubuntu14_04                                          |
| owner            | a103a240c5b643f085af45390dde02f1                     |
| protected        | False                                                |
| schema           | /v2/schemas/image                                    |
| size             | 262078976                                            |
| status           | active                                               |
| tags             |                                                      |
| updated_at       | 2017-09-30T14:22:39Z                                 |
| virtual_size     | None                                                 |
| visibility       | shared                                               |
+------------------+------------------------------------------------------+
```

4. In the preceding command, we have passed the parameter value to set the image name as `Ubuntu14_04`, the file location as the downloaded Ubuntu cloud image location, and followed by the disk format as `qcow2` and the container format as bare.
5. Verify that the uploaded image was successfully saved in the Glance service by executing the following command:

```
openstack image list
```

Refer to the following screenshot for the output reference:



```
stack@openstackbootcamp:~$ openstack image list
+--------------------------------------+---------------------------------+--------+
| ID                                   | Name                            | Status |
+--------------------------------------+---------------------------------+--------+
| a17f1c26-2bf6-4d9d-abf5-628d19890949 | CentOS7                         | active |
| 9ecf1658-0d3b-4805-af4e-8b761d8685e3 | Ubuntu14_04                     | active |
| 1958379b-6a78-41cc-9cc7-d823bf617989 | cirros-0.3.4-x86_64-uec         | active |
| 8f090b77-f97e-481b-bfc2-d7350ac7c3a4 | cirros-0.3.4-x86_64-uec-kernel  | active |
| 6bd286fe-994e-4eea-9d21-ff12667e3f52 | cirros-0.3.4-x86_64-uec-ramdisk | active |
+--------------------------------------+---------------------------------+--------+
stack@openstackbootcamp:~$
```

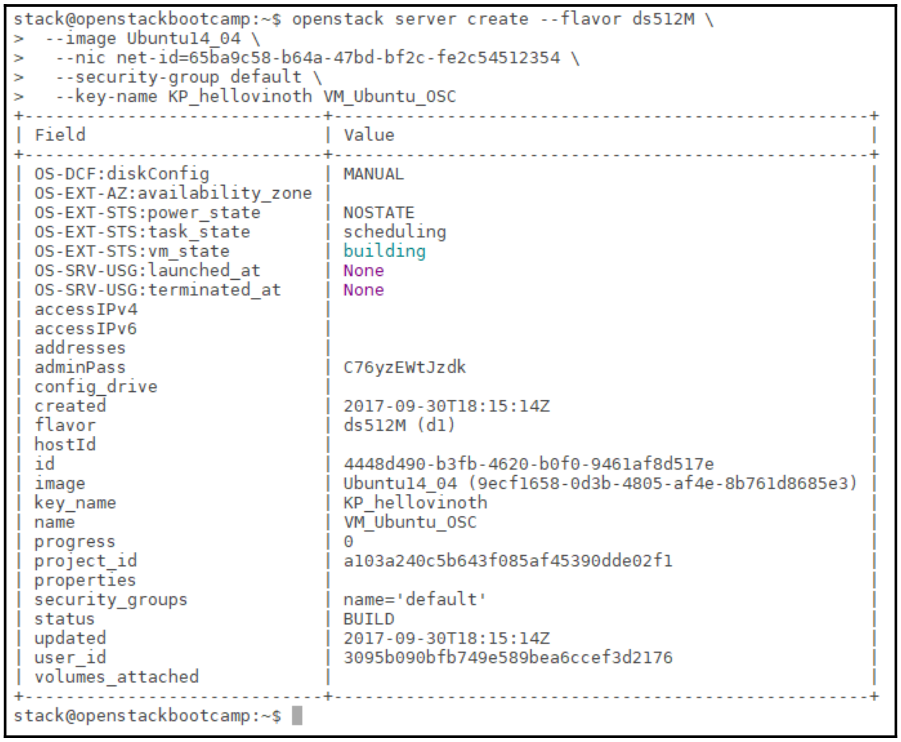
You should see `Ubuntu14_04` in a list of images available with an `active` status.

More information about the OpenStack image create parameters are available at:
`https://docs.openstack.org/user-guide/common/cli-manage-images.html`.

Detailed information about disk and container formats for images are available at:
`https://docs.openstack.org/image-guide/image-formats.html`.

Launching an instance from the new image

Let's use the OSC CLI to provision a new Ubuntu virtual machine from the recently uploaded `Ubuntu14_04` cloud image. As we already discussed the OSC CLI to create a new virtual machine in the previous session, let us quickly create a new instance to test the recently uploaded Ubuntu cloud image:

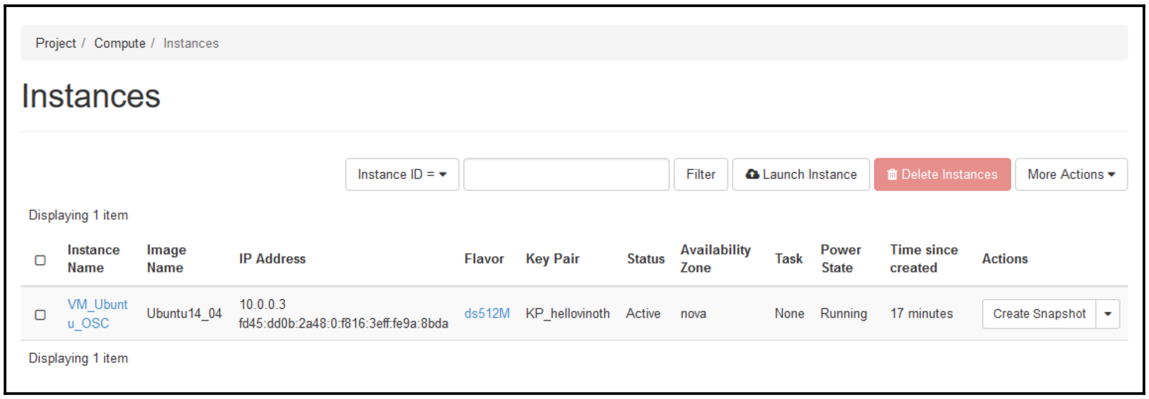


```
stack@openstackbootcamp:~$ openstack server create --flavor ds512M \
>  --image Ubuntu14_04 \
>   --nic net-id=65ba9c58-b64a-47bd-bf2c-fe2c54512354 \
>   --security-group default \
>   --key-name KP_hellovinoth VM_Ubuntu_OSC
+-----------------------------+-----------------------------------------------------+
| Field                       | Value                                               |
+-----------------------------+-----------------------------------------------------+
| OS-DCF:diskConfig           | MANUAL                                              |
| OS-EXT-AZ:availability_zone |                                                     |
| OS-EXT-STS:power_state      | NOSTATE                                             |
| OS-EXT-STS:task_state       | scheduling                                          |
| OS-EXT-STS:vm_state         | building                                            |
| OS-SRV-USG:launched_at      | None                                                |
| OS-SRV-USG:terminated_at    | None                                                |
| accessIPv4                  |                                                     |
| accessIPv6                  |                                                     |
| addresses                   |                                                     |
| adminPass                   | C76yzEWtJzdk                                        |
| config_drive                |                                                     |
| created                     | 2017-09-30T18:15:14Z                                |
| flavor                      | ds512M (d1)                                         |
| hostId                      |                                                     |
| id                          | 4448d490-b3fb-4620-b0f0-9461af8d517e                |
| image                       | Ubuntu14_04 (9ecf1658-0d3b-4805-af4e-8b761d8685e3)  |
| key_name                    | KP_hellovinoth                                      |
| name                        | VM_Ubuntu_OSC                                       |
| progress                    | 0                                                   |
| project_id                  | a103a240c5b643f085af45390dde02f1                    |
| properties                  |                                                     |
| security_groups             | name='default'                                      |
| status                      | BUILD                                               |
| updated                     | 2017-09-30T18:15:14Z                                |
| user_id                     | 3095b090bfb749e589bea6ccef3d2176                    |
| volumes_attached            |                                                     |
+-----------------------------+-----------------------------------------------------+
stack@openstackbootcamp:~$
```

You can see that from the preceding screenshot that I have submitted a new instance creation request with the following parameters:

- `flavor`: `ds512M`
- `image`: `Ubuntu14_04`
- NIC [Network ID]: Private network's ID
- `security_group`: `name='default'`
- `key_name`: `KP_hellovinoth`
- **Instance name**: `VM_Ubuntu_OSC`

You can see the new instance is created with the parameter mentioned previously from the OpenStack Horizon dashboard:



Project / Compute / Instances

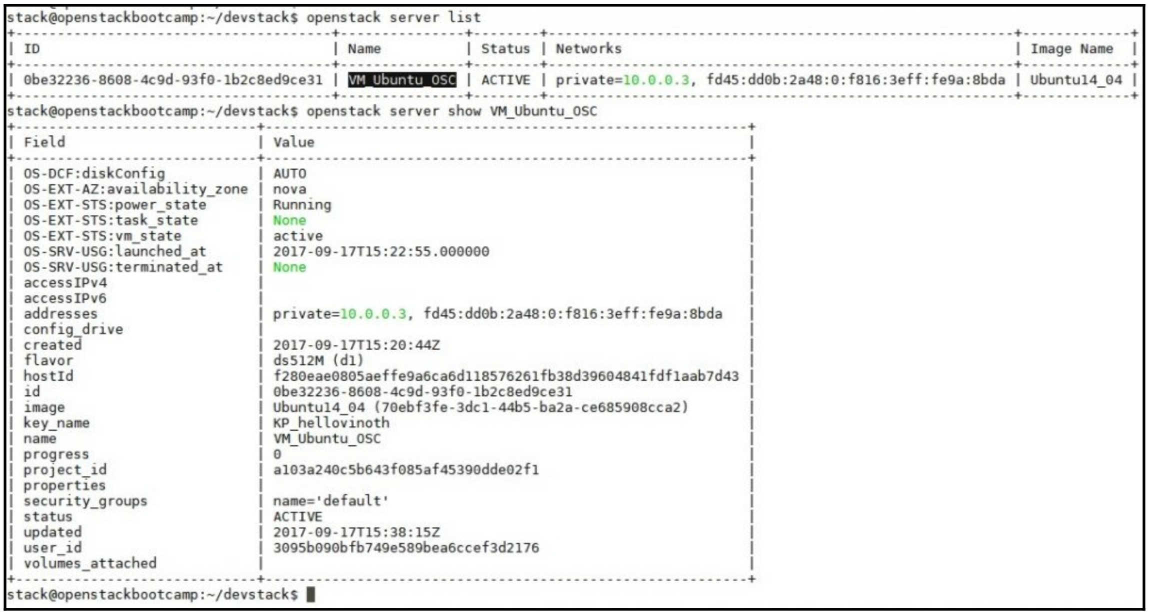
Instances

Displaying 1 item

	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
☐	VM_Ubuntu_OSC	Ubuntu14_04	10.0.0.3 fd45:dd0b:2a48:0:f816:3eff:fe9a:8bda	ds512M	KP_hellovinoth	Active	nova	None	Running	17 minutes	Create Snapshot

Displaying 1 item

Alternatively, you could also use the following CLI to see the detailed information about the new virtual machine:



```
stack@openstackbootcamp:~/devstack$ openstack server list
+--------------------------------------+---------------+--------+--------------------------------------------------------+-------------+
| ID                                   | Name          | Status | Networks                                               | Image Name  |
+--------------------------------------+---------------+--------+--------------------------------------------------------+-------------+
| 0be32236-8608-4c9d-93f0-1b2c8ed9ce31 | VM_Ubuntu_OSC | ACTIVE | private=10.0.0.3, fd45:dd0b:2a48:0:f816:3eff:fe9a:8bda | Ubuntu14_04 |
+--------------------------------------+---------------+--------+--------------------------------------------------------+-------------+
stack@openstackbootcamp:~/devstack$ openstack server show VM_Ubuntu_OSC
+-----------------------------+------------------------------------------------------------------+
| Field                       | Value                                                            |
+-----------------------------+------------------------------------------------------------------+
| OS-DCF:diskConfig           | AUTO                                                             |
| OS-EXT-AZ:availability_zone | nova                                                             |
| OS-EXT-STS:power_state      | Running                                                          |
| OS-EXT-STS:task_state       | None                                                             |
| OS-EXT-STS:vm_state         | active                                                           |
| OS-SRV-USG:launched_at      | 2017-09-17T15:22:55.000000                                       |
| OS-SRV-USG:terminated_at    | None                                                             |
| accessIPv4                  |                                                                  |
| accessIPv6                  |                                                                  |
| addresses                   | private=10.0.0.3, fd45:dd0b:2a48:0:f816:3eff:fe9a:8bda           |
| config_drive                |                                                                  |
| created                     | 2017-09-17T15:20:44Z                                             |
| flavor                      | ds512M (d1)                                                      |
| hostId                      | f280eae0805aeffe9a6ca6d118576261fb38d39604841fdf1aab7d43         |
| id                          | 0be32236-8608-4c9d-93f0-1b2c8ed9ce31                             |
| image                       | Ubuntu14_04 (70ebf3fe-3dc1-44b5-ba2a-ce685908cca2)               |
| key_name                    | KP_hellovinoth                                                   |
| name                        | VM_Ubuntu_OSC                                                    |
| progress                    | 0                                                                |
| project_id                  | a103a240c5b643f085af45390dde02f1                                 |
| properties                  |                                                                  |
| security_groups             | name='default'                                                   |
| status                      | ACTIVE                                                           |
| updated                     | 2017-09-17T15:38:15Z                                             |
| user_id                     | 3095b090bfb749e589bea6ccef3d2176                                 |
| volumes_attached            |                                                                  |
+-----------------------------+------------------------------------------------------------------+
stack@openstackbootcamp:~/devstack$ 
```

Checkpoint

- Use Horizon dashboard to add a CentOS image
- Use the command line to add an Ubuntu image
- Launch an instance from the new image

Block storage (Cinder)

By default, OpenStack creates virtual machines with local storage for the root partition, which means that all of the data will get stored in the local storage of the host machine. When the user deletes the virtual machine, all of the data that resides on the / partition will also get deleted. In specific cases, the user may need to store all of the vital data in persistent storage, which will be available even after virtual deletion. For that, Cinder provides a reliable way to store vital data, such as database file, and application data in persistent storage by mounting Cinder volume under a selected directory such as `/var` or `/mnt`, or by creating a virtual machine from Cinder volume.

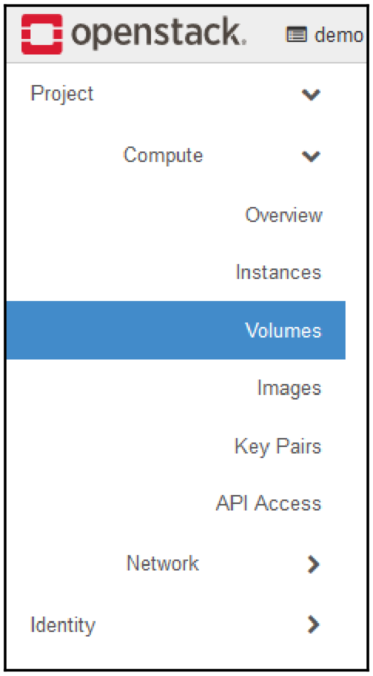
In this session, we are going to perform operations around OpenStack block storage service (Cinder) using both the Horizon dashboard and command-line client:

Goal	Create volumes using the Cinder (block storage) service
Activities	• Create a volume using Horizon • Attach the volume to the existing virtual machine • Verify the attached volume from the instance • Detach the volume from the instance

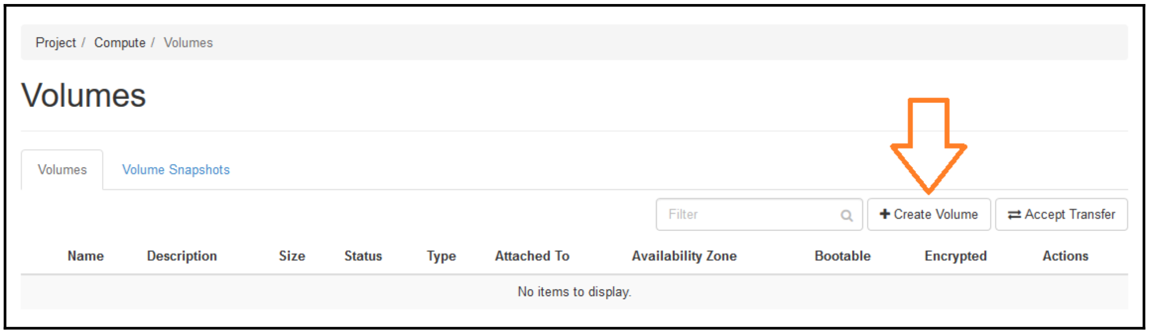
Managing Cinder volume using Horizon

Follow these instructions to create a new volume in Cinder:

1. Click the **Volumes** menu in the left navigation panel on the **Project** tab in Horizon to see what volumes are available for the project **demo**:



2. Click the **+Create Volume** button in the top right corner of the **Volumes** panel:

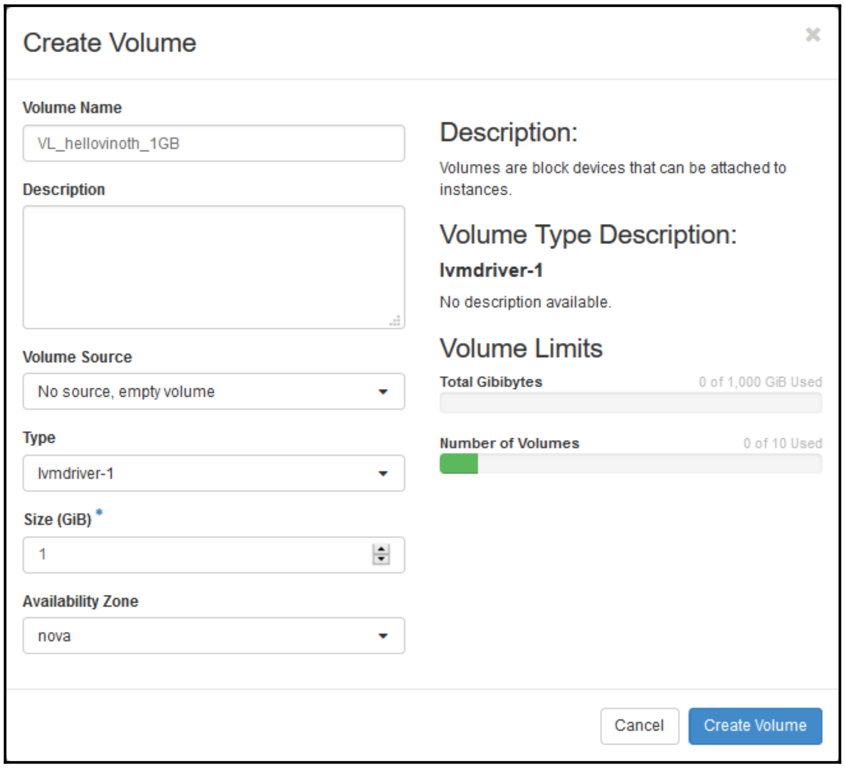


Project / Compute / Volumes

Volumes

Name	Description	Size	Status	Type	Attached To	Availability Zone	Bootable	Encrypted	Actions
No items to display.									

3. In response, the **Create Volume** window will be displayed with the **Volume Source** option set to **No source, empty volume** by default:



Create Volume

Volume Name: VL_hellovinoth_1GB

Description:

Volume Source: No source, empty volume

Type: lvmdriver-1

Size (GiB) *: 1

Availability Zone: nova

Description:

Volumes are block devices that can be attached to instances.

Volume Type Description:

lvmdriver-1

No description available.

Volume Limits

Total Gibibytes 0 of 1,000 GiB Used

Number of Volumes 0 of 10 Used

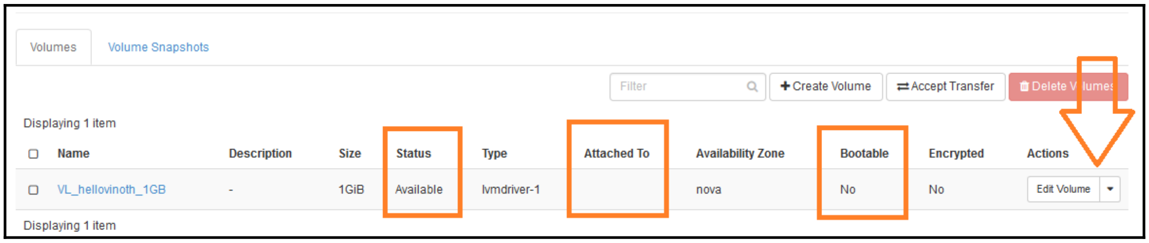
The field area for the create volume pop-up has the following options:

- **Volume Name** and **Description**: Name of the volume for your reference.
- **Volume Source**: **No source, empty volume**: It will create a new non-partitioned volume on the back-end storage. The first time when you use/mount the volume to the operating system, you need to format and partition it.
- **Volume**: With Cinder, you can take snapshots (complete copies of the existing volumes) and then create the new volumes from it.
- **Image**: Cinder can create a volume from the image available through Glance. That is an easy way to create a bootable volume.
- **Type**: The **Type** field allows you to choose a specific storage backend for multi-backend Cinder configuration. Our DevStack environment only supports LVM backend, so you can see that the field has only one option available as **lvmdriver-1**.
- **Size (GiB)***: Based on the quota limit for the selected project, you could create a volume of any size.
- **Availability Zone**: You could select between different availability zones to create new volumes. However, our DevStack has only one default availability zone configured called **nova**.

4. In the **Create Volume** window, specify the following values:

Parameter	**Value**
Volume_Name	`VL_hellovinoth_1GB`
Volume_Source	**No source, empty volume**
Type	**lvmdriver-1**
Size	**1**
Availability Zone	**nova**

5. Click the **+Create Volume** button and pay attention to how the **Status** field changes for the new volume creation:

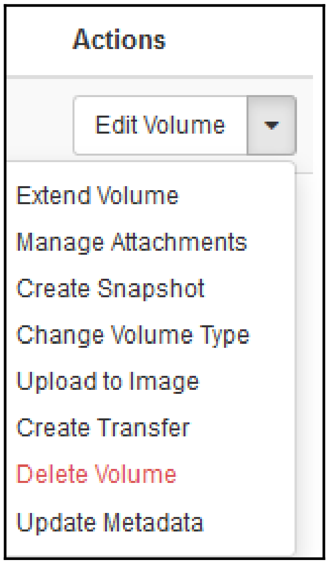


Displaying 1 item

	Name	Description	Size	Status	Type	Attached To	Availability Zone	Bootable	Encrypted	Actions
▢	VL_hellovinoth_1GB	-	1GiB	Available	lvmdriver-1		nova	No	No	Edit Volume

Displaying 1 item

6. Let's test the volume by attaching it to an instance that, we have created during the previous session. Before that let's explore the available options for managing the Cinder volumes.
7. Click the drop-down button next to the **Edit Volume** button on the right side of the volume list:



The drop-down field has the following listed options available:

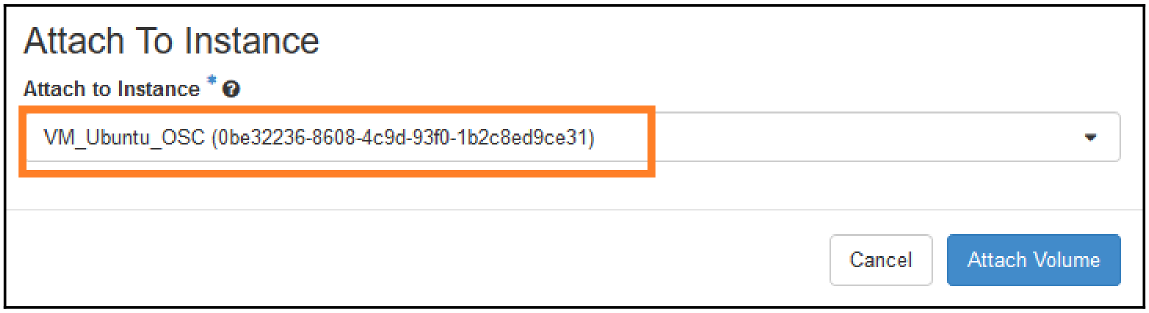
- **Extend Volume**: Allows the user to increase the size of the existing volume. Notably, we cannot reduce the size of the volumes.
- **Manage Attachments**: Allows the user to attach/detach the volume from the virtual machines.
- **Create Snapshot**: Enables the user to create a complete copy of the existing volume called snapshot. This snapshot can be later used to as a volume source while creating a new volume.

- **Change Volume Type**: Permits the user to change the volume backend driver. Our Devstack setup has only one backend Cinder driver configured by default. So, this option will not be available for now.
- **Upload to Image**: Enables the user to upload the volume to the image service as an image.
- **Create Transfer**: Ownership of a volume can be transferred from one project to another. Once a volume transfer is created in a donor project, it then can be accepted by a recipient project. The Transfer ID and the authorization key are required by the receiver to accept the transfer.
- **Delete Volume**: Will delete the volume permanently.
- **Update Metadata**: Allows the user to specify the resource metadata.

Attaching Cinder volume to the virtual machine

Follow the steps to attach the volume to the existing Ubuntu instance that we have created during the *Launch an instance from the new image* session:

1. Click the **Manage Attachments** option to attach the volume to the instance. Please note that this is a continuation of *step 6*, mentioned in the previous session, *Manage Cinder volume using Horizon*:

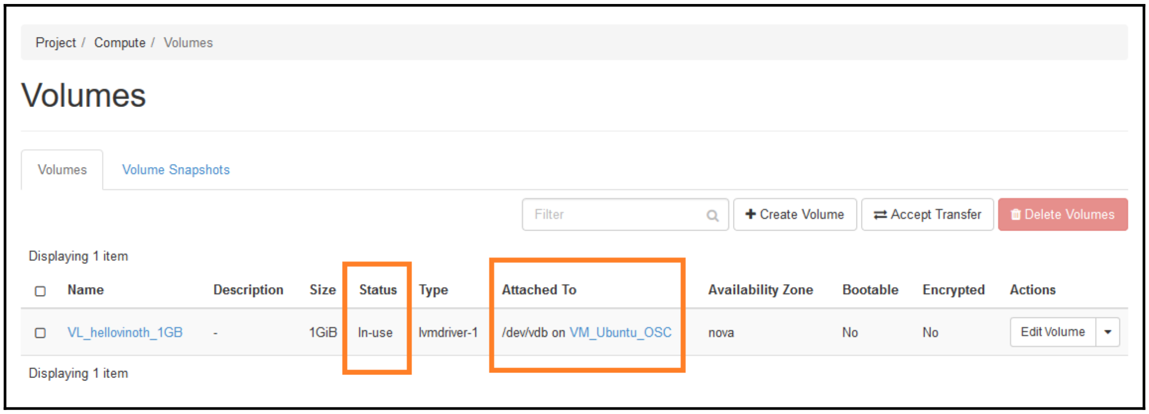


Attach To Instance

Attach to Instance *

VM_Ubuntu_OSC (0be32236-8608-4c9d-93f0-1b2c8ed9ce31)

2. In response, you will get the **Manage volume Attachments** pop-up window. From the **Attach to Instance*** field drop-down button, you need to select the virtual machine to attach the volume. Then, hit the **Attach Volume** button to submit the volume attachment request to the Cinder service.
3. Shortly, you could see the **Volumes** window changes the volume status to **In-use** and has attached to the selected instance:


Project / Compute / Volumes

Volumes

Displaying 1 item

Name	Description	Size	Status	Type	Attached To	Availability Zone	Bootable	Encrypted	Actions
VL_hellovinoth_1GB	-	1GiB	In-use	lvmdriver-1	/dev/vdb on VM_Ubuntu_OSC	nova	No	No	Edit Volume

Displaying 1 item

4. Now, let's verify the volume attachment from the virtual machine.
5. To do this, first, we need to log in into the Ubuntu virtual machine, which will include mapping the floating IP for SSH login. If you find any difficulties in connecting the Ubuntu instance via SSH shell, please refer to the preceding session, *Connect to the Instance Using SSH*.

TIP

The Ubuntu cloud image we are using in our lab environment allows only the key-pair based authentication, and the default username for login is `ubuntu`.

6. Use the following command to connect the Ubuntu instance from the controller node:

```
ssh ubuntu@<your_floating_IP_here>
```

7. Once you are inside the Ubuntu instance, you need to switch to the `root` user to access the privileged operating system functionality. Use the following command to switch to the `root` user:

```
sudo su
```

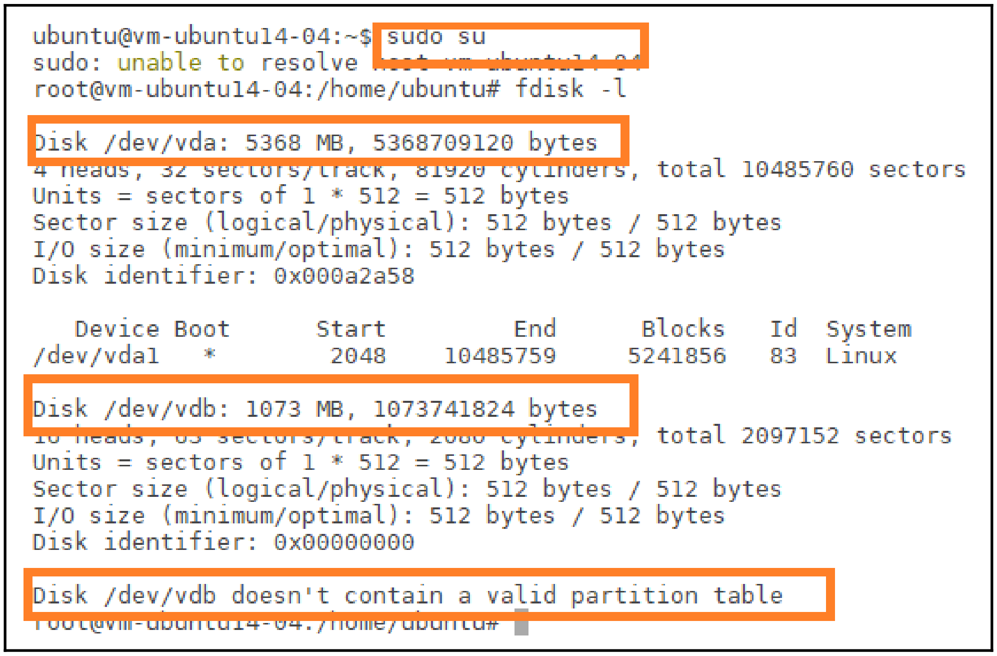
TIP

Remember that we have created this attached volume with the volume source as **No source, empty volume**, which would have created a new non-partitioned volume on the back-end storage. So, the first time when we attach the volume to any operating system, we need to format and partition it manually.

8. Run the following command as the root user to see the attached Cinder volume from the Ubuntu instance:

```
fdisk -l
```

9. In response to the previous command, you will see the output like the one in the following screenshot:



```
ubuntu@vm-ubuntu14-04:~$ sudo su
sudo: unable to resolve host vm-ubuntu14-04
root@vm-ubuntu14-04:/home/ubuntu# fdisk -l

Disk /dev/vda: 5368 MB, 5368709120 bytes
4 heads, 32 sectors/track, 81920 cylinders, total 10485760 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000a2a58

   Device Boot      Start         End      Blocks   Id  System
/dev/vda1   *        2048    10485759     5241856   83  Linux

Disk /dev/vdb: 1073 MB, 1073741824 bytes
16 heads, 63 sectors/track, 2080 cylinders, total 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

Disk /dev/vdb doesn't contain a valid partition table
root@vm-ubuntu14-04:/home/ubuntu#
```

I have highlighted the critical information in the preceding screenshot for your convenience:

- `Disk /dev/vda: 5368 MB, 5368709120 bytes`: This disk represents the 5 GB primary disk for the / partition of the Ubuntu OS, which was created with the flavor `ds512M` while creating the virtual machine
- `Disk /dev/vdb: 1073 MB, 1073747824 bytes`: This disk represents the 1 GB secondary disk, which is attached to the Cinder volume
- `Disk /dev/vdb does not contain a valid partition table`: This represents that the disk is not yet formatted

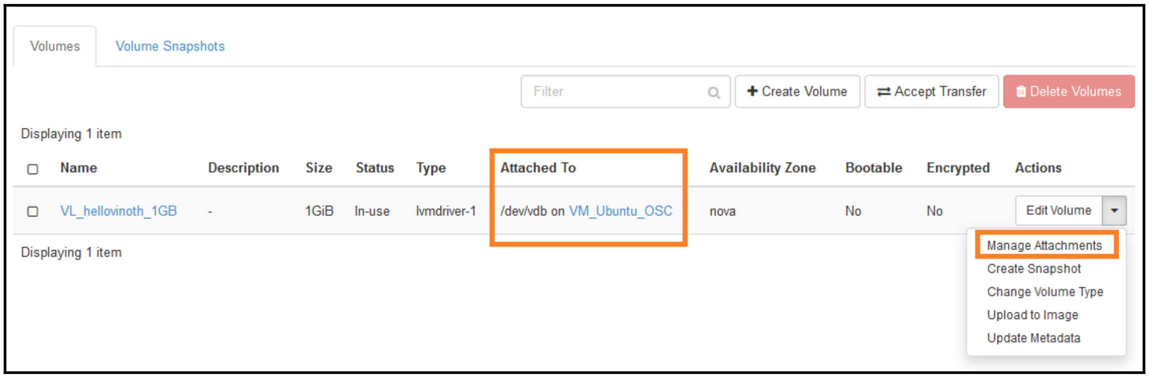
Formatting the disk is not in the scope of our exercise; you may refer any Linux forum to do the disk format and mount.

Since the volume is created from the empty non-partitioned source, we need to do these formatting steps only for the first time when using this volume. In this case, detaching the volume and attaching it again to the same virtual machine or the different virtual machine does not require the disk formatting steps to mount the volume.

Detaching Cinder volume from the virtual machine

Follow these instructions to detach the volume from the virtual machine:

1. Click the **Volumes** menu in the left navigation panel on the **Project** tab in Horizon to see what volumes are available for the project **demo**.
2. Click the drop-down button next to the **Edit Volume** button on the right side of the volume list:



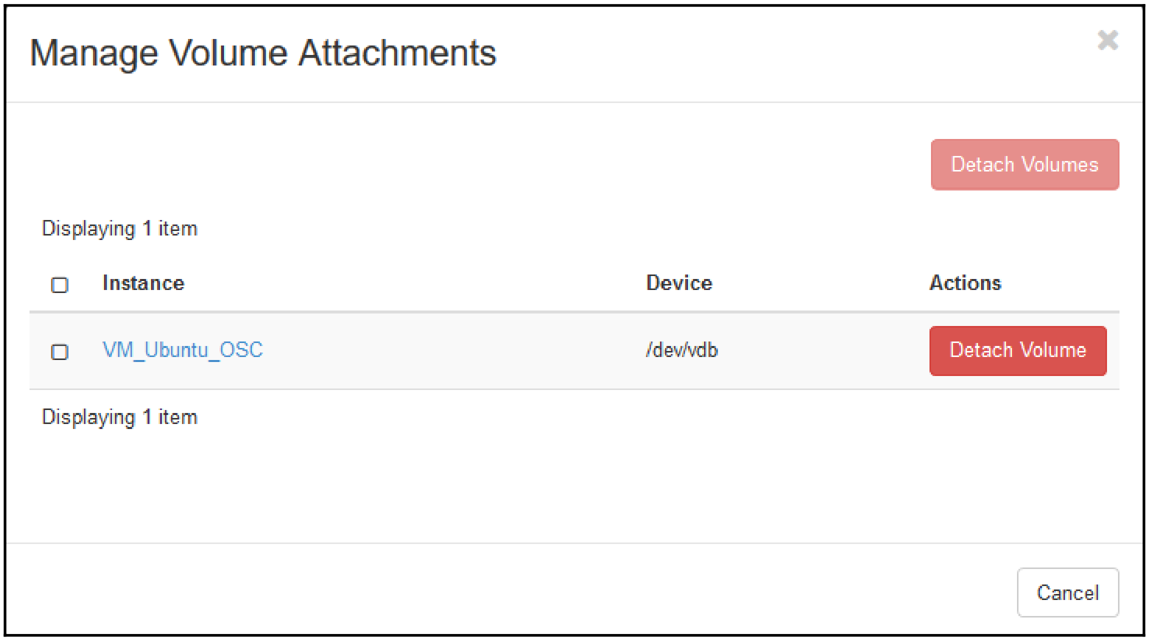
Displaying 1 item

	Name	Description	Size	Status	Type	Attached To	Availability Zone	Bootable	Encrypted	Actions
☐	VL_hellovinoth_1GB	-	1GiB	In-use	lvmdriver-1	/dev/vdb on VM_Ubuntu_OSC	nova	No	No	Edit Volume

Displaying 1 item

3. Choose the **Manage Attachments** button from the drop-down list.

4. In response, you will get the **Manage Volume Attachments** pop-up window, like the one shown in the following screenshot:

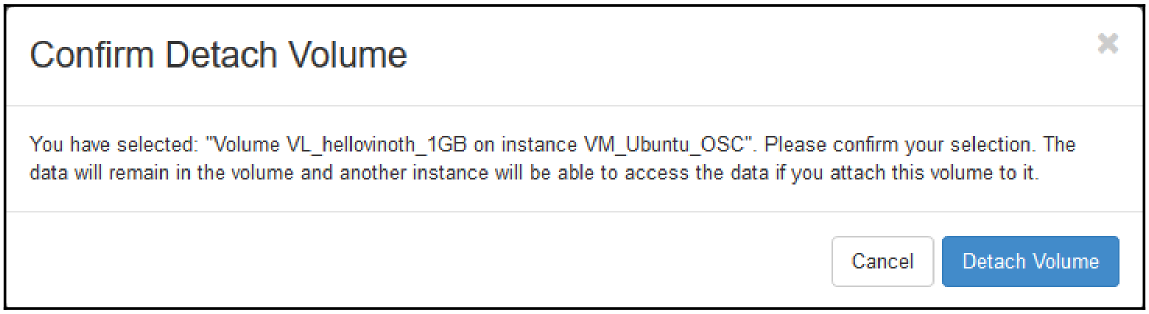


Displaying 1 item

	Instance	Device	Actions
☐	VM_Ubuntu_OSC	/dev/vdb	Detach Volume

Displaying 1 item

5. Click the **Detach Volume** button in the pop-up window to detach the volume from the virtual machine. In response, Horizon will pop-up a **Confirm Detach Volume** window, asking the user to confirm the volume detaching the process:



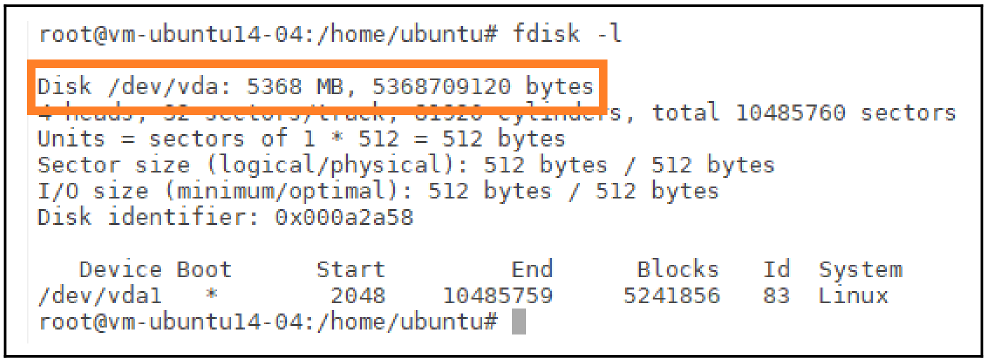
You have selected: "Volume VL_hellovinoth_1GB on instance VM_Ubuntu_OSC". Please confirm your selection. The data will remain in the volume and another instance will be able to access the data if you attach this volume to it.

6. Now, let's check the virtual machine's shell to confirm the volume detachment process.

7. We need to run the following command in the instance's SSH shell. If you find any difficulties in connecting an instance via SSH, please refer to *step 8*.
8. Run the following command as a root user to see the disk information:

```
fdisk -l
```

9. In response to the previous command, you will see the output like the one in the following screenshot:



```
root@vm-ubuntu14-04:/home/ubuntu# fdisk -l

Disk /dev/vda: 5368 MB, 5368709120 bytes
4 heads, 32 sectors/track, 81920 cylinders, total 10485760 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000a2a58

   Device Boot      Start         End      Blocks   Id  System
/dev/vda1   *        2048    10485759     5241856   83  Linux
root@vm-ubuntu14-04:/home/ubuntu# 
```

From the preceding output, you can see only the `/dev/vda` partition of the Ubuntu OS. However, the `Disk /dev/vdb` is missing, as the Cinder service detached the disk `vdb` from the instance.

Checkpoint

- Create a new volume using Horizon
- Attach the volume to the existing virtual machine and verify the attached volume from the instance
- Detach the volume from the instance

Identity (KeyStone)

KeyStone is an OpenStack component that provides identity, catalogue, and policy services. You can compare it to the Active Directory Services for Windows.

In this session, we are going to use the Horizon dashboard to create new projects and users in KeyStone:

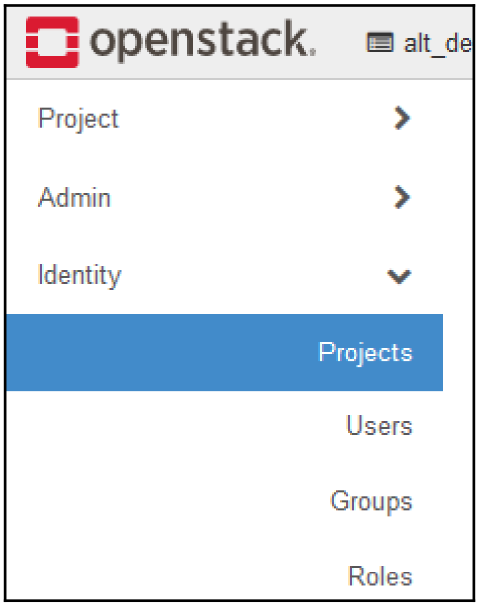
Goal	Use Horizon to add projects/users to the KeyStone identity service
Activities	Adding projects and users

Adding projects and users

So far, we are using the **demo** user account, which has only the member role mapped, and it has no authorization to do the admin activities. In this session, we are about to add new projects and users, which will require admin role access. So, you must log in to the Horizon dashboard as an **admin** user to continue with the upcoming exercises.

By default, in DevStack setup, the user credentials for **admin** login is `admin` / `nomoresecret`. You could also refer to the *With Horizon dashboard* section for your reference:

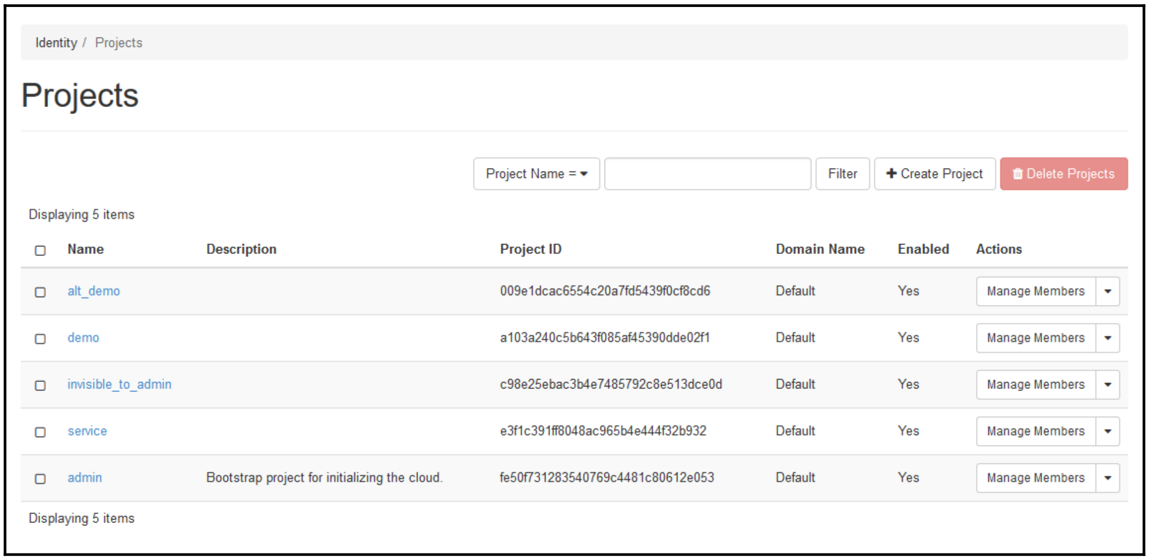
1. Log in to Horizon as the **admin** user. Since you logged in as the user with administrative privileges, the main screen shows both **Project** and **Admin** tabs, with the **Identity** tab being displayed by default:



The tabs available on the main screen are:

- **Project**: This tab enables the **admin** user to have a dedicated project with non-privileged access
- **Admin**: This tab allows you to perform administrative configuration tasks for the whole OpenStack environment
- **Identity**: This tab allows you to add and modify the users and projects to the OpenStack

2. Click the **Projects** menu in the left navigation panel on the **Identity** tab in Horizon to check the list of all available projects in the lab environment, including the **demo** project we used before:

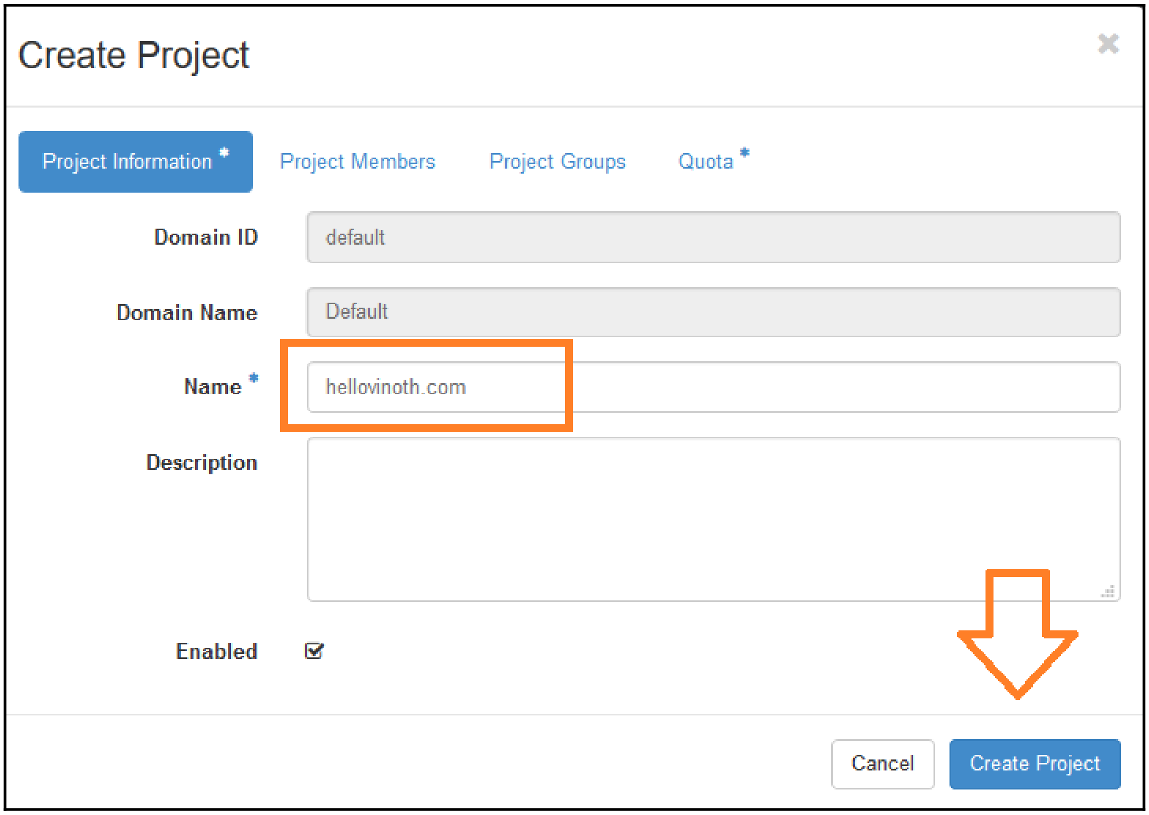


Displaying 5 items

	Name	Description	Project ID	Domain Name	Enabled	Actions
☐	alt_demo		009e1dcac6554c20a7fd5439f0cf8cd6	Default	Yes	Manage Members
☐	demo		a103a240c5b643f085af45390dde02f1	Default	Yes	Manage Members
☐	invisible_to_admin		c98e25ebac3b4e7485792c8e513dce0d	Default	Yes	Manage Members
☐	service		e3f1c391ff8048ac965b4e444f32b932	Default	Yes	Manage Members
☐	admin	Bootstrap project for initializing the cloud.	fe50f731283540769c4481c80612e053	Default	Yes	Manage Members

Displaying 5 items

3. Click the **+Create Project** button in the top right corner:


Create Project

Domain ID: default

Domain Name: Default

Name *: hellovinoth.com

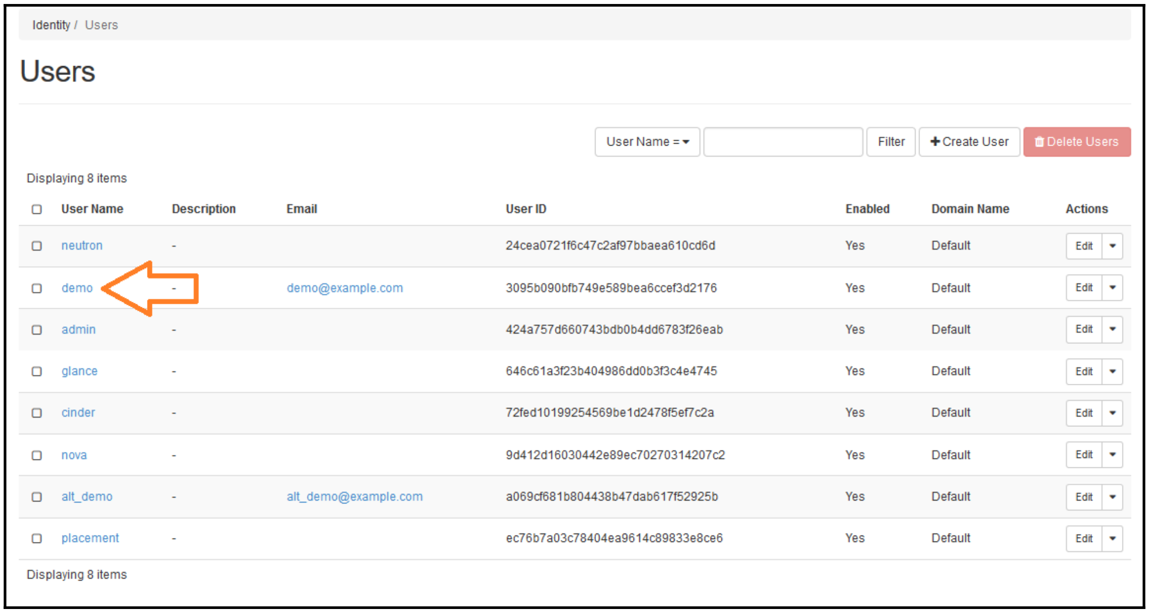
Description:

Enabled: ✓

In response, you will get the **Create Project** pop-up window. In that, specify the name for your new project and leave the rest unchanged. Then click the **Create Project** button.

By now, you should see that your new project is added to the project list. In my case, I could see my new project `hellovinoth.com` in the project list:

4. Click the **Users** menu under the **Identity** tab in Horizon.
5. In response, you should see the list of all of the users in the lab environment, including the **demo** user we used before:



Users

Displaying 8 items

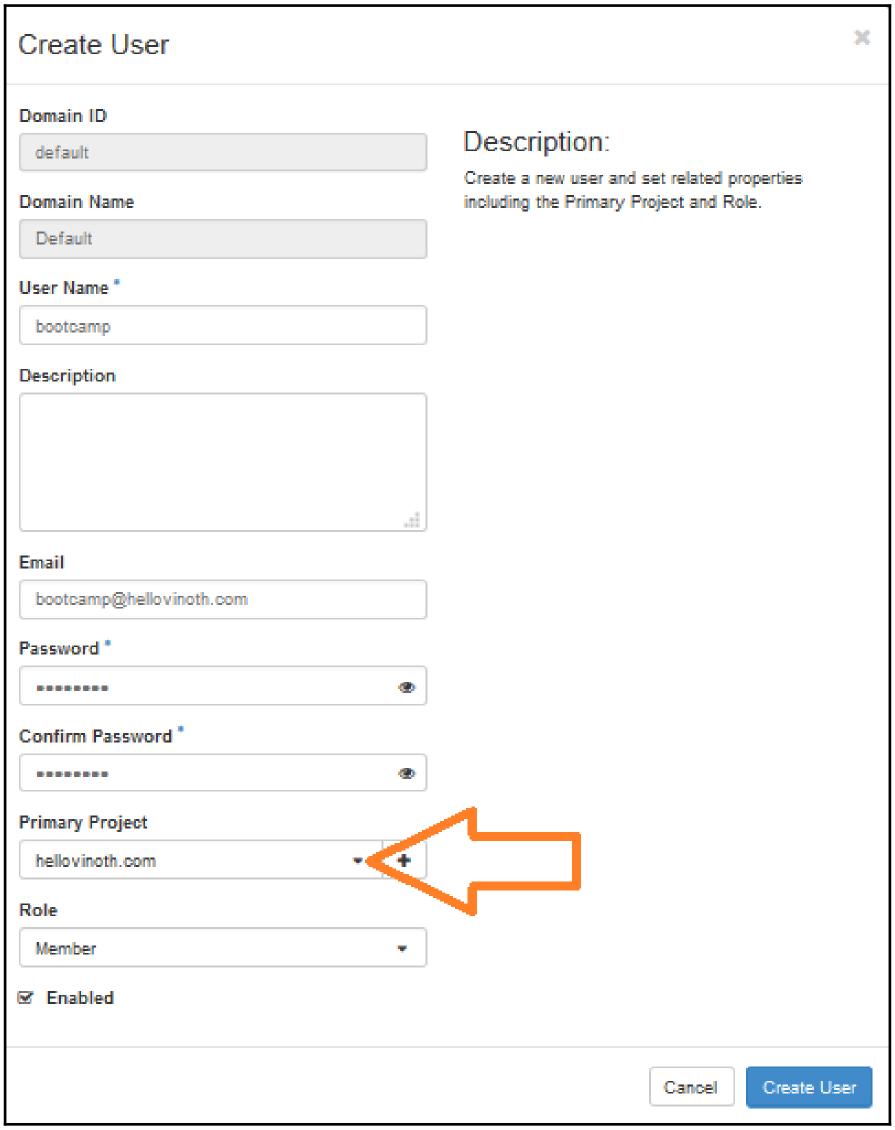
☐	User Name	Description	Email	User ID	Enabled	Domain Name	Actions
☐	neutron	-		24cea0721f6c47c2af97bbaea610cd6d	Yes	Default	Edit ▾
☐	demo	-	demo@example.com	3095b090bfb749e589bea6ccef3d2176	Yes	Default	Edit ▾
☐	admin	-		424a757d660743bdb0b4dd6783f26eab	Yes	Default	Edit ▾
☐	glance	-		646c61a3f23b404986dd0b3f3c4e4745	Yes	Default	Edit ▾
☐	cinder	-		72fed10199254569be1d2478f5ef7c2a	Yes	Default	Edit ▾
☐	nova	-		9d412d16030442e89ec70270314207c2	Yes	Default	Edit ▾
☐	alt_demo	-	alt_demo@example.com	a069cf681b804438b47dab617f52925b	Yes	Default	Edit ▾
☐	placement	-		ec76b7a03c78404ea9614c89833e8ce6	Yes	Default	Edit ▾

Displaying 8 items

6. Click the **Create User** button in the top right corner.
7. In response, you will get the **Create User** pop-up window. In that, specify the following values and click the **Create User** button:

Parameter	**Value**
User Name*	`bootcamp`
Password*	`nomoresecret`
Primary Project	`hellovinoth.com`
Role	`member`

The following screenshot shows the sample **Create User** pop-up window:



Create User

Domain ID
default

Domain Name
Default

User Name *
bootcamp

Description

Email
bootcamp@hellovinoth.com

Password *
••••••••

Confirm Password *
••••••••

Primary Project
hellovinoth.com

Role
Member

☑ Enabled

Description:
Create a new user and set related properties including the Primary Project and Role.

By now, you should see that your new user has been added to the user's list. In my case, I could see my new user `bootcamp` in the user list:

1. Let's test our new project and user. Click the **Sign Out** link in the top right corner of the window to sign out as the **admin** user.
2. Now, sign in to the Horizon dashboard using the `bootcamp` / `nomoresecret` credentials.

Checkpoint

Adding Projects and Users using the Horizon dashboard.

Networking service (neutron)

In this session, we are going to look at the neutron network, router, and routing to the external world:

Goal	Create and configure networking with neutron
Activities	• Create a new private network using Horizon • Launch an instance using the newly created network • Create a new router for the project • Configure external network connectivity

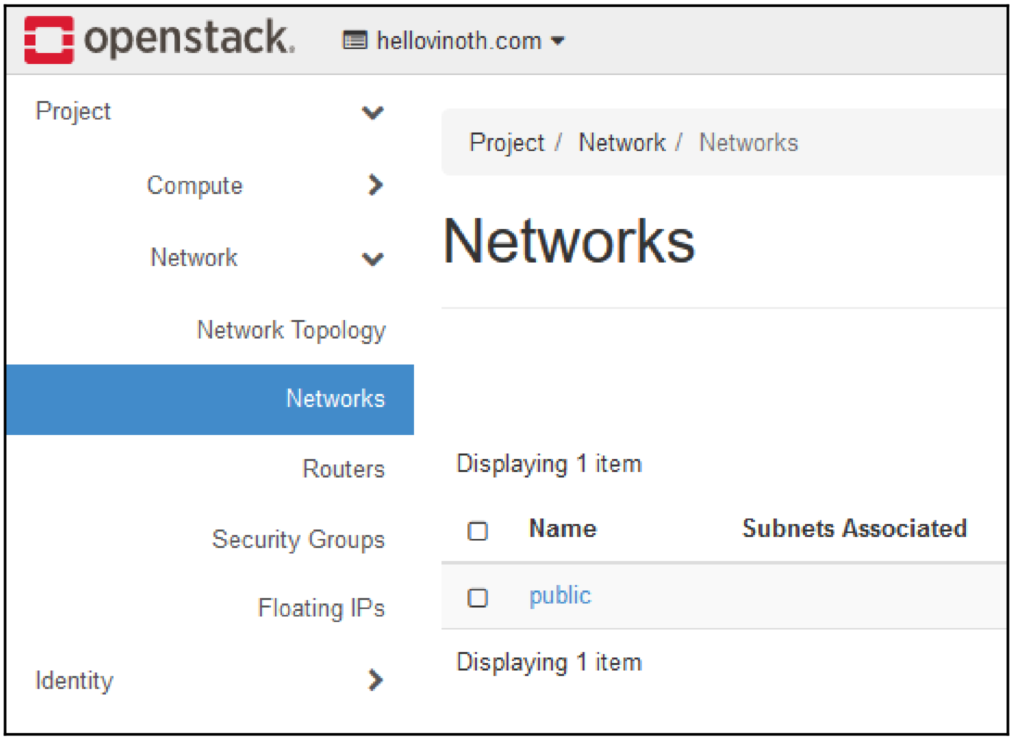
Creating a network using Horizon

In this section, we will create the new private/tenant network for the project we have created in the previous *Adding projects and users* section using the Horizon dashboard:

1. Sign-in to the Horizon dashboard using the `bootcamp` / `nomoresecret` credentials to log in to the project `hellovinoth.com`.

You may use the user credentials of the user you have created in the previous session to log in to the project.

2. Click the **Networks** menu in the left navigation panel on the **Project** tab in Horizon to see the available **Networks** list. By now, you should see only the public network listed:



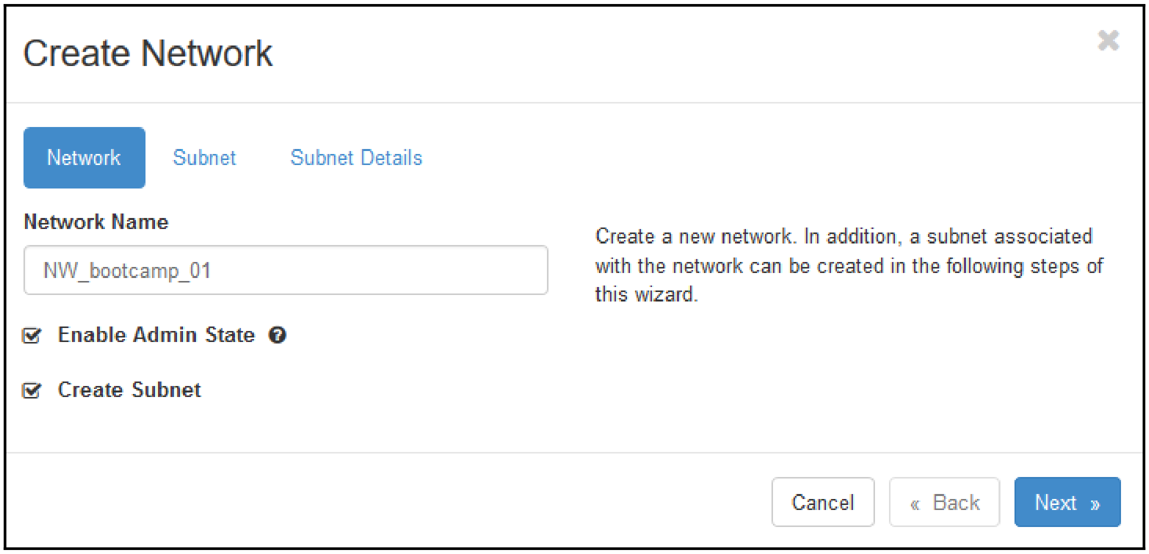
Networks

Displaying 1 item

Name	Subnets Associated
public	

Displaying 1 item

3. Click the **Create Network** button in the top right corner:


Create Network

Network Name

NW_bootcamp_01

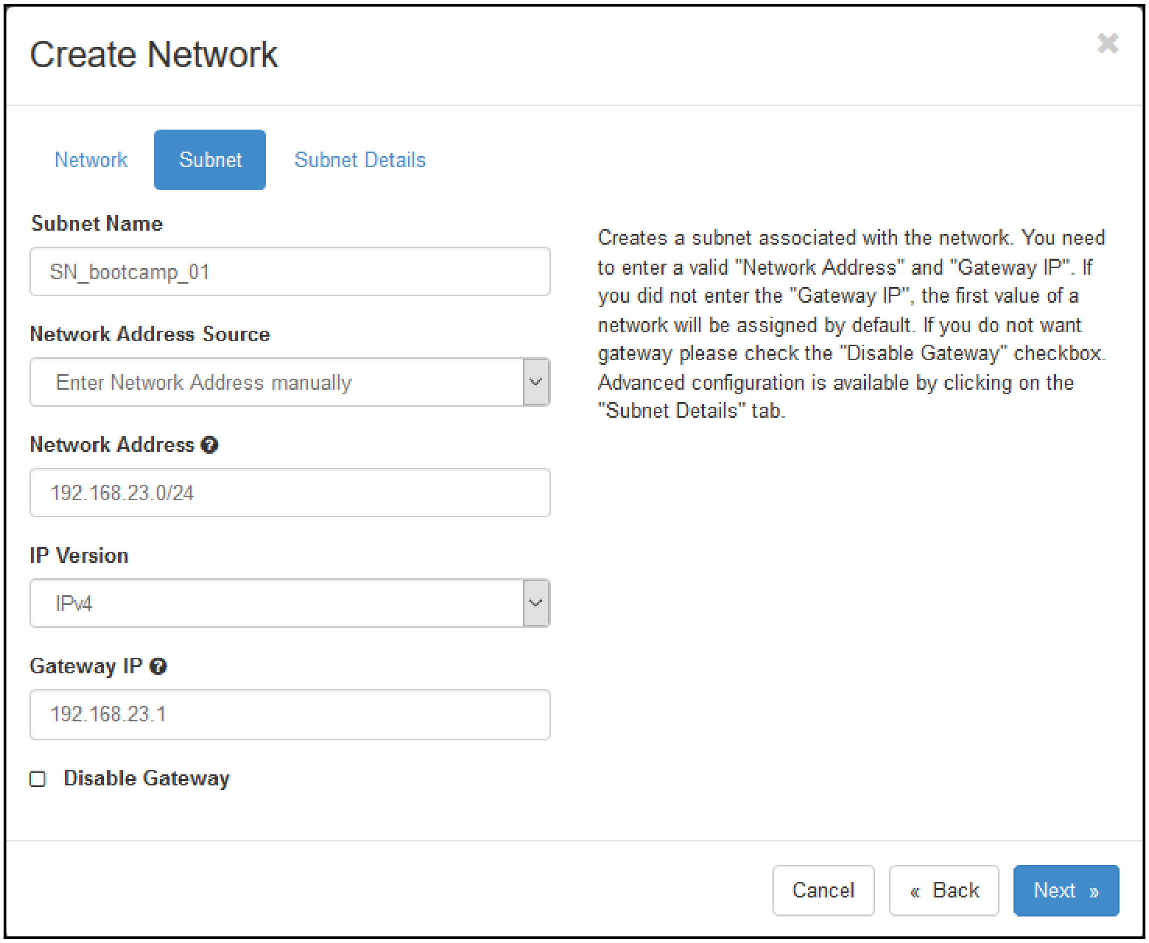
Create a new network. In addition, a subnet associated with the network can be created in the following steps of this wizard.

☑ Enable Admin State

☑ Create Subnet

In response, the **Create Network** screen will get displayed with three tabs: **Network/Subnet/Subnet Details**:

- **Network**: This tab specifies the name of the new network and whether it should be automatically **up** or leave it in a **down** state after creation (**Admin** state unchecked means that the network will be in **down** state and does not forward packets):


Create Network

Subnet Name
SN_bootcamp_01

Network Address Source
Enter Network Address manually

Network Address
192.168.23.0/24

IP Version
IPv4

Gateway IP
192.168.23.1

☐ Disable Gateway

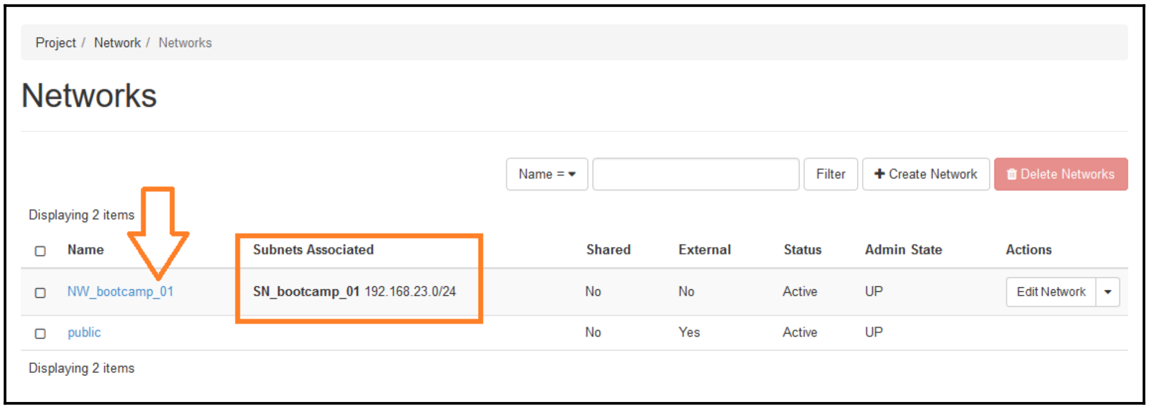
Creates a subnet associated with the network. You need to enter a valid "Network Address" and "Gateway IP". If you did not enter the "Gateway IP", the first value of a network will be assigned by default. If you do not want gateway please check the "Disable Gateway" checkbox. Advanced configuration is available by clicking on the "Subnet Details" tab.

- **Subnet**: Allows creating a new subnet and mapping it the network. It allows you to specify the subnet name, IP address range to use in CIDR format, IP version, and the optional Gateway IP.
- **Subnet Detail**: Allows enabling of the DHCP server for the network and configure some specific DHCP parameters such as allocation pool and DNS name servers.

4. In the **Create Network** window, specify the following values and click the **Create** button:

Network	
Network Name	`NW_bootcamp_01`
Subnet	
Subnet Name	`SN_bootcamp_01`
Network Address Source	**Enter Network Address Manually**
Network Address	`192.168.23.0/24`
IP Version	**IPv4**
Gateway IP	`192.168.23.1`
Subnet Details	
DNS Name Server	`8.8.8.8`
Leave the rest default or empty	

By now, you should see that the `NW_bootcamp_01` private network appears in the networks list:


Project / Network / Networks

Networks

Displaying 2 items

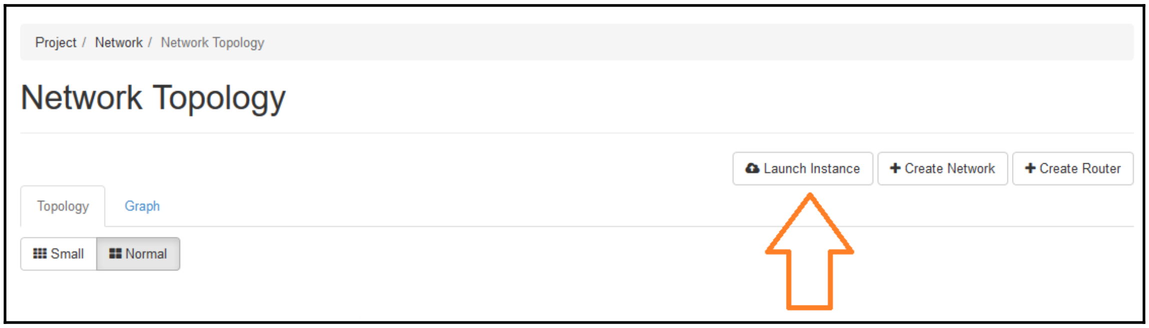
Name	Subnets Associated	Shared	External	Status	Admin State	Actions
NW_bootcamp_01	**SN_bootcamp_01** 192.168.23.0/24	No	No	Active	UP	Edit Network
public		No	Yes	Active	UP	

Displaying 2 items

Verifying the network connectivity

Let's test our new network by launching two new VMs in it and verifying if those VMs have network connectivity between each other. To verify the network connectivity, follow these instructions:

1. Click the **Network Topology** menu in the left navigation panel on the **Project** tab in Horizon.
2. On the **Network Topology** screen click the **Launch Instance** button on the right top edge of the screen:



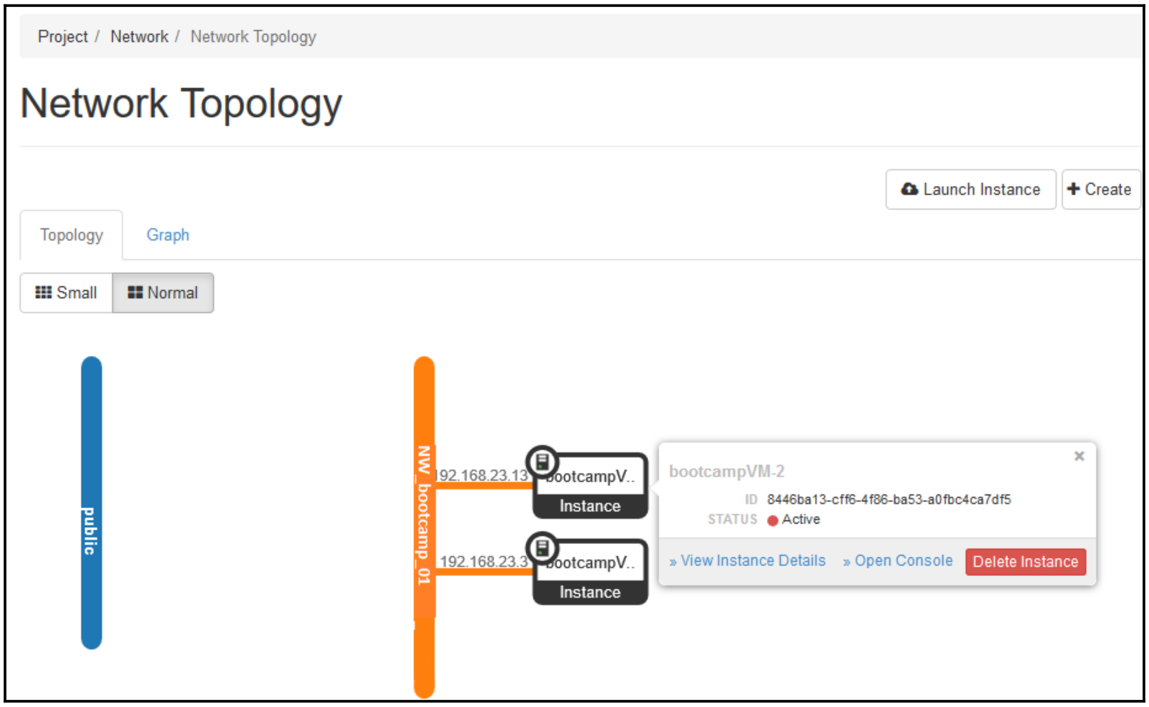
3. In response, you will get a **Launch Instance** pop-up window. In the **Launch Instance** window, specify the following values and click the **Launch** button. You could refer to the *Launching a new instance using Horizon* section to launch a virtual machine using the Horizon dashboard:

Details	
Instance Name*	`bootcampVM`
Availability Zone	`nova`
Count*	`2`
Source	
Select Boot Source	`Image`
Create New Volume	`No`
Image Name	`cirros-0.3.4-x86_64-uec`
Flavor	
Allocated Flavor name	`m1.nano`

Networks	
Allocated Network Name	`NW_bootcamp_01`
Security Groups	
Allocated Security Group Name	`Default`

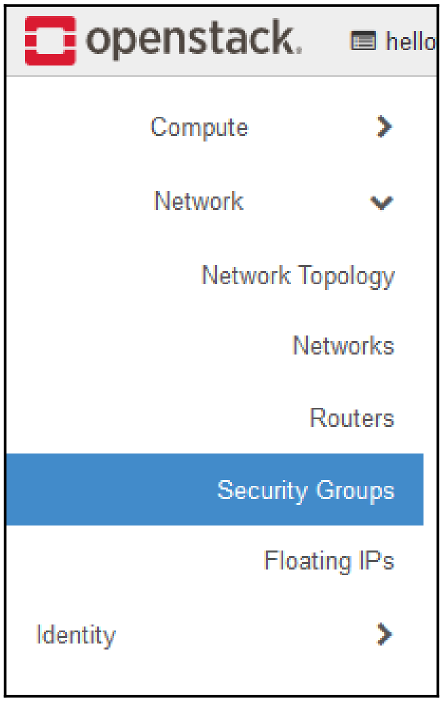
Set the instance count as two to launch two new instances at the same time with the same configuration.

After submitting the launch instance request by clicking the **Launch Instance** button, on the **Network Topology** screen, you will see how VMs are started and connected to the `NW_bootcamp_01` network:

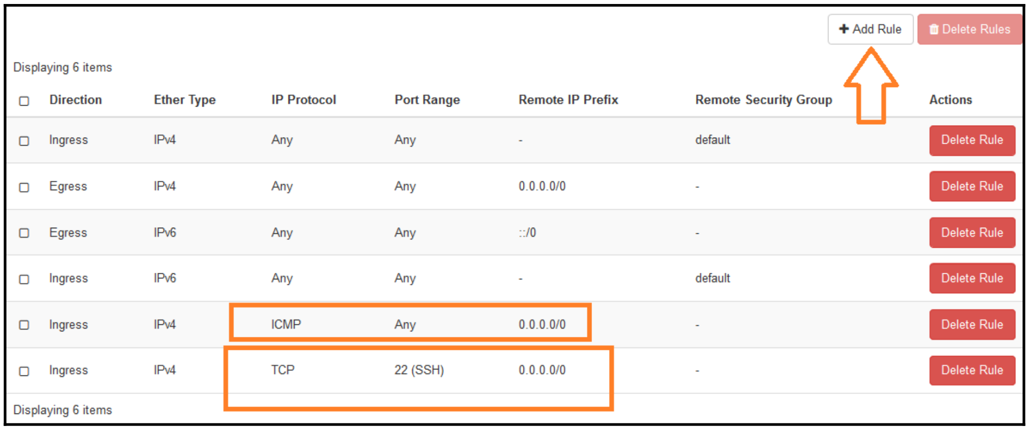


TIP Note the private IP addresses for the VMs in the `NW_bootcamp_01` network. In my case, from the preceding screenshot, I could identify the IP address of my new instances as `192.168.23.3` and `192.168.23.13`. For instance, `bootcampVM-1` and `bootcampVM-2`, respectively.

1. Before we log in to the instance and verify the network connectivity, we need to add the security group rule to the default **Security Group** to allow ICMP and SSH traffic. To add a security group rule via Horizon, click the **Security Groups** menu in the left navigation panel on the **Network** tab in Horizon:



2. By now, you should see the available security groups for the selected project. Click on the **Manage Rules** button on the right side of the default **Security Groups** list.
3. In response, you will get a **Manage Security Group Rules** window with the list of all of the security rules for the selected security group:

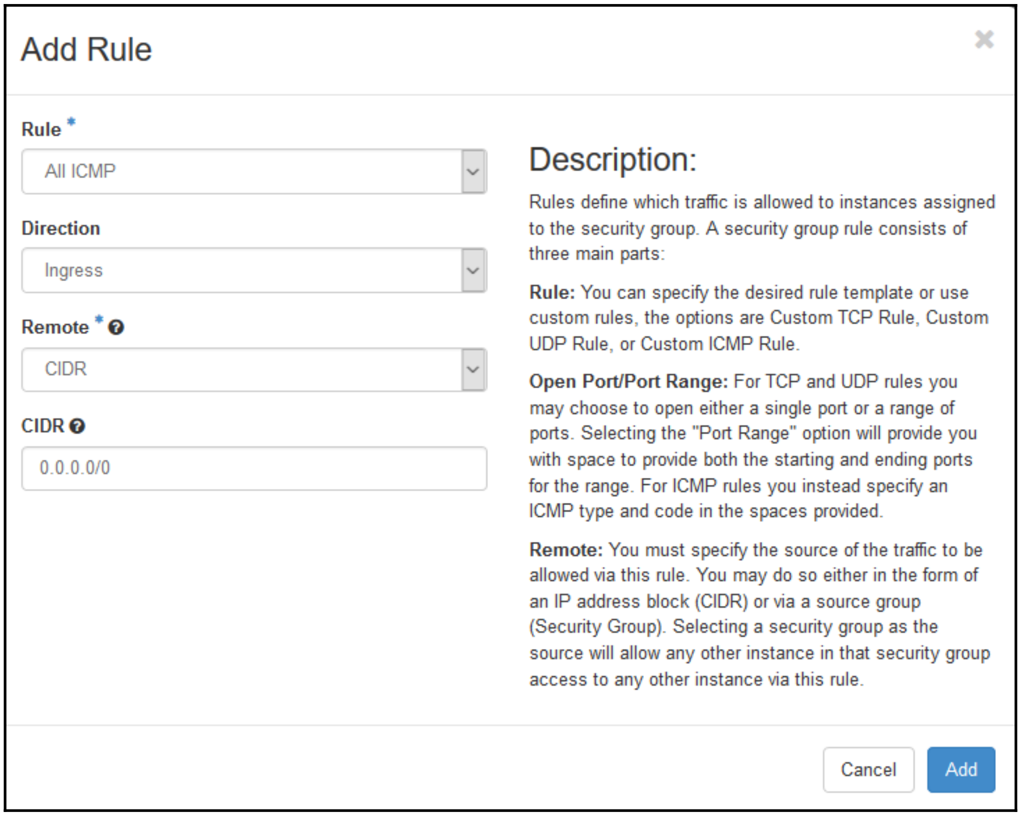


Displaying 6 items

	Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Actions
☐	Ingress	IPv4	Any	Any	-	default	Delete Rule
☐	Egress	IPv4	Any	Any	0.0.0.0/0	-	Delete Rule
☐	Egress	IPv6	Any	Any	::/0	-	Delete Rule
☐	Ingress	IPv6	Any	Any	-	default	Delete Rule
☐	Ingress	IPv4	ICMP	Any	0.0.0.0/0	-	Delete Rule
☐	Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	Delete Rule

Displaying 6 items

4. Click the **+ Add Rule** button on the top right corner of the window to add a new rule to the selected security group:



Add Rule

Rule *: All ICMP

Direction: Ingress

Remote *: CIDR

CIDR: 0.0.0.0/0

Description:

Rules define which traffic is allowed to instances assigned to the security group. A security group rule consists of three main parts:

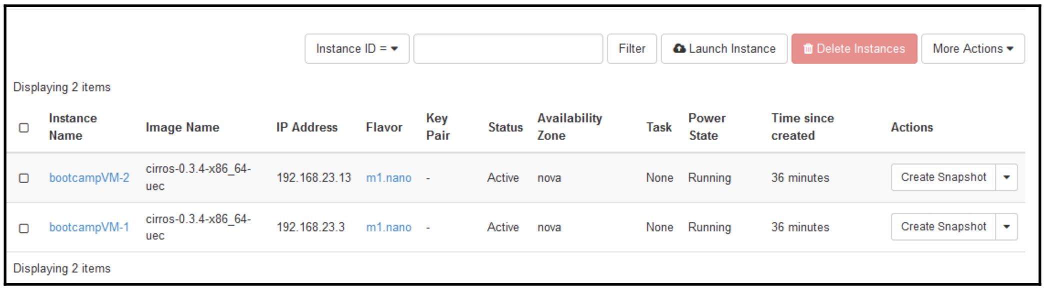
Rule: You can specify the desired rule template or use custom rules, the options are Custom TCP Rule, Custom UDP Rule, or Custom ICMP Rule.

Open Port/Port Range: For TCP and UDP rules you may choose to open either a single port or a range of ports. Selecting the "Port Range" option will provide you with space to provide both the starting and ending ports for the range. For ICMP rules you instead specify an ICMP type and code in the spaces provided.

Remote: You must specify the source of the traffic to be allowed via this rule. You may do so either in the form of an IP address block (CIDR) or via a source group (Security Group). Selecting a security group as the source will allow any other instance in that security group access to any other instance via this rule.

I have added ALL ICMP and SSH rules to the **Default Security Group** to allow ping/SSH connectivity between the instance.

5. Now, jump to the **Instance** menu in the left navigation panel on the Horizon dashboard, which will display the available instance list:



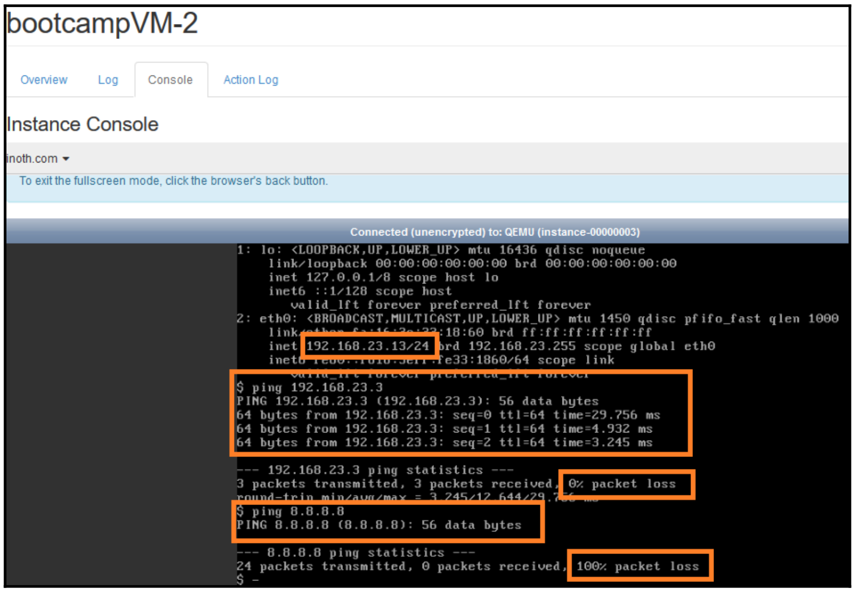
Instance ID = ▾ | Filter | Launch Instance | Delete Instances | More Actions ▾

Displaying 2 items

☐	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
☐	bootcampVM-2	cirros-0.3.4-x86_64-uec	192.168.23.13	m1.nano	-	Active	nova	None	Running	36 minutes	Create Snapshot ▾
☐	bootcampVM-1	cirros-0.3.4-x86_64-uec	192.168.23.3	m1.nano	-	Active	nova	None	Running	36 minutes	Create Snapshot ▾

Displaying 2 items

6. Connect to any instance using the VNC Console and log in to the operating system. You may refer to the *Connecting to the instance using VNC console* section in case of any difficulties in connecting the instance via the VNC console.
7. Once you are logged in to the virtual machine, try to ping the other virtual machine IP address:



bootcampVM-2

Overview | Log | Console | Action Log

Instance Console

noth.com ▾

To exit the fullscreen mode, click the browser's back button.

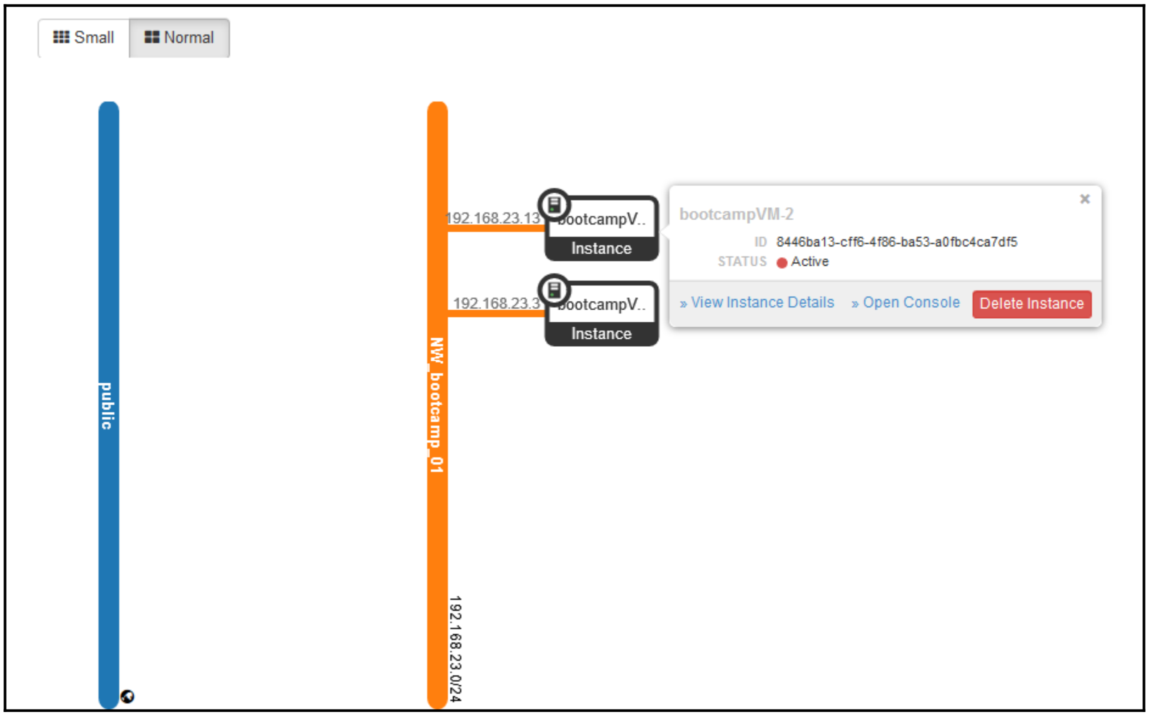
Connected (unencrypted) to: QEMU (instance-00000003)

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc pfifo_fast qlen 1000
    link/ether fa:16:3e:33:18:60 brd ff:ff:ff:ff:ff:ff
    inet 192.168.23.13/24 brd 192.168.23.255 scope global eth0
    inet6 fe80::f816:3eff:fe33:1860/64 scope link
       valid_lft forever preferred_lft forever
$ ping 192.168.23.3
PING 192.168.23.3 (192.168.23.3): 56 data bytes
64 bytes from 192.168.23.3: seq=0 ttl=64 time=29.756 ms
64 bytes from 192.168.23.3: seq=1 ttl=64 time=4.932 ms
64 bytes from 192.168.23.3: seq=2 ttl=64 time=3.245 ms

--- 192.168.23.3 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 3.245/12.644/29.756 ms
$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes

--- 8.8.8.8 ping statistics ---
24 packets transmitted, 0 packets received, 100% packet loss
$ _
```

If everything is configured correctly, you should be able to ping another instance successfully and unable to ping `8.8.8.8` (Google Public DNS).

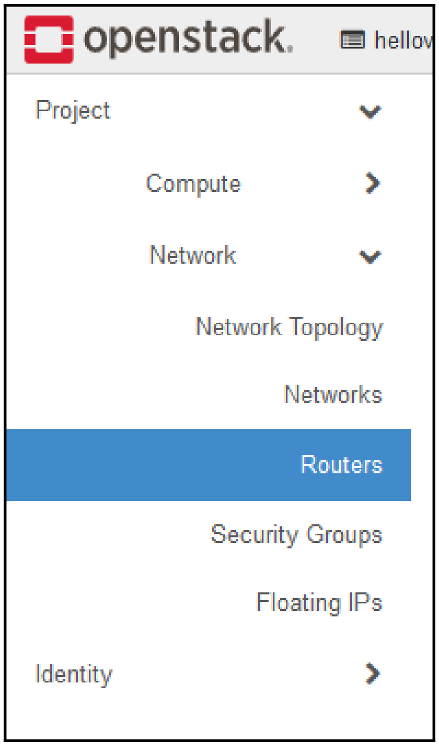
Configuring routing for external networks

While verifying the network connectivity, we were unable to ping the external IP address (`8.8.8.8`). That was because the VM is not connected to any router that will route the packets from the VM to an external network:

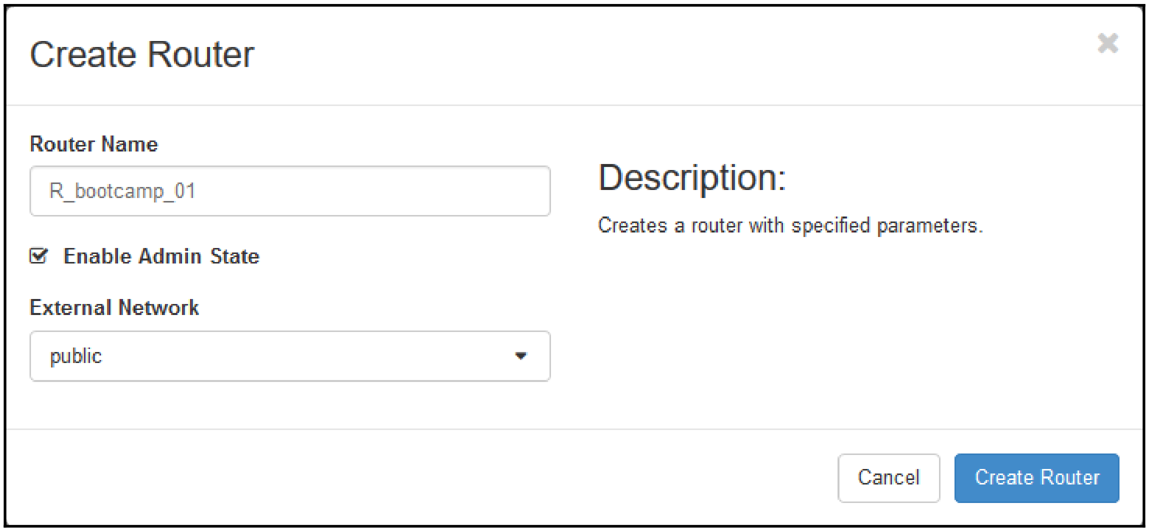


To configure the router for the external connectivity, follow these steps:

1. Click the **Routers** menu in the left navigation panel on the **Network** tab in Horizon:



2. By now, you can see the list of available routers for the selected project. Click the **Create Router** button in the top right corner of the **Routers** window.
3. In response, you will get a **Create Router** pop-up window. In that specify the router name, and then select the external network from the dropdown. By default, the DevStack setup has the **public** network configured for external connectivity:



Create Router

Router Name: R_bootcamp_01

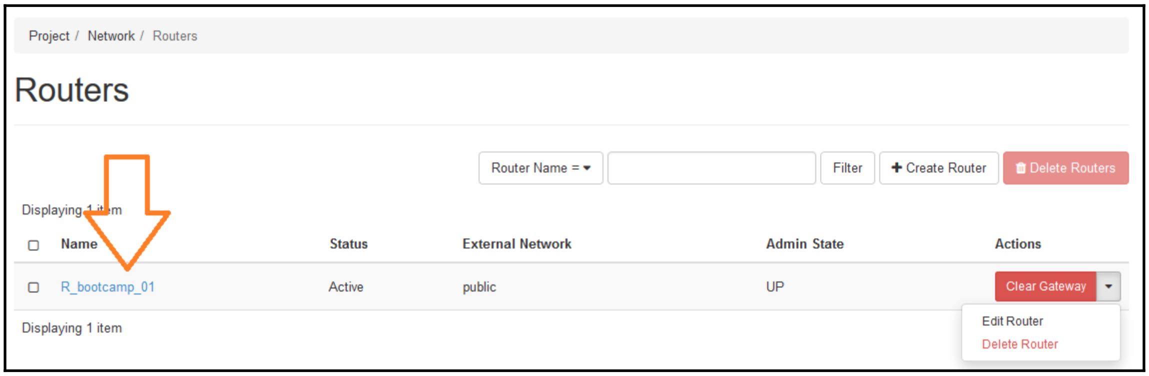
☑ Enable Admin State

External Network: public

Description:

Creates a router with specified parameters.

4. After filling the **Create Router** pop-up window, click the **Create Router** button. In response, a new router will be created and listed in the routers window:



Project / Network / Routers

Routers

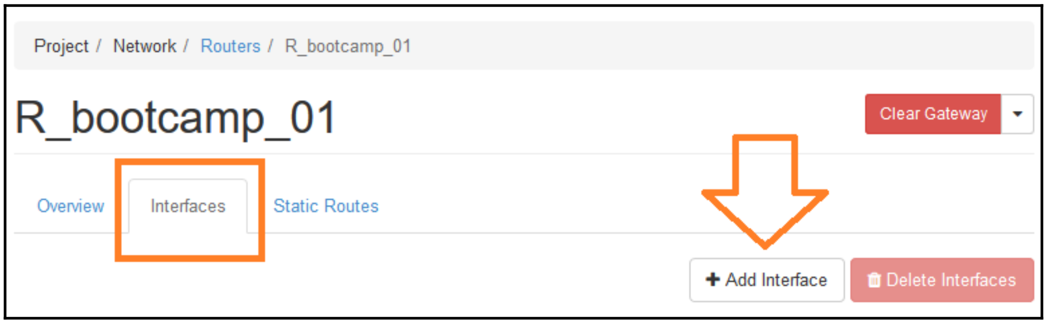
Displaying 1 item

Name	Status	External Network	Admin State	Actions
R_bootcamp_01	Active	public	UP	Clear Gateway

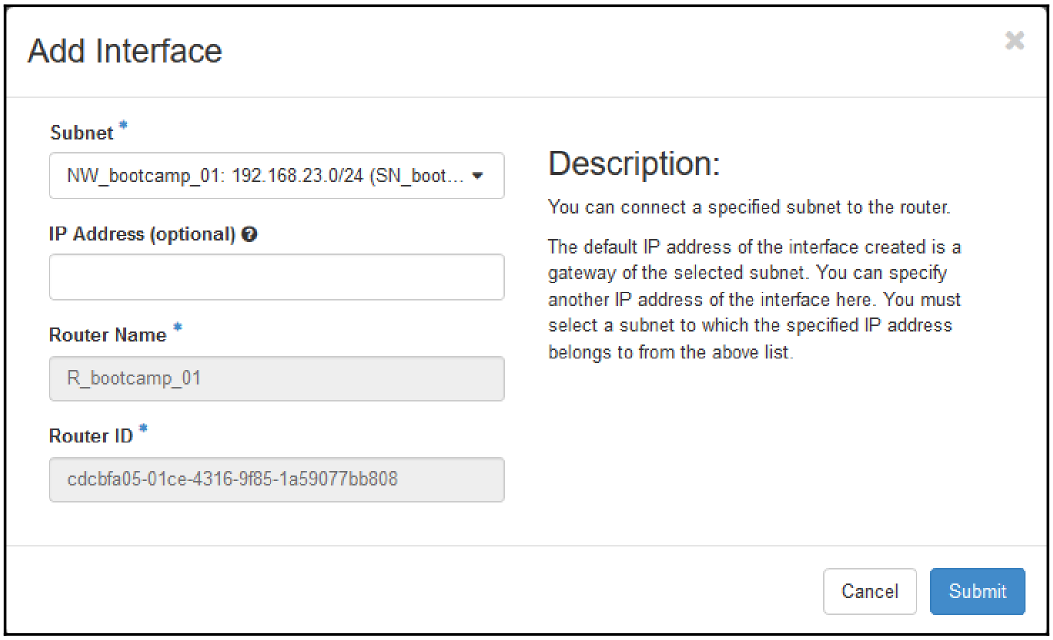
Displaying 1 item

5. From the router list, click on the name of the new router `R_bootcamp_01`, which will open the router overview window with three tabs, namely **Overview**, **Interfaces**, and **Static Routes**.

6. Select the **Interfaces** tab on the router overview window, which will display the available interfaces for the selected router. By now, the new router should have no interface added yet:



7. Click the **+Add Interface** button on the right top corner of the router interfaces window.
8. In response, you will get **+Add Interfaces** pop-up window displayed. In that, select the subnet from the drop-down option and click the **Submit** button:



Subnet *: NW_bootcamp_01: 192.168.23.0/24 (SN_boot...

IP Address (optional):

Router Name *: R_bootcamp_01

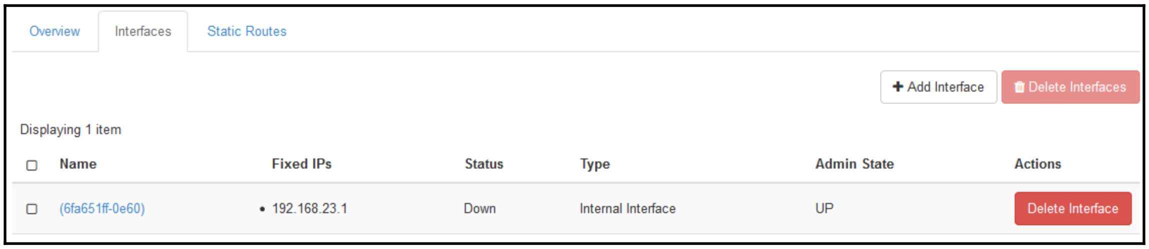
Router ID *: cdcbfa05-01ce-4316-9f85-1a59077bb808

Description:

You can connect a specified subnet to the router.

The default IP address of the interface created is a gateway of the selected subnet. You can specify another IP address of the interface here. You must select a subnet to which the specified IP address belongs to from the above list.

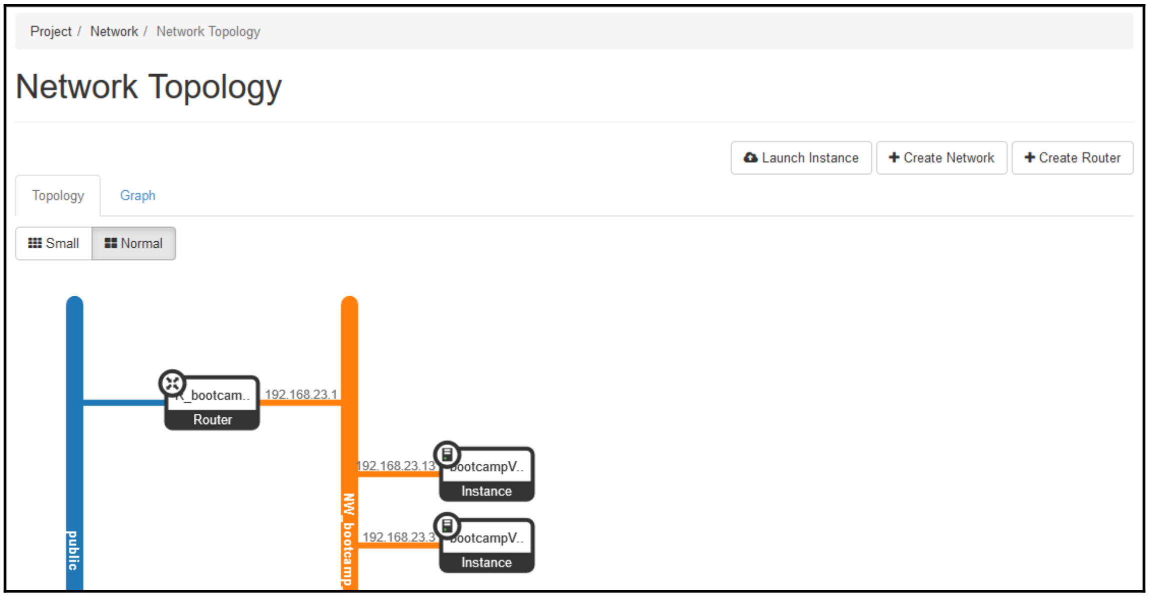
By now, you should see the newly added interface listed in the router interfaces window:



Displaying 1 item

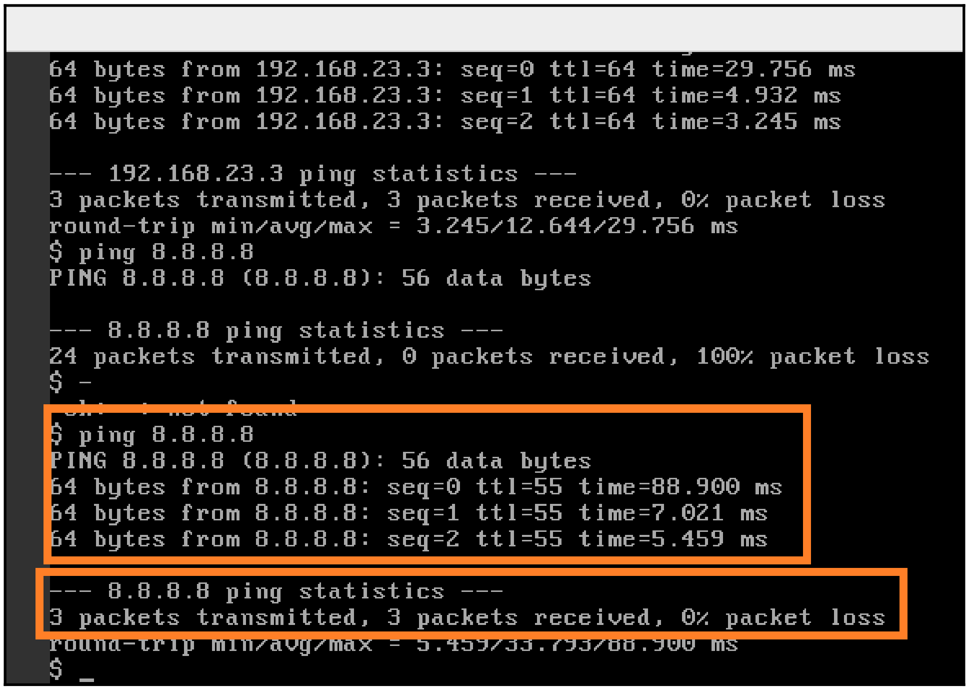
	Name	Fixed IPs	Status	Type	Admin State	Actions
☐	(6fa651ff-0e60)	• 192.168.23.1	Down	Internal Interface	UP	Delete Interface

9. Jump to the **Network Topology** menu in the left navigation panel on the **Network** tab in Horizon. Now, you can see the network topology diagram with the router `R_bootcamp_01` connecting the external network **public** and the private network `NW_bootcamp_01`:



10. Switch the window focuses back to the VNC console window. Please refer to *step 9*.

Now, try to ping the external IP address from the virtual machine:



```
64 bytes from 192.168.23.3: seq=0 ttl=64 time=29.756 ms
64 bytes from 192.168.23.3: seq=1 ttl=64 time=4.932 ms
64 bytes from 192.168.23.3: seq=2 ttl=64 time=3.245 ms

--- 192.168.23.3 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 3.245/12.644/29.756 ms
$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes

--- 8.8.8.8 ping statistics ---
24 packets transmitted, 0 packets received, 100% packet loss
$ -
sh: -: not found
$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=55 time=88.900 ms
64 bytes from 8.8.8.8: seq=1 ttl=55 time=7.021 ms
64 bytes from 8.8.8.8: seq=2 ttl=55 time=5.459 ms

--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 5.459/33.793/88.900 ms
$ _
```

If you can ping the external IP address like `8.8.8.8` successfully, congratulate yourself. You have configured a neutron networking with external network connectivity for your new project.

Checkpoint

- Create a new private network using Horizon
- Launch an instance using the newly created network
- Verify the network connectivity between the newly created instance
- Create a new router for the project
- Configure external network connectivity

Add-on exercises

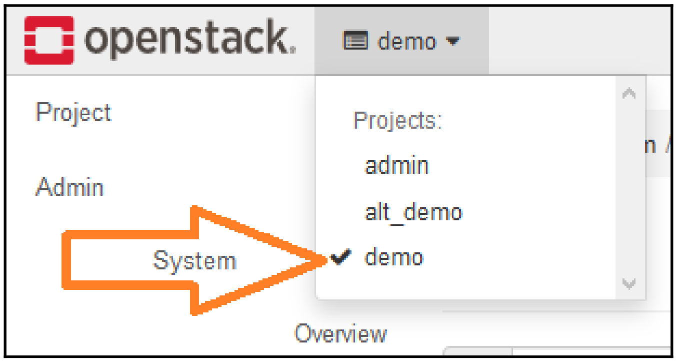
So far, we have seen lab exercises based on OpenStack component categories. In this session, we will be walking through some cherry-picked activities via horizon.

Making existing Glance images public

Only the admin privileged user could make any images public in the OpenStack. The public Glance image will get shared between all of the projects in the OpenStack cluster.

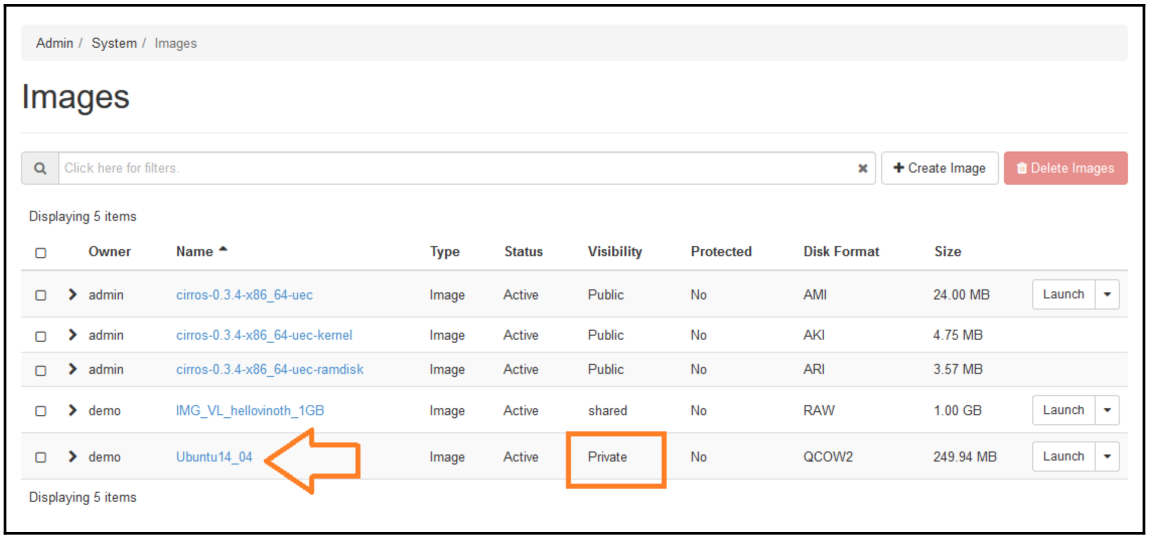
To make the existing image as public, do follow the instruction given here:

1. Log in to the Horizon dashboard with the **admin** user credentials. You could also refer to the *With Horizon dashboard* section for your reference.
2. Change the project to **demo**, as we have already uploaded the `ubuntu14` cloud image to the project **demo** during the previous exercises. We can make that private Ubuntu image public:



3. Click the **Images** menu in the left navigation panel under the **Admin** tab on the Horizon dashboard.

4. By now, you should see the list of all of the available images in the OpenStack cluster, irrespective of the project the images belong to:



Admin / System / Images

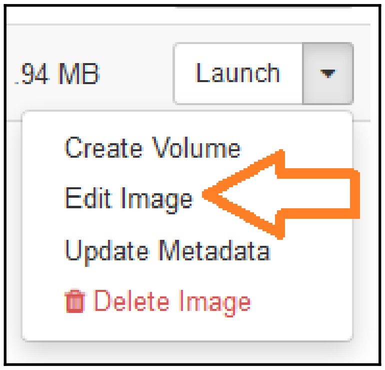
Images

Displaying 5 items

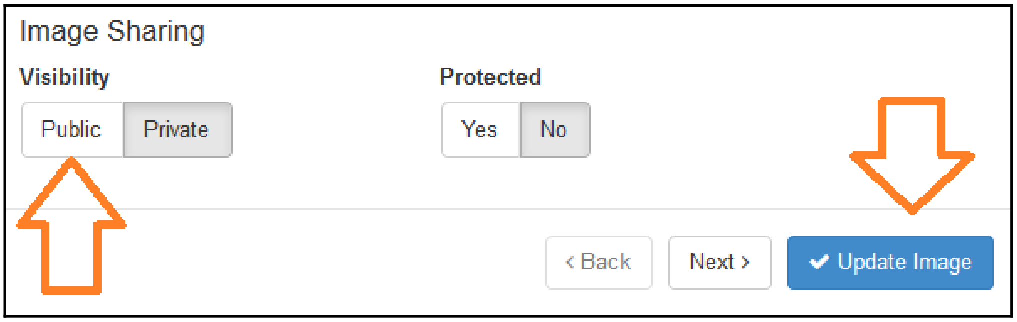
	Owner	Name	Type	Status	Visibility	Protected	Disk Format	Size	
	admin	cirros-0.3.4-x86_64-uec	Image	Active	Public	No	AMI	24.00 MB	Launch
	admin	cirros-0.3.4-x86_64-uec-kernel	Image	Active	Public	No	AKI	4.75 MB	
	admin	cirros-0.3.4-x86_64-uec-ramdisk	Image	Active	Public	No	ARI	3.57 MB	
	demo	IMG_VL_hellovinoth_1GB	Image	Active	shared	No	RAW	1.00 GB	Launch
	demo	Ubuntu14_04	Image	Active	Private	No	QCOW2	249.94 MB	Launch

Displaying 5 items

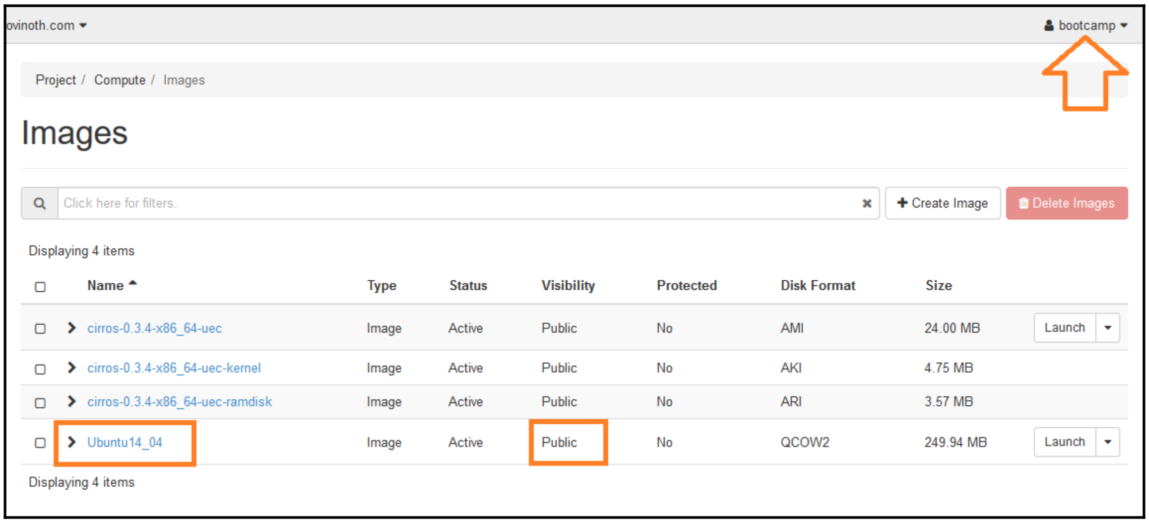
5. Click the drop-down button next to the **Launch** button on the `Ubuntu14_04` private image list:



6. Click the **Edit Image** button. In response, you will get an **Edit Image** pop-up window:



7. Click the **Public** button under the image sharing option at the bottom left of the **Edit Image** pop-up window and then click the **Update Image** button to make the image as **Public**.
8. Then, log out from the **admin** account and log in as any other non-privileged user to check the image list:



Displaying 4 items

	Name	Type	Status	Visibility	Protected	Disk Format	Size	
☐	cirros-0.3.4-x86_64-uec	Image	Active	Public	No	AMI	24.00 MB	Launch
☐	cirros-0.3.4-x86_64-uec-kernel	Image	Active	Public	No	AKI	4.75 MB	
☐	cirros-0.3.4-x86_64-uec-ramdisk	Image	Active	Public	No	ARI	3.57 MB	
☐	Ubuntu14_04	Image	Active	Public	No	QCOW2	249.94 MB	Launch

Displaying 4 items

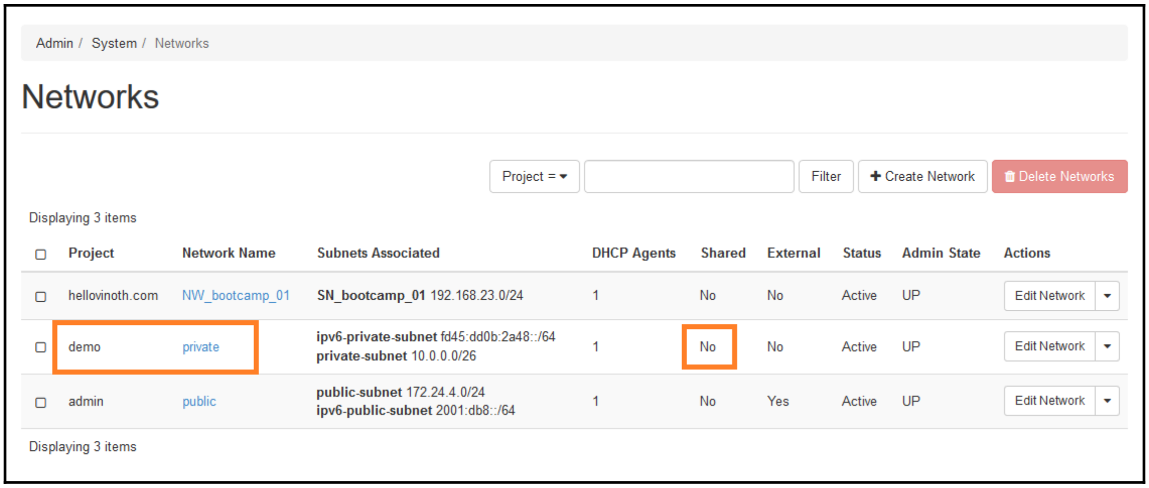
Sharing networks between projects

To share the networks between the projects, follow these instructions:

1. Log in to the Horizon dashboard with the **admin** user credentials.

You can also refer to the *With Horizon dashboard* section for your reference.

2. Click the **Networks** menu in the left navigation panel under the **Admin** tab on the Horizon dashboard.
3. By now, you can see the list of networks across all the projects. Let's share the **private** network of the **demo** project with other projects:



Admin / System / Networks

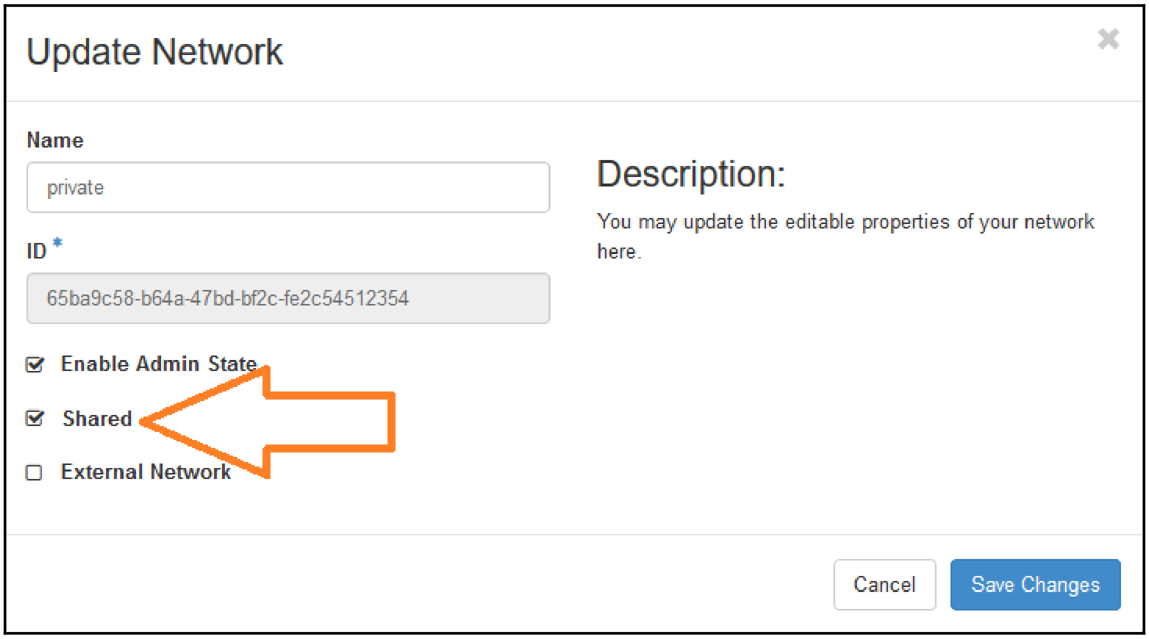
Networks

Displaying 3 items

Project	Network Name	Subnets Associated	DHCP Agents	Shared	External	Status	Admin State	Actions
hellovinoth.com	NW_bootcamp_01	**SN_bootcamp_01** 192.168.23.0/24	1	No	No	Active	UP	Edit Network
demo	private	**ipv6-private-subnet** fd45:dd0b:2a48::/64 **private-subnet** 10.0.0.0/26	1	No	No	Active	UP	Edit Network
admin	public	**public-subnet** 172.24.4.0/24 **ipv6-public-subnet** 2001:db8::/64	1	No	Yes	Active	UP	Edit Network

Displaying 3 items

4. Click the **Edit Network** button next to the private network. In response, you will get an **Update Network** pop-up window:



Update Network

Name

private

ID *

65ba9c58-b64a-47bd-bf2c-fe2c54512354

✓ Enable Admin State

✓ Shared

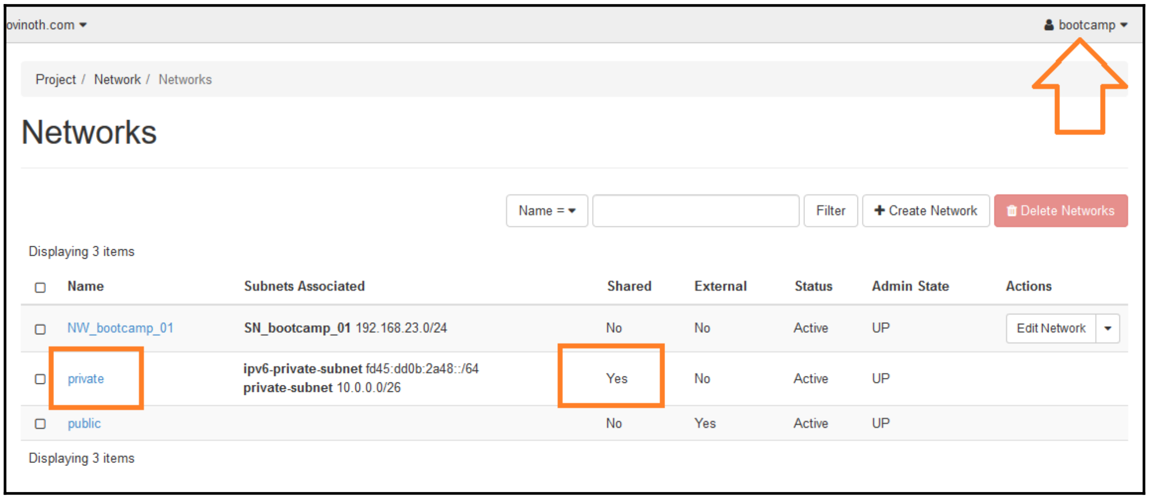
☐ External Network

Description:

You may update the editable properties of your network here.

5. In the **Update Network** window, check the **Shared** checkbox and click the **Save Changes** button.
6. By now, in the networks list, you should see the value of the **Shared** column get changed to **Yes** for the private network.

7. Sign out as the **admin** user and sign back in using the **bootcamp** user credentials to verify the network share:



Networks

Displaying 3 items

Name	Subnets Associated	Shared	External	Status	Admin State	Actions
NW_bootcamp_01	**SN_bootcamp_01** 192.168.23.0/24	No	No	Active	UP	Edit Network
private	**ipv6-private-subnet** fd45:dd0b:2a48::/64 **private-subnet** 10.0.0.0/26	Yes	No	Active	UP	
public		No	Yes	Active	UP	

Displaying 3 items

Creating new flavors

In OpenStack, a flavor describes the compute, memory, and storage capacity of an instance. Only admin privileged users can create, edit, and delete the flavor.

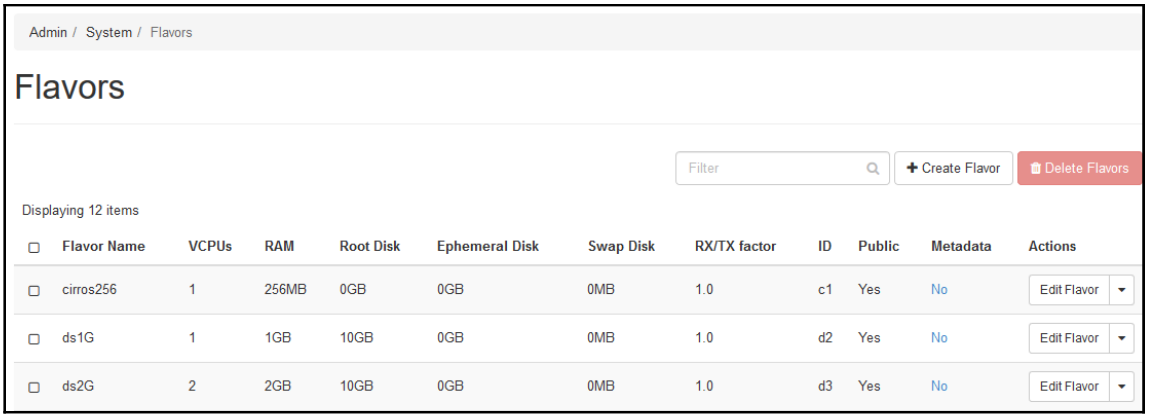
1. Log in to the Horizon dashboard with the **admin** user credentials.

TIP

You could also refer to the *With Horizon dashboard* section for your reference.

2. Click the **Flavors** menu in the left navigation panel under the **Admin** tab on the Horizon dashboard.

3. By now, you should see the list of all of the available flavors in the OpenStack cluster:



Admin / System / Flavors

Flavors

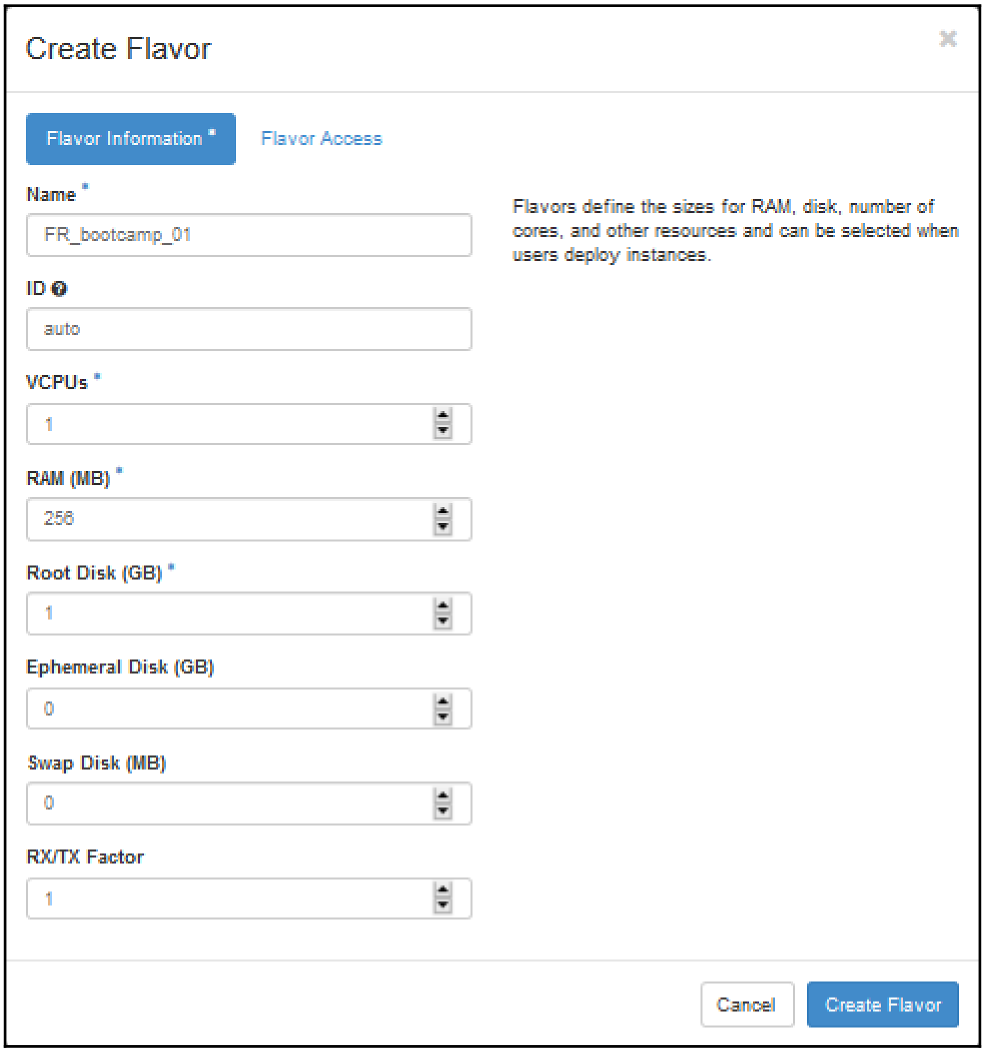
Displaying 12 items

Flavor Name	VCPUs	RAM	Root Disk	Ephemeral Disk	Swap Disk	RX/TX factor	ID	Public	Metadata	Actions
cirros256	1	256MB	0GB	0GB	0MB	1.0	c1	Yes	No	Edit Flavor
ds1G	1	1GB	10GB	0GB	0MB	1.0	d2	Yes	No	Edit Flavor
ds2G	2	2GB	10GB	0GB	0MB	1.0	d3	Yes	No	Edit Flavor

4. Click the **Create Flavor** button in the right top corner of the **Flavors** window. In response, you will get a **Create Flavor** pop-up window. In the **Create Flavor** window, specify the following values and click the **Create Flavor** button:

Flavor Information*	
Name*	`FR_bootcamp_01`
ID	`auto`
vCPU*	**1**
RAM (MB)*	**256**
Root Disk (GB)*	**1**
Ephemeral Disk (GB)	**0**
Swap Disk (MB)	**0**
RX / TX Factor	**1**

The following screenshot shows the sample **Create Flavor** pop-up screen for your reference:


Create Flavor

Name * FR_bootcamp_01

ID auto

VCPUs * 1

RAM (MB) * 256

Root Disk (GB) * 1

Ephemeral Disk (GB) 0

Swap Disk (MB) 0

RX/TX Factor 1

Flavors define the sizes for RAM, disk, number of cores, and other resources and can be selected when users deploy instances.

By now, in the flavors list, you should see the newly created flavor `FR_bootcamp_01` and its value.

5. Now, launch a new instance using the newly created flavor `FR_bootcamp_01`. You can refer to the *Launching a new instance using Horizon* section to launch a virtual machine using Horizon dashboard.

Transferring Cinder volume between projects

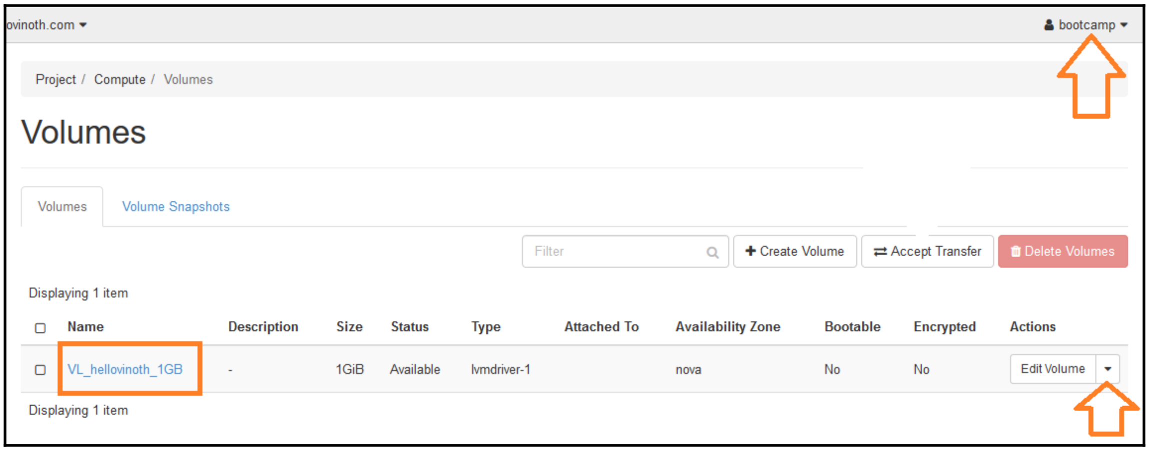
Ownership of a volume can be transferred from one project to another. Once a volume transfer is created in a donor project, it then can be accepted by a recipient project.

To transfer Cinder volume from one project to another, follow these instructions:

1. Log in to the Horizon dashboard with the **bootcamp** user credentials.
2. Click the **Volume** menu in the left navigation panel under the **Project** tab on the Horizon dashboard.

By now, you should see the list of all available volumes in the selected project. In my case, I can see the list of volumes in the `hellovinoth.com` project.

In case you have no volumes listed in the selected project, then you can create the one to initiate the volume transfer. You may refer to the *Managing Cinder volume using Horizon* section to create a new volume:

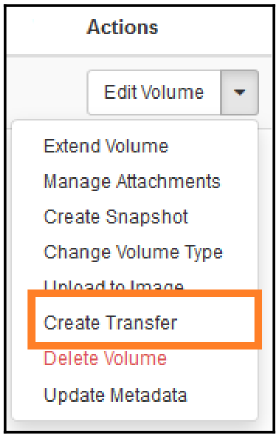


Displaying 1 item

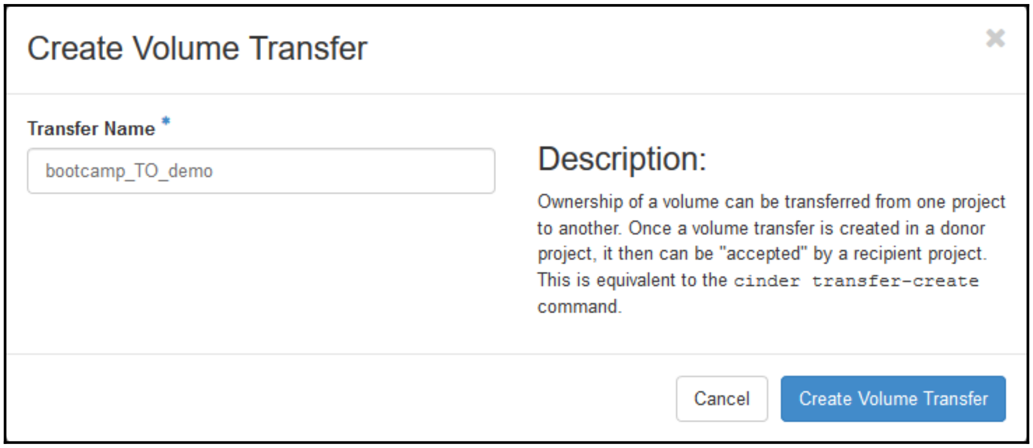
	Name	Description	Size	Status	Type	Attached To	Availability Zone	Bootable	Encrypted	Actions
☐	VL_hellovinoth_1GB	-	1GiB	Available	lvmdriver-1		nova	No	No	Edit Volume

Displaying 1 item

1. Click the drop-down button next to the **Edit Volume** button in the selected volume list. In response, you will see the list of available actions to manage the selected volume:



2. Click the **Create Transfer** button from the drop-down menu to initiate the volume transfer.
3. In response, you will get the **Create Volume Transfer** pop-up window. In that specify any name for the volume transfer for your reference and then click the **Create Volume Transfer** button:

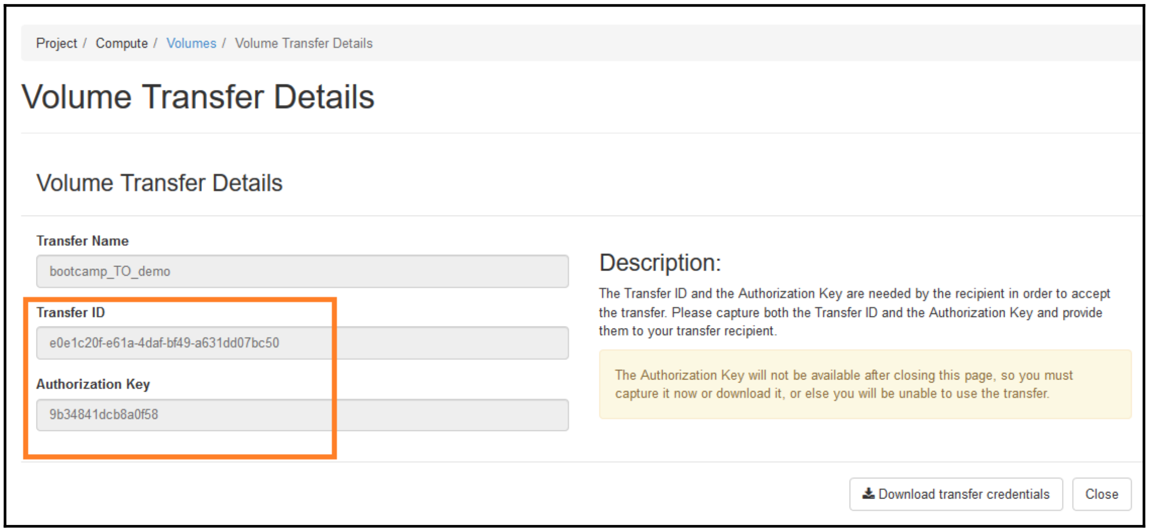


bootcamp_TO_demo

Description:

Ownership of a volume can be transferred from one project to another. Once a volume transfer is created in a donor project, it then can be "accepted" by a recipient project. This is equivalent to the `cinder transfer-create` command.

4. In response, you will get the **Volume Transfer Details** pop-up window displaying the **Transfer ID** and the **Authorization Key** for the volume transfer. Please take note of both the **Transfer ID** and the **Authorization Key**. Then click the **Close** button:



Transfer Name: bootcamp_TO_demo

Transfer ID: e0e1c20f-e61a-4daf-bf49-a631dd07bc50

Authorization Key: 9b34841dcb8a0f58

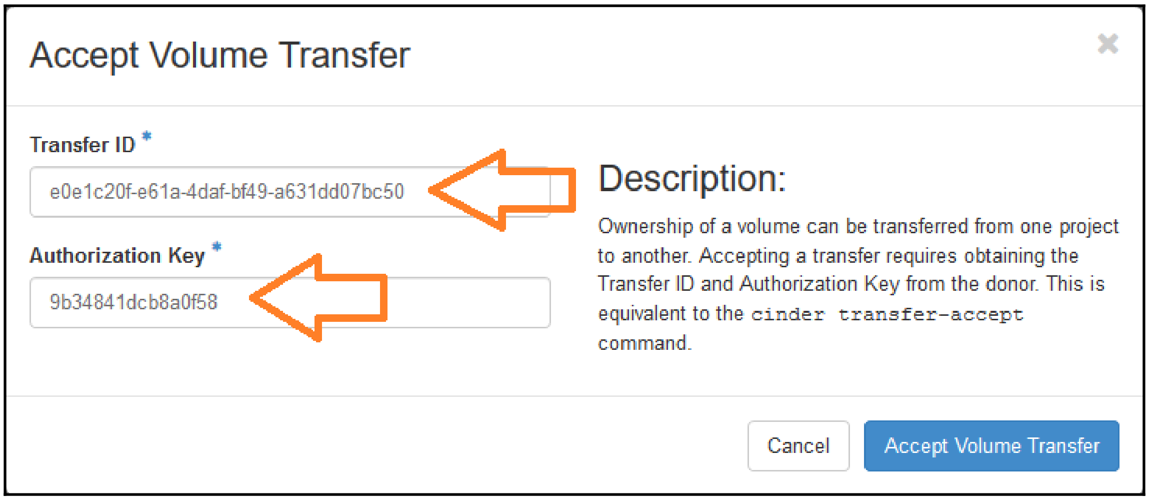
Description:

The Transfer ID and the Authorization Key are needed by the recipient in order to accept the transfer. Please capture both the Transfer ID and the Authorization Key and provide them to your transfer recipient.

The Authorization Key will not be available after closing this page, so you must capture it now or download it, or else you will be unable to use the transfer.

5. By now, you can see from the volume list that the selected volume is waiting for the transfer confirmation.
6. Sign out as the **bootcamp** user and sign back in using **demo** user credentials to accept the volume transfer to the project **demo**.
7. Navigate to the **Volumes** menu under the **Project** tab to see the list of available volumes.
8. Click the **Accept Transfer** button in the right top corner of the **Volumes** window.

9. In response, you will get the **Accept Volume Transfer** pop-up window. In that paste the **Transfer ID** and the **Authorization Key**, which we copied in *step 5*. Then click the **Accept Volume Transfer** button:


Accept Volume Transfer

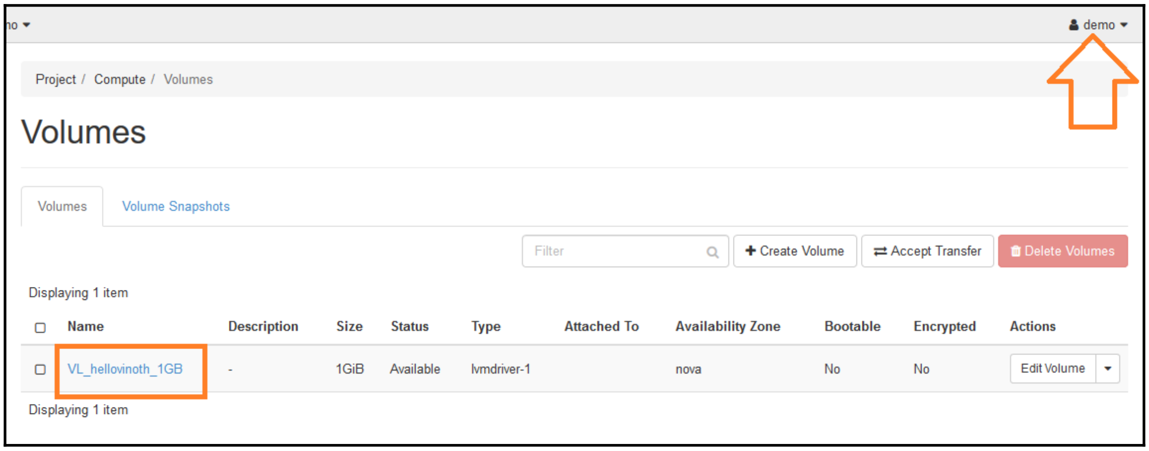
Transfer ID *
e0e1c20f-e61a-4daf-bf49-a631dd07bc50

Authorization Key *
9b34841dcb8a0f58

Description:

Ownership of a volume can be transferred from one project to another. Accepting a transfer requires obtaining the Transfer ID and Authorization Key from the donor. This is equivalent to the `cinder transfer-accept` command.

By now, you can see the volume being transferred from the project `hellovinoth.com` to the current **demo** project:



Volumes

Displaying 1 item

Name	Description	Size	Status	Type	Attached To	Availability Zone	Bootable	Encrypted	Actions
VL_hellovinoth_1GB	-	1GiB	Available	lvmdriver-1		nova	No	No	Edit Volume

Displaying 1 item

Summary

From this chapter, the reader will gain the hands-on experience with OpenStack Horizon and OSC CLI. The hands-on experience with OpenStack will have helped the reader in understanding how each component of OpenStack worked in bringing up the Cloud environment. Also, the method of learning OpenStack by doing so helps the reader to gain confidence in operating and administrating OpenStack.

In the next chapter, the reader will be provided with undisclosed tasks to take the chapter as exam.

7
Day 7 - Collective Training

After the brief field training session, it is time to cross-check our aptitude in administrating the OpenStack cloud environment. To do that, let's have a comprehensive evaluation by solving undisclosed tasks in OpenStack.

This chapter will include undisclosed tasks for readers to take this chapter as an exam. Comprehensive practice with admin and end-user use cases will test the reader's ability to manage the OpenStack cloud. The comprehensive practice session is categorized as follows:

- Administrative tasks
- Project specific tasks
- Extended activities

Administrative tasks

To complete the following listed administrative task, you need to use **admin** user account.

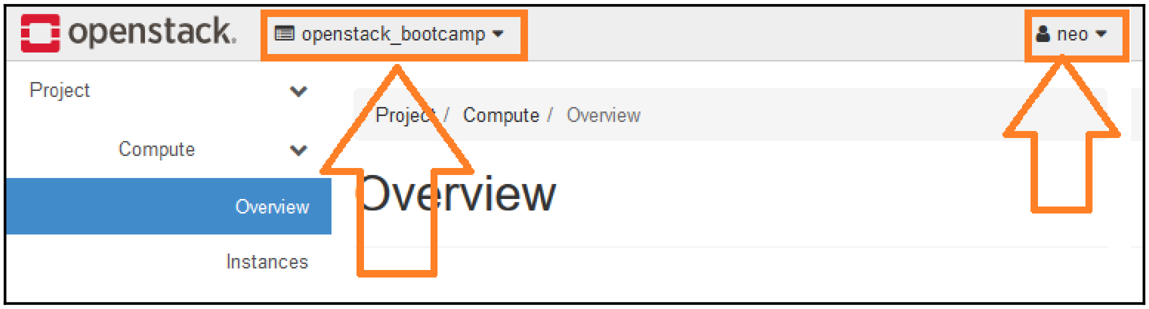
Task 1: Create a new project with the following details:

Parameter	**Value**
Name	`openstack_bootcamp`
Description	`Project for Collective training`

Task 2: Create an OpenStack user account `neo` with the the following data sheet:

Parameter	**Value**
Username	`neo`
Description	`User for Collective training`
Email	`neo@hellovinoth.com`
Password	`nomoresecret`
Primary Project	`openstack_bootcamp`
Role	`Member`

On successful completion of the above task, you should be able to login to Horizon using the username `neo` and password `nomoresecret`. Moreover, you should see the project window like the one shown here:

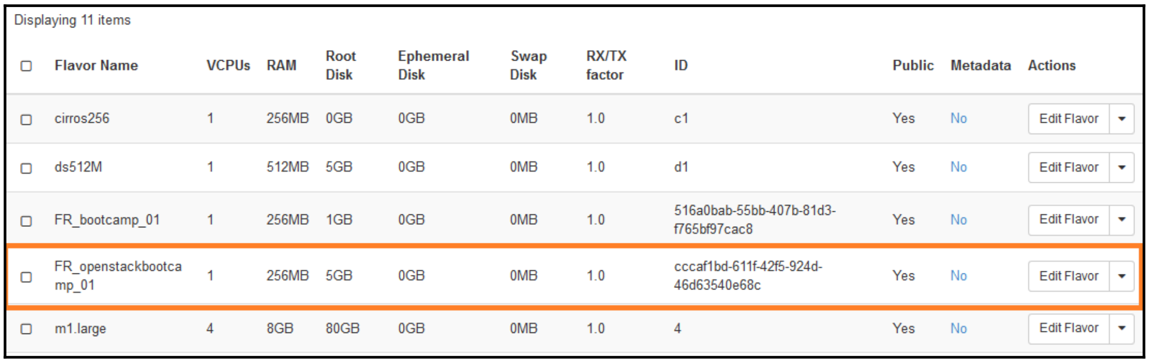


Task 3: Create a new flavor named `FR_openstackbootcamp_01` with the following data sheet:

Parameter	**Value**
Name	`FR_openstackbootcamp_01`
ID	`auto`
VCPUs	**1**
RAM	**256**
Root Disk	**5**
Ephemeral Disk	**0**

Swap Disk	**0**
RX/TX Factor	1

On successful completion, you should see the new flavor in the flavors list similar to the one shown here:



Displaying 11 items

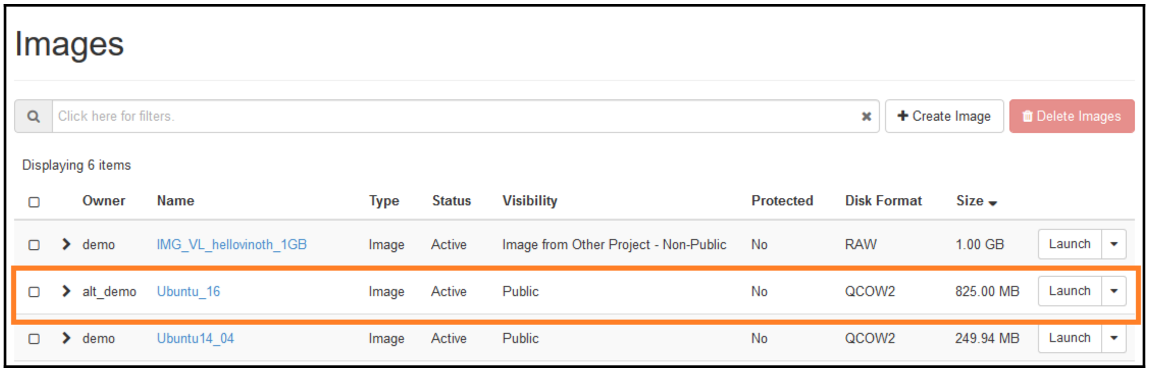
☐	Flavor Name	VCPUs	RAM	Root Disk	Ephemeral Disk	Swap Disk	RX/TX factor	ID	Public	Metadata	Actions
☐	cirros256	1	256MB	0GB	0GB	0MB	1.0	c1	Yes	No	Edit Flavor
☐	ds512M	1	512MB	5GB	0GB	0MB	1.0	d1	Yes	No	Edit Flavor
☐	FR_bootcamp_01	1	256MB	1GB	0GB	0MB	1.0	516a0bab-55bb-407b-81d3-f765bf97cac8	Yes	No	Edit Flavor
☐	FR_openstackbootcamp_01	1	256MB	5GB	0GB	0MB	1.0	cccaf1bd-611f-42f5-924d-46d63540e68c	Yes	No	Edit Flavor
☐	m1.large	4	8GB	80GB	0GB	0MB	1.0	4	Yes	No	Edit Flavor

Task 4: Create a public image in Glance using the the following data:

Parameter	**Value**
Image Name	`Ubuntu_16`
Source Type	**File**
File	`Downloaded image from local storage`
Format	`QCOW2 - QEMU Emulator`
Visibility	**Public**
Minimum Disk (GB)	**1**
Minimum RAM (MB)	**128**
Protected	**Yes**

Download Ubuntu cloud image file from the following link at: `https://cloud-images.ubuntu.com/releases/16.04/release-20170919/ubuntu-16.04-server-cloudimg-amd64-disk1.img`.

You should see the new image `Ubuntu_16` in the images list:



Images

Displaying 6 items

	Owner	Name	Type	Status	Visibility	Protected	Disk Format	Size
☐	demo	IMG_VL_hellovinoth_1GB	Image	Active	Image from Other Project - Non-Public	No	RAW	1.00 GB
☐	alt_demo	Ubuntu_16	Image	Active	Public	No	QCOW2	825.00 MB
☐	demo	Ubuntu14_04	Image	Active	Public	No	QCOW2	249.94 MB

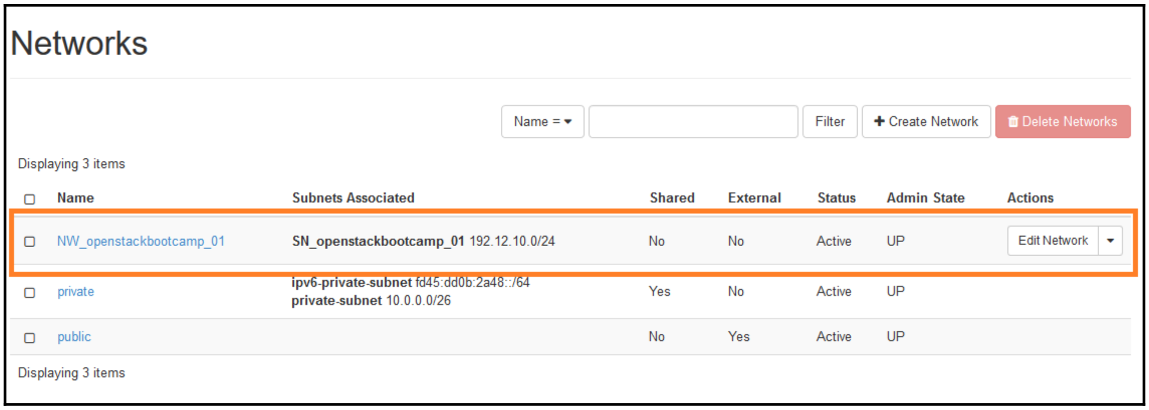
Project specific tasks

Complete the below tasks as the user `neo` in the project `openstack_bootcamp`.

Task 5: Create a private network named `NW_openstackbootcamp_01` under the project `openstack_bootcamp` with the following parameters:

Parameter	**Value**
Network Name	`NW_openstackbootcamp_01`
Subnet Name	`SN_openstackbootcamp_01`
Network Address Source	`Enter Network Address Manually`
Network Address	`192.12.10.0/24`
IP Version	`IPv4`
Gateway IP	`192.12.10.1`
DNS Name Servers	`8.8.8.8`

On successful completion, you should see the new private network in the network list:



Networks

Name = ▾ | Filter | + Create Network | Delete Networks

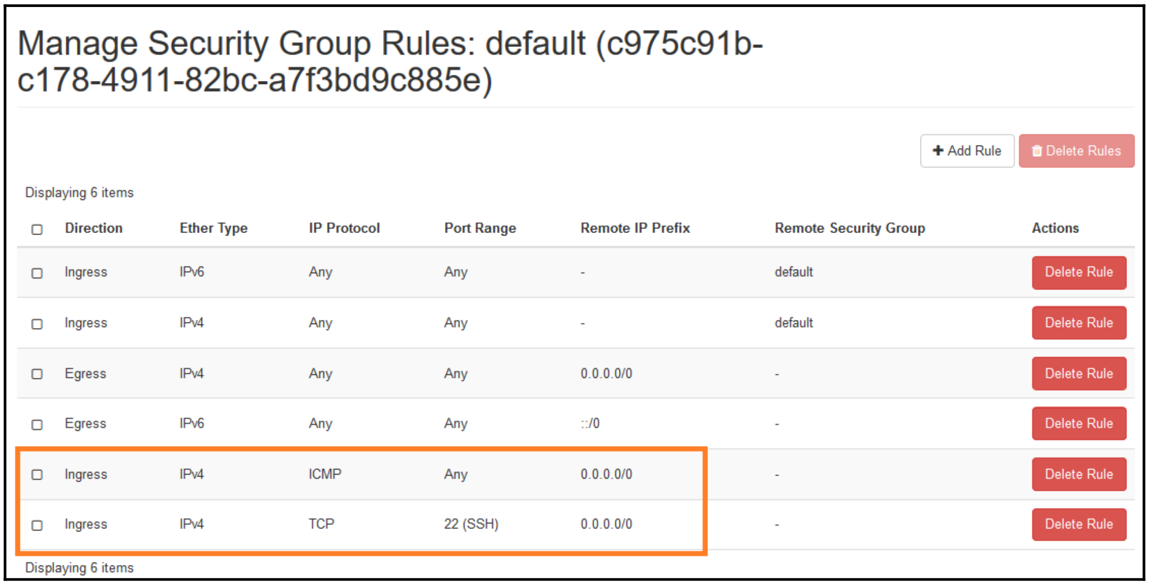
Displaying 3 items

	Name	Subnets Associated	Shared	External	Status	Admin State	Actions
☐	NW_openstackbootcamp_01	**SN_openstackbootcamp_01** 192.12.10.0/24	No	No	Active	UP	Edit Network ▾
☐	private	**ipv6-private-subnet** fd45:dd0b:2a48::/64 **private-subnet** 10.0.0.0/26	Yes	No	Active	UP	
☐	public		No	Yes	Active	UP	

Displaying 3 items

Task 6: Add new rules to the existing security group named `default` under the project `openstack_bootcamp` to allow access through SSH, and ICMP.

You should get the rule list similar to the one shown here:



Manage Security Group Rules: default (c975c91b-c178-4911-82bc-a7f3bd9c885e)

+ Add Rule | Delete Rules

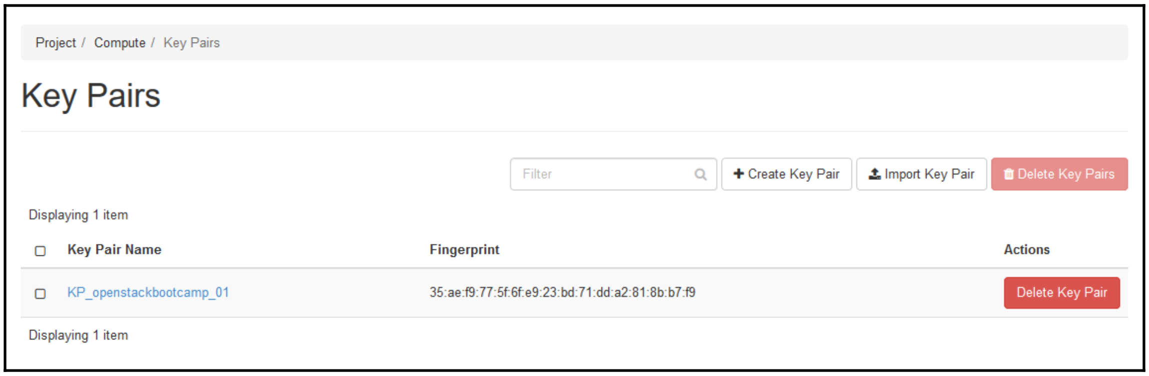
Displaying 6 items

	Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Actions
☐	Ingress	IPv6	Any	Any	-	default	Delete Rule
☐	Ingress	IPv4	Any	Any	-	default	Delete Rule
☐	Egress	IPv4	Any	Any	0.0.0.0/0	-	Delete Rule
☐	Egress	IPv6	Any	Any	::/0	-	Delete Rule
☐	Ingress	IPv4	ICMP	Any	0.0.0.0/0	-	Delete Rule
☐	Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	Delete Rule

Displaying 6 items

Task 7: Generate a new public keypair named `KP_openstackbootcamp_01` to use with instance SSH authentication.

You should see the keypair list like the one shown here:


Project / Compute / Key Pairs

Key Pairs

Displaying 1 item

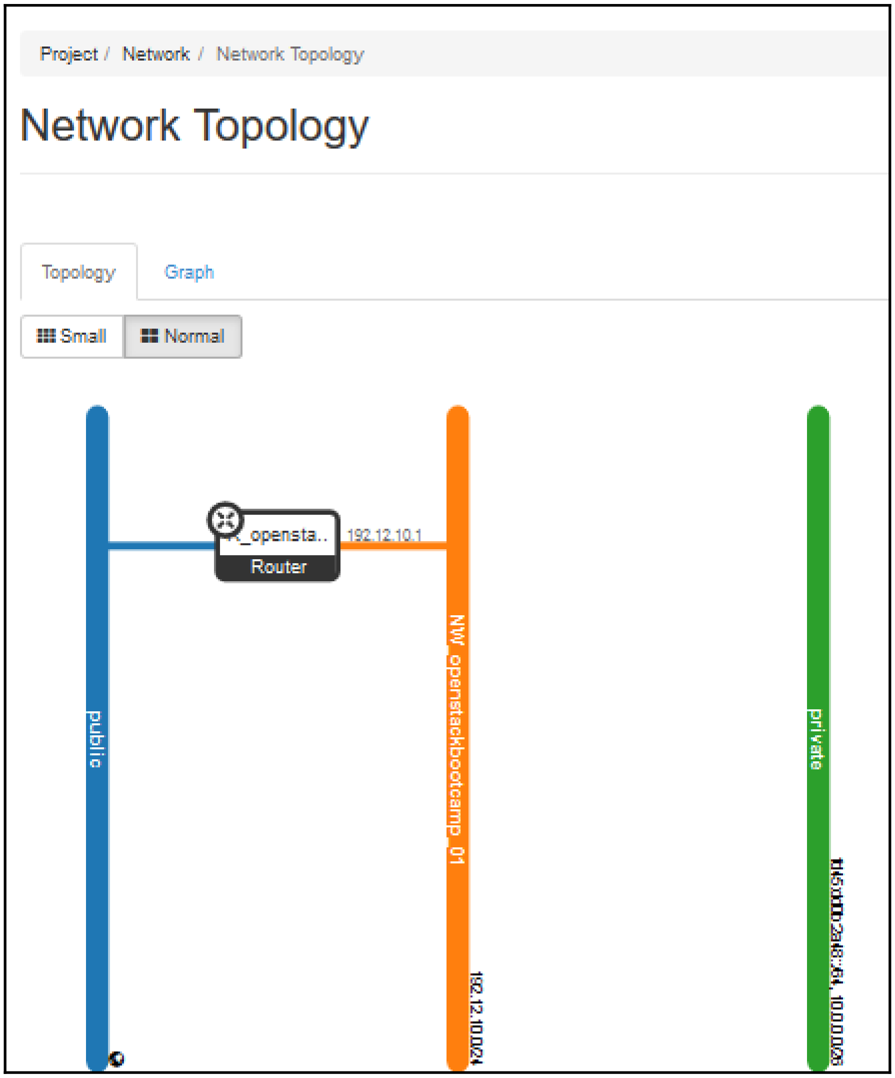
Key Pair Name	Fingerprint	Actions
KP_openstackbootcamp_01	35:ae:f9:77:5f:6f:e9:23:bd:71:dd:a2:81:8b:b7:f9	Delete Key Pair

Displaying 1 item

TIP: To use this keypair for SSH login, you need to copy the downloaded keypair file from your local PC to the OpenStack controller. This is because the DevStack setup has limitations in connecting and accessing the floating IP outside the OpenStack cluster.

Task 8: Create the new router named `R_openstackbootcamp_01` under the project `openstack_bootcamp` to connect the `NW_openstackbootcamp_01` private network to the public network.

On successful completion of the preceding task, you should see the **Network Topology** diagram similar to the one shown here:



Task 9: Create a new instance name `VM_openstackbootcamp_01` under the project `openstack_bootcamp` using the following data sheet:

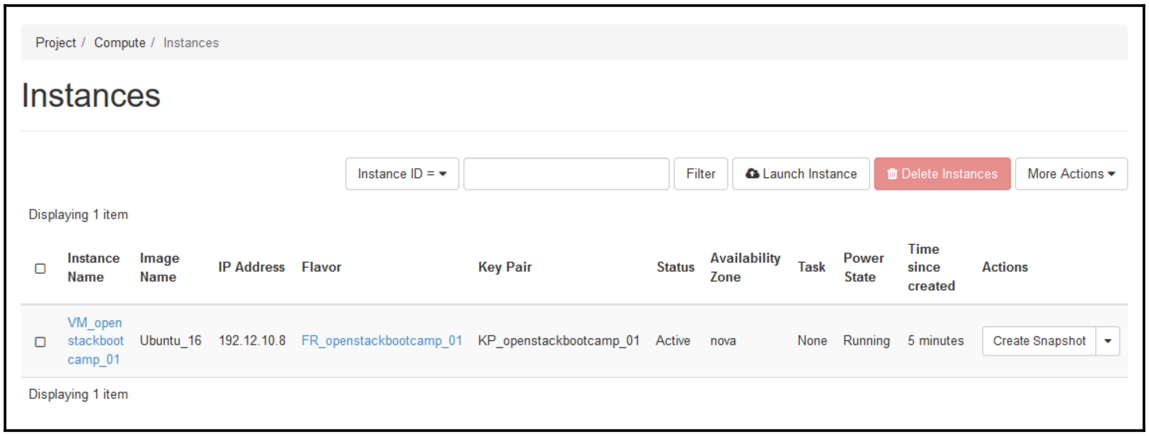
Details Tab	
Instance Name*	`VM_openstackbootcamp_01`
Availability Zone	`nova`

Count*	1
Source Tab	
Select Boot Source	`Image`
Create New Volume	`No`
Image Name	`Ubuntu_16`

Flavor Tab	
Allocated Flavor name	`FR_openstackbootcamp_01`

Networks Tab	
Allocated Network Name	`NW_openstackbootcamp_01`
Security Groups Tab	
Allocated Security Group Name	`Default`
Key Pair Tab	
Key Pair Name	`KP_openstackbootcamp_01`

You should see the newly provisioned instance in the instance list like the one shown here:



Displaying 1 item

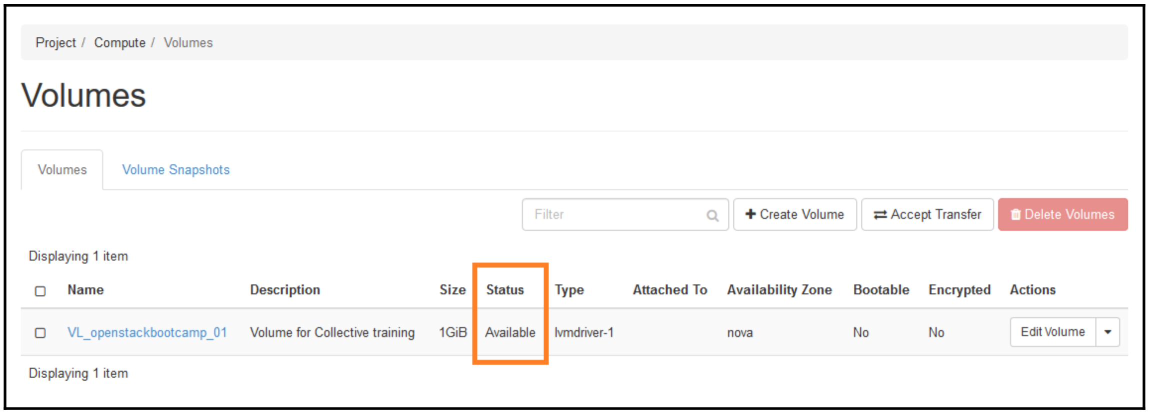
	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
☐	VM_openstackbootcamp_01	Ubuntu_16	192.12.10.8	FR_openstackbootcamp_01	KP_openstackbootcamp_01	Active	nova	None	Running	5 minutes	Create Snapshot

Displaying 1 item

Task 10: Create a new empty volume named `VL_openstackbootcamp_01` under the project `openstack_bootcamp` with the following parameters:

Parameter	**Value**
Volume Name	`VL_openstackbootcamp_01`
Description	`Volume for Collective training`
Volume Source	`No Source, Empty Volume`
Type	`lvmdriver-1`
Size	`1`
Availability Zone	`nova`

On successful creation of Cinder volume with the parameters mentioned previously, you should see the new volume in the volume list similar to the one in the following figure:



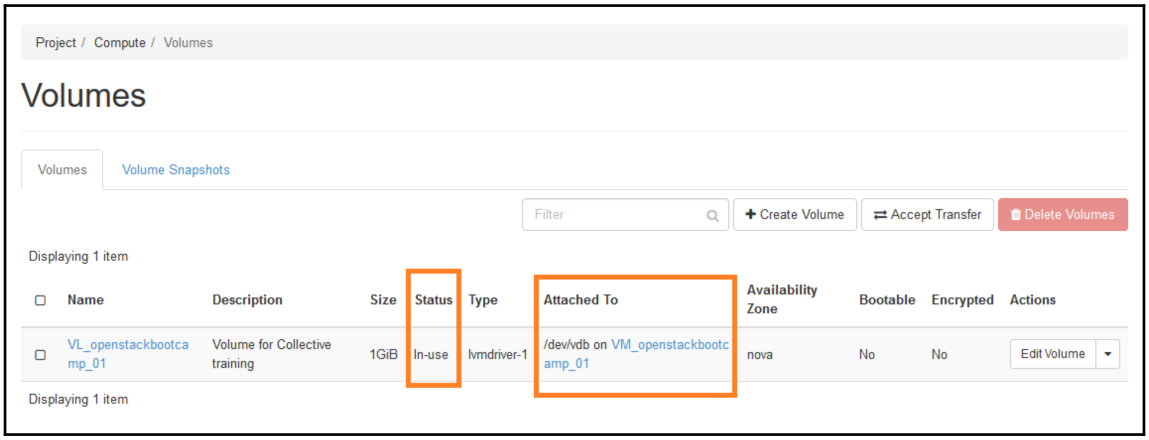
Displaying 1 item

Name	Description	Size	Status	Type	Attached To	Availability Zone	Bootable	Encrypted	Actions
VL_openstackbootcamp_01	Volume for Collective training	1GiB	Available	lvmdriver-1		nova	No	No	Edit Volume

Displaying 1 item

Task 11: Attach the created volume `VL_openstackbootcamp_01` to the new instance `VM_openstackbootcamp_01`.

After attaching the volume to the virtual machine, you should see the volume status like the one in the figure:



Project / Compute / Volumes

Volumes

Volumes | Volume Snapshots

Filter | + Create Volume | ⇄ Accept Transfer | Delete Volumes

Displaying 1 item

	Name	Description	Size	Status	Type	Attached To	Availability Zone	Bootable	Encrypted	Actions
☐	VL_openstackbootcamp_01	Volume for Collective training	1GiB	In-use	lvmdriver-1	/dev/vdb on VM_openstackbootcamp_01	nova	No	No	Edit Volume

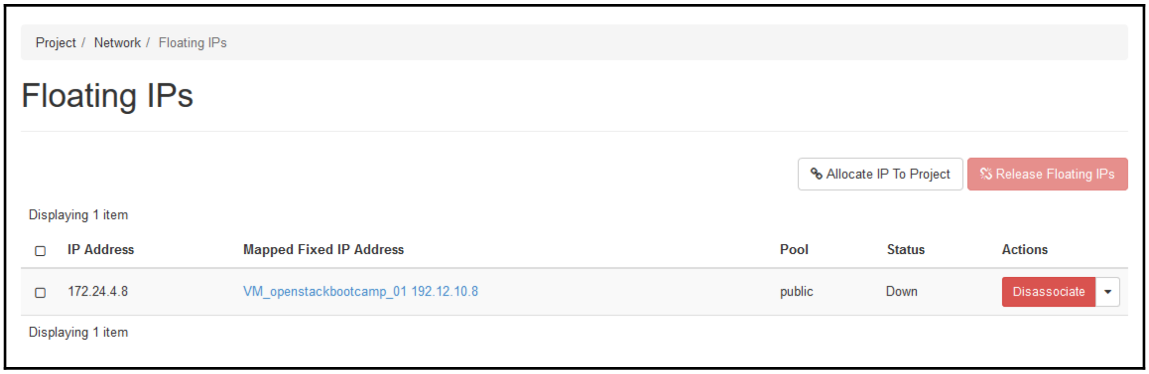
Displaying 1 item

Task 12: Associate floating IP to the virtual machine `VM_openstackbootcamp_01`.

To associate floating IP through the Horizon, you should use the **Floating IPs** menu from the left navigation panel under the **Network** panel on the **Project** tab.

Alternatively, you could use the **Associate Floating IP** option from the **Actions** drop-down menu on the **Instance** window to allocate and associate floating IP from the public network to the instance.

You could verify the floating IP association from the **Floating IPs** menu. You will get the window displayed like the one shown in the figure:


Project / Network / Floating IPs

Floating IPs

Displaying 1 item

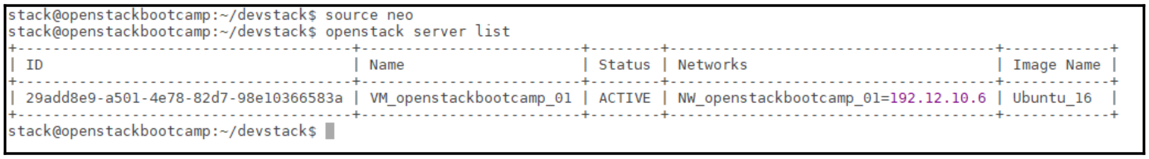
IP Address	Mapped Fixed IP Address	Pool	Status	Actions
172.24.4.8	VM_openstackbootcamp_01 192.12.10.8	public	Down	Disassociate

Displaying 1 item

Task 13: Without accessing Horizon dashboard, build a user credential file using the below information to get access to CLI OpenStack client:

Parameter	**Value**
`IDENTITY_API_VERSION`	`3`
`AUTH_URL`	`http://192.168.1.6/identity_admin`
`USERNAME`	`neo`
`USER_DOMAIN`	`default`
`PASSWORD`	`nomoresecret`
`PROJECT_NAME`	`openstack_bootcamp`
`PROJECT_DOMAIN`	`default`
`REGION_NAME`	`RegionOne`

You could verify the environment file build by using the CLI to list the instance under `openstack_bootcamp` project. You should see the output like the one shown here:

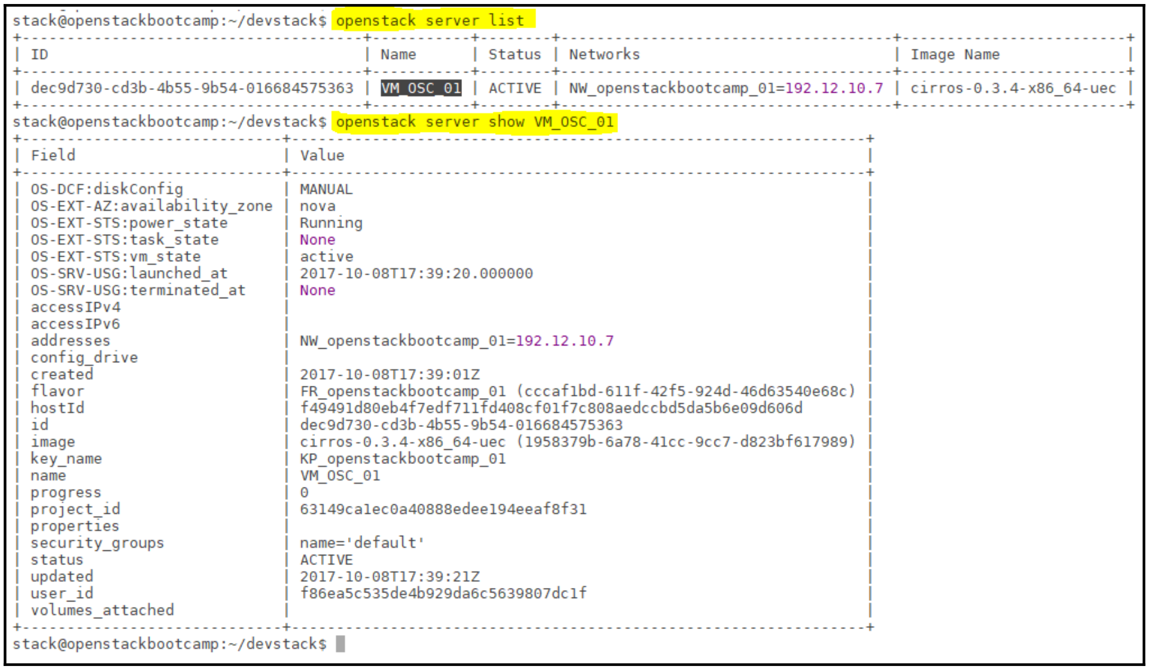

```
stack@openstackbootcamp:~/devstack$ source neo
stack@openstackbootcamp:~/devstack$ openstack server list
+--------------------------------------+-------------------------+--------+--------------------------------------+------------+
| ID                                   | Name                    | Status | Networks                             | Image Name |
+--------------------------------------+-------------------------+--------+--------------------------------------+------------+
| 29add8e9-a501-4e78-82d7-98e10366583a | VM_openstackbootcamp_01 | ACTIVE | NW_openstackbootcamp_01=192.12.10.6  | Ubuntu_16  |
+--------------------------------------+-------------------------+--------+--------------------------------------+------------+
stack@openstackbootcamp:~/devstack$
```

Task 14: Delete the existing virtual machine named `VM_openstackbootcamp_01` using the OSC CLI.

Task 15: Through OpenStack Client, launch a new instance named `VM_OSC_01` under the project `openstack_bootcamp` using the following details:

Parameter	**Value**
VM Name	`VM_OSC_01`
Flavor	`FR_openstackbootcamp_01`
Image	`cirros-0.3.4-x86_64-uec`
Network	`NW_openstackbootcamp_01`
Security Group	`default`
Key Pair Name	`KP_openstackbootcamp_01`

On successful provision of a new virtual machine using OSC CLI, you should get the output similar to the one with the instance details satisfying the above-tabulated parameters:



```
stack@openstackbootcamp:~/devstack$ openstack server list
+--------------------------------------+-----------+--------+-------------------------------------+-------------------------+
| ID                                   | Name      | Status | Networks                            | Image Name              |
+--------------------------------------+-----------+--------+-------------------------------------+-------------------------+
| dec9d730-cd3b-4b55-9b54-016684575363 | VM_OSC_01 | ACTIVE | NW_openstackbootcamp_01=192.12.10.7 | cirros-0.3.4-x86_64-uec |
+--------------------------------------+-----------+--------+-------------------------------------+-------------------------+
stack@openstackbootcamp:~/devstack$ openstack server show VM_OSC_01
+-----------------------------+----------------------------------------------------------------+
| Field                       | Value                                                          |
+-----------------------------+----------------------------------------------------------------+
| OS-DCF:diskConfig           | MANUAL                                                         |
| OS-EXT-AZ:availability_zone | nova                                                           |
| OS-EXT-STS:power_state      | Running                                                        |
| OS-EXT-STS:task_state       | None                                                           |
| OS-EXT-STS:vm_state         | active                                                         |
| OS-SRV-USG:launched_at      | 2017-10-08T17:39:20.000000                                     |
| OS-SRV-USG:terminated_at    | None                                                           |
| accessIPv4                  |                                                                |
| accessIPv6                  |                                                                |
| addresses                   | NW_openstackbootcamp_01=192.12.10.7                            |
| config_drive                |                                                                |
| created                     | 2017-10-08T17:39:01Z                                           |
| flavor                      | FR_openstackbootcamp_01 (cccaf1bd-611f-42f5-924d-46d63540e68c) |
| hostId                      | f49491d80eb4f7edf711fd408cf01f7c808aedccbd5da5b6e09d606d       |
| id                          | dec9d730-cd3b-4b55-9b54-016684575363                           |
| image                       | cirros-0.3.4-x86_64-uec (1958379b-6a78-41cc-9cc7-d823bf617989) |
| key_name                    | KP_openstackbootcamp_01                                        |
| name                        | VM_OSC_01                                                      |
| progress                    | 0                                                              |
| project_id                  | 63149ca1ec0a40888edee194eeaf8f31                               |
| properties                  |                                                                |
| security_groups             | name='default'                                                 |
| status                      | ACTIVE                                                         |
| updated                     | 2017-10-08T17:39:21Z                                           |
| user_id                     | f86ea5c535de4b929da6c5639807dc1f                               |
| volumes_attached            |                                                                |
+-----------------------------+----------------------------------------------------------------+
stack@openstackbootcamp:~/devstack$
```

Task 16: Using OSC CLI, attach floating IP to the instance `VM_OSC_01`.

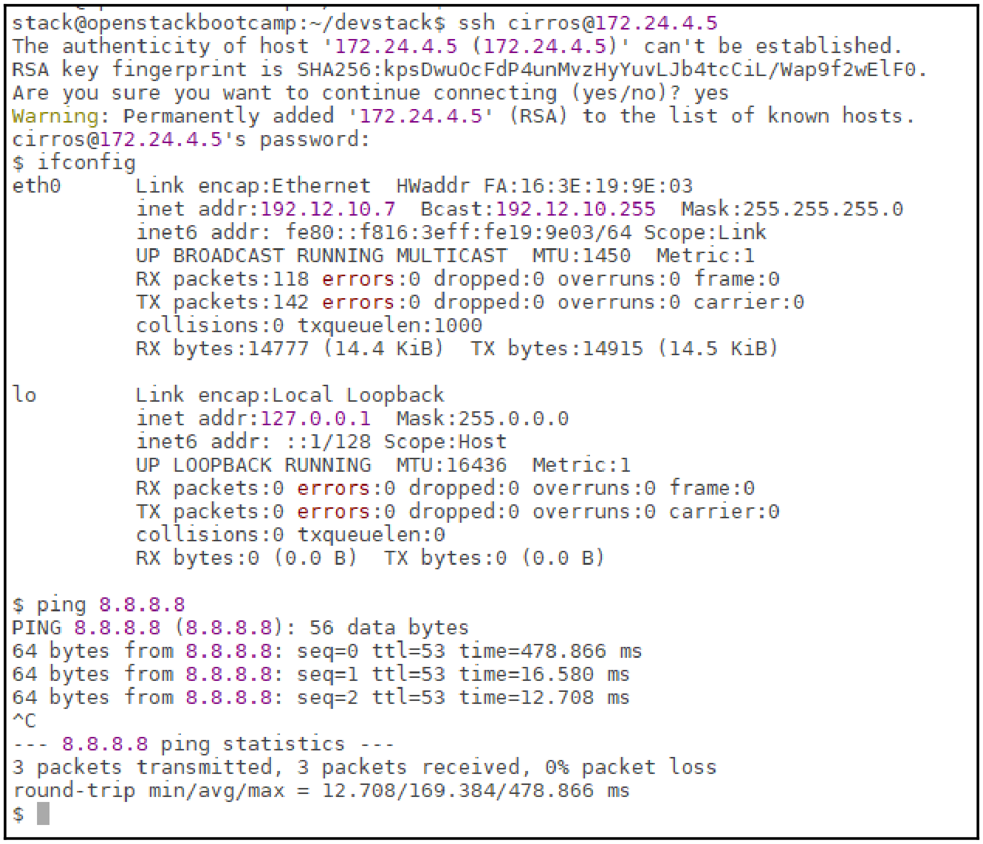
On successful completion of the floating IP association, you should be able to ping the floating IP of the instance `VM_OSC_01`.

Task 17: Access the instance `VM_OSC_01` through SSH connection using the floating IP recently attached. You may use username `cirros` and password `cubswin:)` to access the `cirros` instance.

Alternatively, you could also use the key pair `KP_openstackbootcamp_01` for SSH authentication instead of using the password.

Task 18: Verify the external network connectivity on the instance `VM_OSC_01`.

You could refer to the following figure to verify the completion of the task mentioned previously:



```
stack@openstackbootcamp:~/devstack$ ssh cirros@172.24.4.5
The authenticity of host '172.24.4.5 (172.24.4.5)' can't be established.
RSA key fingerprint is SHA256:kpsDwu0cFdP4unMvzHyYuvLJb4tcCiL/Wap9f2wElF0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '172.24.4.5' (RSA) to the list of known hosts.
cirros@172.24.4.5's password:
$ ifconfig
eth0      Link encap:Ethernet  HWaddr FA:16:3E:19:9E:03
          inet addr:192.12.10.7  Bcast:192.12.10.255  Mask:255.255.255.0
          inet6 addr: fe80::f816:3eff:fe19:9e03/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
          RX packets:118 errors:0 dropped:0 overruns:0 frame:0
          TX packets:142 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14777 (14.4 KiB)  TX bytes:14915 (14.5 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=53 time=478.866 ms
64 bytes from 8.8.8.8: seq=1 ttl=53 time=16.580 ms
64 bytes from 8.8.8.8: seq=2 ttl=53 time=12.708 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 12.708/169.384/478.866 ms
$
```

Extended activities

Complete the task using the data points provided here. You are free to do any changes in the OpenStack environment to satisfy the following requisite.

Activity 1

Under the project `openstack_bootcamp`. Create a new instance named `VM_activity_01` of size 256MB RAM / 3GB Root Disk / 1vCPU using the image `cirros-0.3.4-x86_64-uec` in the network named `private`.

Activity 2

Provision new `cirros` instance named `VM_activity_02` under the project `demo` using the flavor `FR_bootcamp_01` on the shared network `NW_openstackbootcamp_01`.

Activity 3

Create a new empty Cinder volume named `VL_activity_03` of a 1GB size under the project `openstack_bootcamp`. Then attach the volume `VL_activity_03` to the instance `VM_activity_02` under the project `demo`.

Summary

By now, the reader will have gained the confidence to perform administrative and project-oriented task in OpenStack. All of the above collective training and challenging tasks will help the reader to unleash the power of hands-on experience in understanding the role of each component in OpenStack.

In the next chapter, I will guide you through a step-by-step procedure for building an OpenStack private cloud from scratch.

8
Day 8 - Build Your OpenStack

The DevStack setup we are using so far in the previous chapter was built using an automated script to create an OpenStack development environment quickly. The DevStack setup cannot be used for the production-ready setup. Moreover, the DevStack build is not and has never been intended to be a general OpenStack Installer.

Notably, many third-party vendors offer production-ready cloud software based on OpenStack that provide deployment and management strategies using third-party tools like Chef, Puppet, Fuel, Ansible, and other tools.

Optionally, you may also recall the available OpenStack distributions from `Chapter 1`, *Day 1 - Build Your Camp*.

In this chapter, I will walk through the step-by-step process of installing the OpenStack cloud manually. The installation procedure documented in this chapter is based on the OpenStack Installation Guide for the Ubuntu 16.04 LTS operating system, which is found at: `http://docs.openstack.org/`.

The step-by-step procedure for building an OpenStack private cloud from scratch covers these topics:

- Preparing your Virtual machine
- Setup prerequisites
- Configuring the database server
- Configuring the message queue
- Configuring the `memcached` server
- Configuring the identity service (Keystone)
- Configuring the image service (Glance)
- Configuring the compute service (Nova)
- Installing and configuring a compute node (nova-compute)

- Configuring the networking service (neutron)
- Installing and configuring a compute node (neutron)
- Installing the OpenStack dashboard
- Adding the compute node to the OpenStack cluster

This guide will walk through the OpenStack installation by using the packages available through the Canonical Ubuntu Cloud archive repository for Ubuntu 16.04 (LTS).

System requirements

The OpenStack components are intended to run on any standard hardware that ranges from desktop machines to enterprise-grade servers, with the only limitations being that the processors of the compute nodes need to support virtualization technologies, such as Intel's VT-x or AMD's AMD-v technologies.

However, for learning purposes, we could even build our OpenStack cloud on the virtual machine like the one we did for DevStack setup. Moreover, all of the instructions for building the OpenStack in the enterprise-grade servers and in the virtual machine are the same.

To build our OpenStack setup on the virtual machine, we need to meet the following minimum hardware requirements:

This chapter assumes that you have access to the virtual machine which has the Ubuntu 16.04 LTS operating system installed with a minimum of 6 GB RAM and 30 GB HDD.

Downloading and installing the VirtualBox and creating new virtual machine is not in the scope of this book. There are lots of free tutorials available online for bringing up your new virtual machine with the specification mentioned previously.

Preparing your virtual machine

To build a simple working OpenStack cloud, you must have the following requirements configured in the virtual box environment before we start the step-by-step manual installation process:

- Ubuntu 16.04 LTS Operating System
- 6GB RAM
- 10GB Disk Space

- 2 vCPUs
- 2 NIC (Adapter 1 - Bridged Adapter and Adapter 2 - Bridged Adapter)

Adding the second adapter to the virtual machine may require manual configuration changes in the OS network interface file. Make sure you have added the second interface with DHCP settings in the `/etc/network/interfaces` file and ensure both the NIC obtained the IP address.

Then, perform `apt-get update & dist-upgrade` and reboot the machine.

Before we begin

Before we start installing the OpenStack, some work must be done as part of environment preparation for a successful installation.

User permissions

OpenStack services can be installed and configured either as the `root` user or as a user with `sudo` privileges. I personally recommend going with root user access.

Configuring network interfaces

As we have 2 NICs added to the virtual machines, let's have the first NIC dedicated to the management and the VM tunnel network traffic. The later NIC is dedicated to the provider network (external) traffic.

Step 1 - configuring the first network interface with static IP address

By now, we have two NICs added in the virtual machine, assigned with the DHCP IP address. Choose any one NIC of your choice and configure the IP address statically. Therefore, the IP will stay fixed for our virtual machine, even after the reboot.

In my case, the virtual machine has two NICs, namely `enp0s3` and `enp0s8`. I have chosen the first NIC `enp0s3` for management traffic and the next NIC `enp0s8` for provider network configuration. Initially, my virtual machine with two bridged adapter networks was assigned with the DHCP IP address `192.168.1.7`, and `192.168.1.8` for `enp0s3` and `enp0s8` respectively. I have modified my first NIC with the static IP address with the same network configuration as the one I received with DHCP by editing the `/etc/network/interfaces` file.

Step 2 - configuring the second interface as the provider interface

To configure the NIC as a provider interface, we need to use a unique network configuration in a network interface file which will enable the NIC without an IP address assigned to it.

Modify the `/etc/network/interfaces` file to contain the following configurations for the chosen provider NIC:

```
# The provider network interface
auto INTERFACE_NAME
iface INTERFACE_NAME inet manual
up ip link set dev $IFACE up
down ip link set dev $IFACE down
```

Replace `INTERFACE_NAME` with the actual interface name (in my case, `enp0s8`) and reboot the server.

Here is the sample network interface file (`/etc/network/interfaces`) from my setup for your reference:

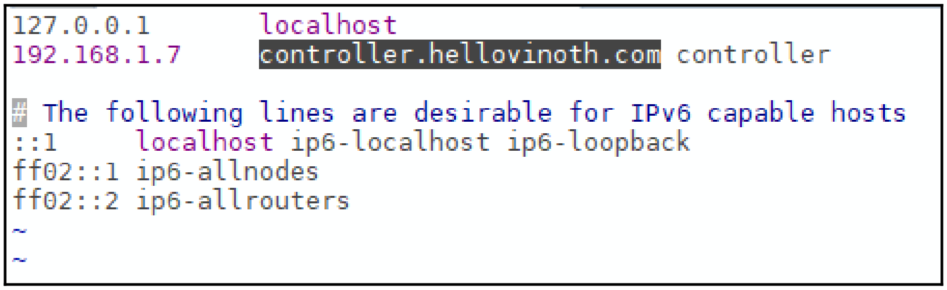
```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*
# The loopback network interface
auto lo
iface lo inet loopback
# The primary network interface
auto enp0s3
iface enp0s3 inet static
        address 192.168.1.7
        network 192.168.1.0
        netmask 255.255.255.0
        gateway 192.168.1.1
        dns-nameservers 8.8.8.8
        broadcast 192.168.1.255
# The provider network interface
auto enp0s8
iface enp0s8 inet manual
up ip link set dev $IFACE up
down ip link set dev $IFACE down
```

After rebooting the server, you should see the two NIC is in active status. The first NIC with the static IP assigned and the later with no IP address assigned to it.

Step 3 - setting the hostnames

Before installing OpenStack, be sure that the server has been configured with the proper hostname and the local DNS name resolution.

Using a text editor, change the value as `controller.hellovinoth.com` in the `/etc/hostname` file on the server. Then, update the `/etc/hosts` file on the server to include the management IP address and the hostname like the one here:



```
127.0.0.1       localhost
192.168.1.7     controller.hellovinoth.com controller

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
~
~
```

Step 4 - verifing network connectivity

To verify that the NIC is configured correctly, try to access the internet (ping `www.google.com`) from the virtual machine after the reboot.

Configuring the Network Time Protocol

A time synchronization package, such as NTP, is a prerequisite, as OpenStack services depend on consistent and synchronized time between the controller, network and compute nodes. For example, the Nova service should synchronize the time across the hosts to avoid time conflicts when scheduling VM provisions on the compute nodes. Also, other services will experience similar issues when the time is not synchronized.

To install NTP, issue the following commands on all of the nodes in the environment:

```
apt install chrony
service chrony restart
```

Configuring the OpenStack repository

Installation of OpenStack on Ubuntu uses packages from the cloud-archive `apt` repository. To enable the cloud-archive repository, download and install the OpenStack Ocata release packages, and execute the following commands on all hosts:

```
apt install software-properties-common
add-apt-repository cloud-archive:ocata
```

Upgrading the system

Before we begin the OpenStack installation, it is recommended that the kernel and other system packages on each node be upgraded to the latest version supported by Ubuntu 16.04 LTS.

To do that, issue the following command on each node, followed by a reboot to allow the changes to take effect:

```
apt update && apt dist-upgrade -y
reboot
apt install python-openstackclient crudini -y
```

OpenStack installation

The steps in the later part of the chapter, document the building your own OpenStack which includes installation, and configuration of KeyStone, Glance, Nova compute, neutron, and Horizon, on a single node (all-in-one step-up). As we are building an all-in-one single node OpenStack setup, the node representation as controller/network/compute node represent the same all-in-one node throughout the *OpenStack installation* section.

Configuring the database server

On the controller node, run the following command to install the MySQL database server:

```
apt install mariadb-server python-pymysql
```

Once the `mariadb` package installation is completed, do the following-mentioned configuration steps to set the IP address that the MariaDB service will bind to, and allow the connectivity to the MariaDB server from other hosts in the environment.

Step 1 - creating file

Create the `/etc/mysql/mariadb.conf.d/90-openstack.cnf` file and paste the following mentioned configuration lines in it:

```
[mysqld]
bind-address = 192.168.1.7
default-storage-engine = innodb
innodb_file_per_table = on
max_connections = 4096
collation-server = utf8_general_ci
character-set-server = utf8
```

The bind-address value should be the management IP of the controller node. In my case, my management IP address is `192.168.1.7`. You may refer the *Configuring network interfaces* section to find your management IP address.

Step 2 - finalizing the database installation

The MySQL secure installation utility is used to build the default MySQL database and set a password for the MySQL root user. The following command will secure the database server by running the MySQL secure installation script:

```
service mysql restart
mysql_secure_installation
```

While running the MySQL secure installation script, you will be prompted to enter a password and change various settings. For this installation, the chosen `root` password is `bootcamp`. Alternatively, you may choose a more secure password of your choice.

Answer [Y] to the remaining questions to exit the configuration process. At this point, MySQL server has been successfully installed on the controller node.

Step 3 - creating database for OpenStack services

Before we install and configure the OpenStack service, we must create a dedicated database for each service in the MariaDB server. Copy and paste the following commands to create and grant proper access to each service database:

```
echo "CREATE DATABASE keystone;"|mysql
echo "GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost'
IDENTIFIED BY 'bootcamp';"|mysql
echo "GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' IDENTIFIED
BY 'bootcamp';"|mysql
echo "CREATE DATABASE glance;"|mysql
echo "GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'localhost'
 IDENTIFIED BY 'bootcamp';"|mysql
echo "GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%' IDENTIFIED
BY 'bootcamp';"|mysql
echo "CREATE DATABASE nova_api;"|mysql
echo "GRANT ALL PRIVILEGES ON nova_api.* TO 'nova'@'localhost'
IDENTIFIED BY 'bootcamp';"|mysql
echo "GRANT ALL PRIVILEGES ON nova_api.* TO 'nova'@'%' IDENTIFIED
BY 'bootcamp';"|mysql
echo "CREATE DATABASE nova;"|mysql
echo "GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost'
IDENTIFIED BY 'bootcamp';"|mysql
echo "GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%' IDENTIFIED
BY 'bootcamp';"|mysql
echo "CREATE DATABASE nova_cell0;"|mysql
echo "GRANT ALL PRIVILEGES ON nova_cell0.* TO 'nova'@'localhost'
 IDENTIFIED BY 'bootcamp';"|mysql
echo "GRANT ALL PRIVILEGES ON nova_cell0.* TO 'nova'@'%' IDENTIFIED
BY 'bootcamp';"|mysql
echo "CREATE DATABASE neutron;"|mysql
echo "GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'localhost'
 IDENTIFIED BY 'bootcamp';"|mysql
echo "GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'%' IDENTIFIED
 BY 'bootcamp';"|mysql
echo "CREATE DATABASE cinder;"|mysql
echo "GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'localhost'
IDENTIFIED BY 'bootcamp';"|mysql
echo "GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%' IDENTIFIED
BY 'bootcamp';"|mysql
```

In all of the preceding commands, I have given the same DB passwords as `bootcamp` for all of the DB users. Notably, throughout this chapter, I will be using `bootcamp` as the password for all of the authentication configuration. Optionally, you may replace it with any password of your choice. However, don't forget to have a note of all of the passwords.

Using the preceding command, we have created the database KeyStone, Glance, `nova_api`, nova, `nova_cell0`, neutron, Cinder, and then we grant proper permission to allow the connectivity to the selected database from other hosts in the environment.

Configuring the message queue

Advanced Message Queue Protocol (**AMQP**) is the messaging technology chosen to use with OpenStack-based cloud components such as Nova, Glance, Cinder, and neutron to communicate internally via AMQP and to each other using API calls.

The following are instructions to install RabbitMQ, an AMQP broker. Popular alternatives include Qpid and ZeroMQ. On the controller node, follow the steps to install and configure the messaging server:

```
apt install rabbitmq-server
rabbitmqctl add_user openstack bootcamp
rabbitmqctl set_permissions openstack ".*" ".*" ".*"
```

You may notice that in the preceding command, I have chosen the RabbitMQ password as `bootcamp` for the user `openstack`.

Configuring the memcached server

On the controller node, follow the steps to install and configure the `memcached` server:

```
apt install memcached python-memcache
```

Once the package installation is completed, edit the `/etc/memcached.conf` file and replace your management IP address in the existing line that had `-l 127.0.0.1` and restarts the `memcached` service:

```
service memcached restart
```

Configuring the identity service (KeyStone)

KeyStone is the identity service for OpenStack and is used to authenticate and authorize users and services in the OpenStack cloud.

Step 1 - installing and configure components

On the controller node, copy and paste the following commands one by one to install and configure the OpenStack KeyStone service:

```
apt install keystone
crudini --set /etc/keystone/keystone.conf database connection
mysql+pymysql://keystone:bootcamp@controller/keystone
crudini --set /etc/keystone/keystone.conf token provider fernet
su -s /bin/sh -c "keystone-manage db_sync" keystone
keystone-manage fernet_setup --keystone-user keystone --keystone-group
keystone
keystone-manage credential_setup --keystone-user keystone --keystone-group
keystone
keystone-manage bootstrap --bootstrap-password bootcamp \
--bootstrap-admin-url http://controller.hellovinoth.com:35357/v3/ \
--bootstrap-internal-url http://controller.hellovinoth.com:5000/v3/ \
--bootstrap-public-url http://controller.hellovinoth.com:5000/v3/ \
--bootstrap-region-id RegionOne
```

Replace `bootcamp` with the password you chose for the database and admin user. Also, the hostname `controller.hellovinoth.com` with the hostname of your server.

Step 2 - configuring the Apache HTTP server

Add the `ServerName` parameter to reference the controller node in the `/etc/apache2/apache2.conf` file and remove the default SQLite database. To do that, execute the following commands:

```
sed -i -e '1iServerName controller.hellovinoth.com\'
/etc/apache2/apache2.conf
service apache2 restart
rm -f /var/lib/keystone/keystone.db
```

Step 3 - setting environment variables

To avoid providing credentials every time when you run an OpenStack command, create a file containing the environment variables that can be loaded at any time. To do that, execute the following command:

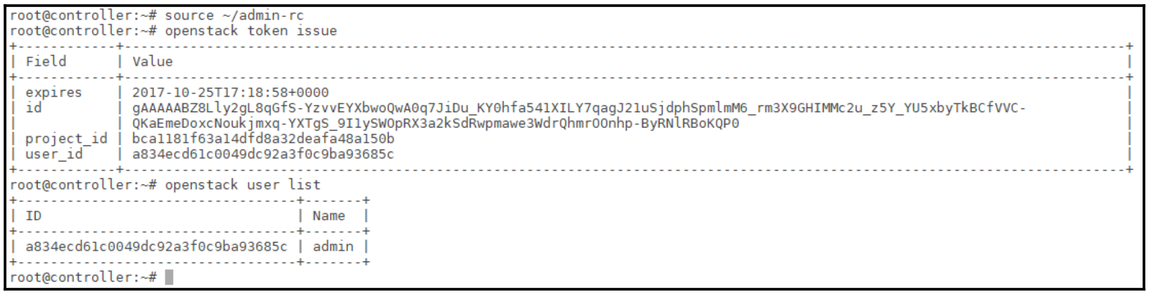
```
cat >> ~/admin-rc <<EOF
export OS_USERNAME=admin
export OS_PASSWORD=bootcamp
export OS_PROJECT_NAME=admin
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_DOMAIN_NAME=Default
export OS_AUTH_URL=http://controller.hellovinoth.com:35357/v3
export OS_IDENTITY_API_VERSION=3
EOF
```

Replace `bootcamp` with the password used in the keystone-manage bootstrap command in the *Step 1 - Installing and configure components* section.

Use the following source command to load the environment variables from the file. Then verify the configuration by executing the following commands:

```
source ~/admin-rc
openstack token issue
openstack user list
```

In response to the preceding command, you will see the output similar to the one here:



```
root@controller:~# source ~/admin-rc
root@controller:~# openstack token issue
+------------+-----------------------------------------------------------------------------------------------------------+
| Field      | Value                                                                                                     |
+------------+-----------------------------------------------------------------------------------------------------------+
| expires    | 2017-10-25T17:18:58+0000                                                                                  |
| id         | gAAAAABZ8Lly2gL8qGfS-YzvvEYXbwoQwA0q7JiDu_KY0hfa541XILY7qagJ21uSjdphSpmlmM6_rm3X9GHIMMc2u_z5Y_YU5xbyTkBCfVVC- |
|            | QKaEmeDoxcNoukjmxq-YXTgS_9I1ySWOpRX3a2kSdRwpmawe3WdrQhmrOOnhp-ByRNlRBoKQP0                                 |
| project_id | bca1181f63a14dfd8a32deafa48a150b                                                                          |
| user_id    | a834ecd61c0049dc92a3f0c9ba93685c                                                                          |
+------------+-----------------------------------------------------------------------------------------------------------+
root@controller:~# openstack user list
+----------------------------------+-------+
| ID                               | Name  |
+----------------------------------+-------+
| a834ecd61c0049dc92a3f0c9ba93685c | admin |
+----------------------------------+-------+
root@controller:~#
```

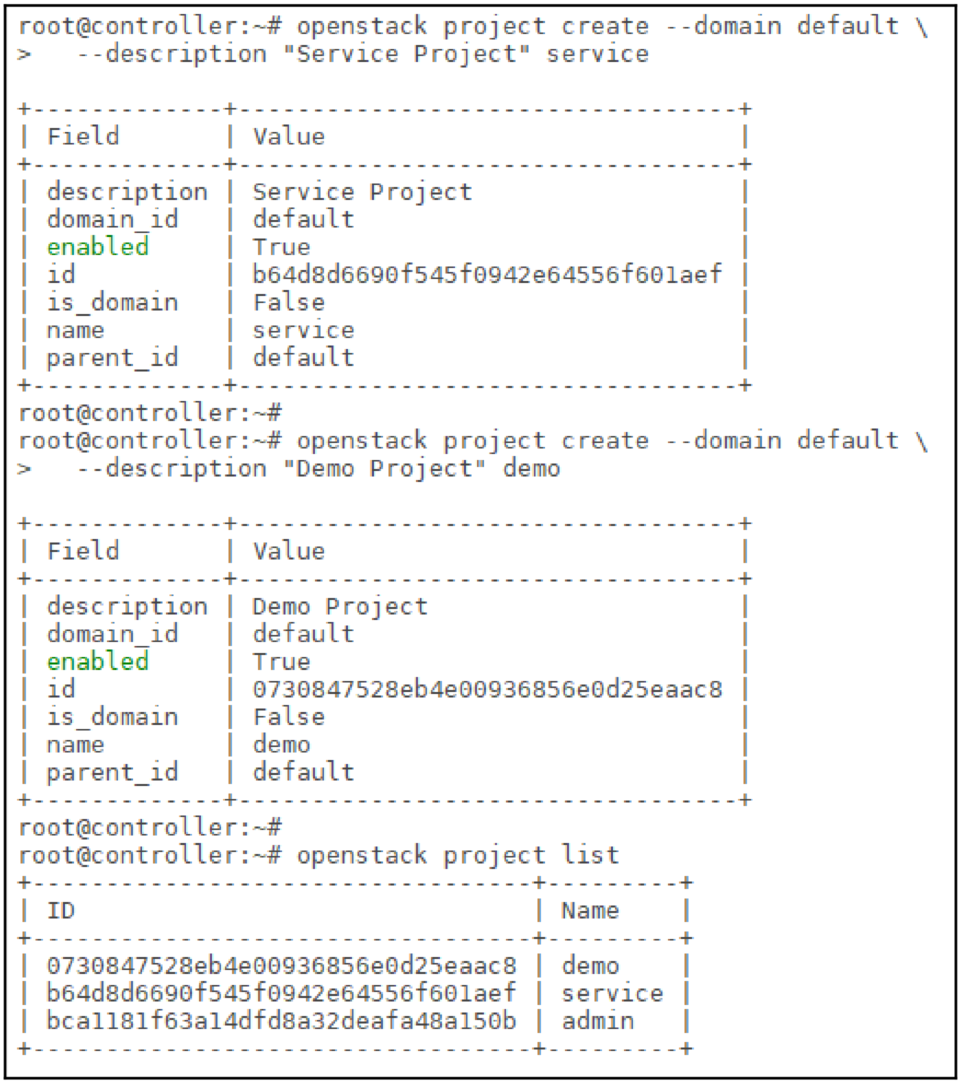
Step 4 - defining projects in KeyStone

Once the installation of KeyStone is complete, it is necessary to set up the domain, project, users, roles, and endpoints that will be used by various OpenStack services.

In KeyStone, a project represents a logical group of users to which resources are assigned. Resources are assigned to projects and not directly to users. An admin project for the administrative user is already created as part of the keystone-manage bootstrap process. Now, let us create a `Service Project` that contains a unique user for each OpenStack service and a project for an unprivileged user, namely `Demo Project`, to manage the regular task by following the instructions:

```
$ openstack project create --domain default \
  --description "Service Project" service
$ openstack project create --domain default \
  --description "Demo Project" demo
```

In response to the preceding commands, you will see the output similar to the one here:



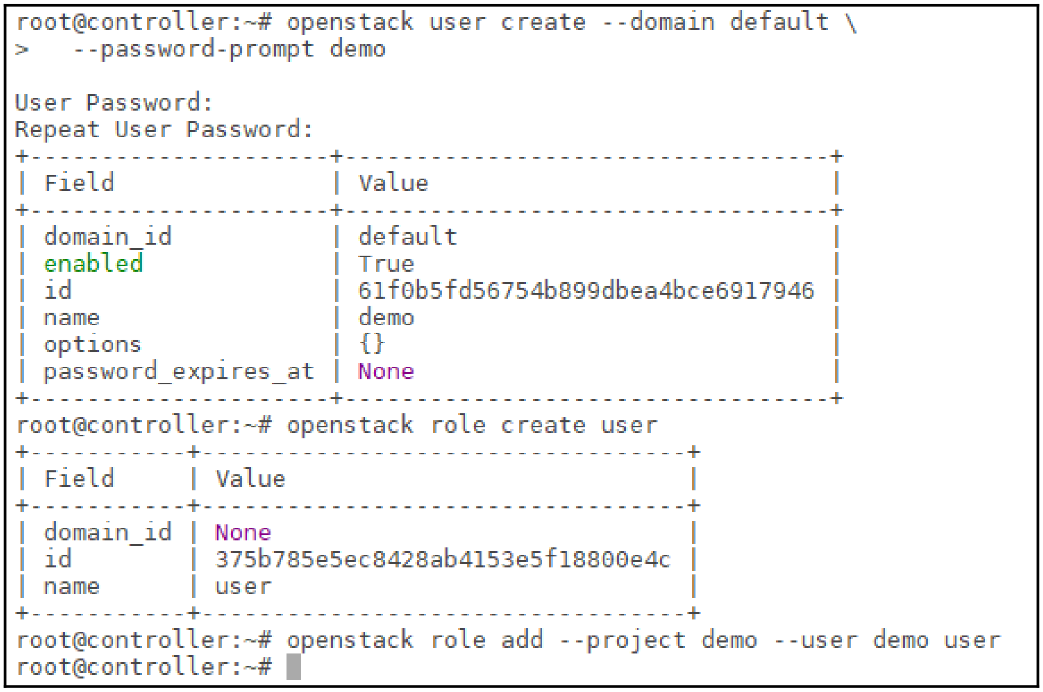
```
root@controller:~# openstack project create --domain default \
>   --description "Service Project" service

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Service Project                  |
| domain_id   | default                          |
| enabled     | True                             |
| id          | b64d8d6690f545f0942e64556f601aef |
| is_domain   | False                            |
| name        | service                          |
| parent_id   | default                          |
+-------------+----------------------------------+
root@controller:~#
root@controller:~# openstack project create --domain default \
>   --description "Demo Project" demo

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Demo Project                     |
| domain_id   | default                          |
| enabled     | True                             |
| id          | 0730847528eb4e00936856e0d25eaac8 |
| is_domain   | False                            |
| name        | demo                             |
| parent_id   | default                          |
+-------------+----------------------------------+
root@controller:~#
root@controller:~# openstack project list
+----------------------------------+---------+
| ID                               | Name    |
+----------------------------------+---------+
| 0730847528eb4e00936856e0d25eaac8 | demo    |
| b64d8d6690f545f0942e64556f601aef | service |
| bca1181f63a14dfd8a32deafa48a150b | admin   |
+----------------------------------+---------+
```

Step 5 - defining users and map role in KeyStone

Additional projects can be created later for other users of the cloud. Next, we need to create both an admin and non-admin user to access the cloud. As admin user is already created as part of the keystone-manage bootstrap process; now, we need to create a non-admin user called `demo` user and map the non-privileged role called `user` role for the demo user, as follows:

```
openstack user create --domain default \
  --password-prompt demo
openstack role create user
openstack role add --project demo --user demo user
```

In response to the preceding command, you will see the output like the one here:



```
root@controller:~# openstack user create --domain default \
>   --password-prompt demo

User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 61f0b5fd56754b899dbea4bce6917946 |
| name                | demo                             |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
root@controller:~# openstack role create user
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | None                             |
| id        | 375b785e5ec8428ab4153e5f18800e4c |
| name      | user                             |
+-----------+----------------------------------+
root@controller:~# openstack role add --project demo --user demo user
root@controller:~#
```

Note that the preceding command for assigning the role `user` to the user `demo` has no output in response.

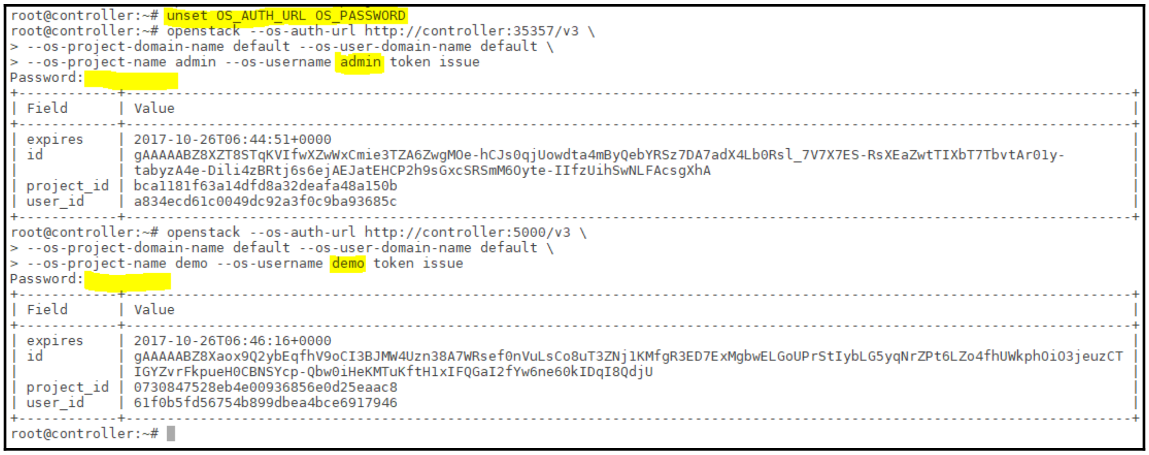
Any roles that are created should be predefined in the `policy.json` files of the corresponding OpenStack services. By default, the policy files use the admin role to allow access to the services. For more information on role management in KeyStone, please refer to the following URL:
`https://docs.openstack.org/oslo.policy/latest/admin/policy-json-file.html`.

Step 6 - verifying KeyStone operation

To verify that the identity service was installed and configured correctly, use the unset command to unset the `OS_AUTH_URL` and `OS_PASSWORD` environment variables. Then, use username-based authentication to request an authentication token using the admin user and the respective password, as follows:

```
unset OS_AUTH_URL OS_PASSWORD
openstack --os-auth-url http://controller:35357/v3 \
--os-project-domain-name default --os-user-domain-name default \
--os-project-name admin --os-username admin token issue
openstack --os-auth-url http://controller:5000/v3 \
--os-project-domain-name default --os-user-domain-name default \
--os-project-name demo --os-username demo token issue
```

In response to the previous command, you will see the output like the one here:



```
root@controller:~# unset OS_AUTH_URL OS_PASSWORD
root@controller:~# openstack --os-auth-url http://controller:35357/v3 \
> --os-project-domain-name default --os-user-domain-name default \
> --os-project-name admin --os-username admin token issue
Password:
+------------+------------------------------------------------------------------------------------------------------------------------------------------+
| Field      | Value                                                                                                                                    |
+------------+------------------------------------------------------------------------------------------------------------------------------------------+
| expires    | 2017-10-26T06:44:51+0000                                                                                                                 |
| id         | gAAAAABZ8XZT8STqKVIfwXZwWxCmie3TZA6ZwgMOe-hCJs0qjUowdta4mByQebYRSz7DA7adX4Lb0Rsl_7V7X7ES-RsXEaZwtTIXbT7TbvtAr01y-                           |
|            | tabyzA4e-Dili4zBRtj6s6ejAEJatEHCP2h9sGxcSRSmM6Oyte-IIfzUihSwNLFAcsgXhA                                                                   |
| project_id | bca1181f63a14dfd8a32deafa48a150b                                                                                                         |
| user_id    | a834ecd61c0049dc92a3f0c9ba93685c                                                                                                         |
+------------+------------------------------------------------------------------------------------------------------------------------------------------+
root@controller:~# openstack --os-auth-url http://controller:5000/v3 \
> --os-project-domain-name default --os-user-domain-name default \
> --os-project-name demo --os-username demo token issue
Password:
+------------+------------------------------------------------------------------------------------------------------------------------------------------+
| Field      | Value                                                                                                                                    |
+------------+------------------------------------------------------------------------------------------------------------------------------------------+
| expires    | 2017-10-26T06:46:16+0000                                                                                                                 |
| id         | gAAAAABZ8Xaox9Q2ybEqfhV9oCI3BJMW4Uzn38A7WRsef0nVuLsCo8uT3ZNj1KMfgR3ED7ExMgbwELGoUPrStIybLG5yqNrZPt6LZo4fhUWkph0i03jeuzCT |
|            | IGYZvrFkpueH0CBNSYcp-Qbw0iHeKMTuKftH1xIFQGaI2fYw6ne60kIDqI8QdjU                                                                          |
| project_id | 0730847528eb4e00936856e0d25eaac8                                                                                                         |
| user_id    | 61f0b5fd56754b899dbea4bce6917946                                                                                                         |
+------------+------------------------------------------------------------------------------------------------------------------------------------------+
root@controller:~#
```

While executing the token issue command, you will be prompted to enter the password for the admin and demo users. You should enter the password of the respective users that we created earlier.

Step 7 - creating OpenRC environment file

To increase the efficiency of client operations, OpenStack provides client environment scripts, also known as OpenRC files. With the help of OpenRC environment file, we can avoid providing the credentials every time you run an OpenStack command.

You may refer to the above session *Step 3 - setting environment variables*, in which we have created an OpenRC file for the admin user. Now, we should create another OpernRC file for the demo account by following the following command:

```
cat >> ~/demo-openrc <<EOF
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=demo
export OS_USERNAME=demo
export OS_PASSWORD=bootcamp
export OS_AUTH_URL=http://controller.hellovinoth.com:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2
EOF
```

Now, verify the environment file for the admin and demo account by loading the respective OpenRC file to populate the environment variables and execute the token issue command one by one as follows:

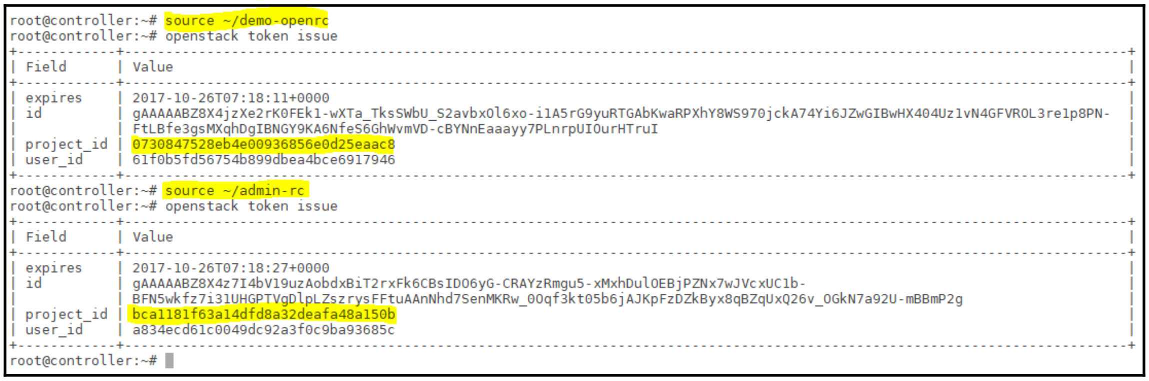
- For demo account:

  ```
  source ~/demo-openrc
  openstack token issue
  ```

- For admin account:

  ```
  source ~/admin-rc
  openstack token issue
  ```

In response to the preceding command, you will see the output like the one here:



```
root@controller:~# source ~/demo-openrc
root@controller:~# openstack token issue
+------------+--------------------------------------------------------------------------------------------------------------------------+
| Field      | Value                                                                                                                    |
+------------+--------------------------------------------------------------------------------------------------------------------------+
| expires    | 2017-10-26T07:18:11+0000                                                                                                 |
| id         | gAAAAABZ8X4jzXe2rK0FEk1-wXTa_TksSWbU_S2avbx0l6xo-i1A5rG9yuRTGAbKwaRPXhY8WS970jckA74Yi6JZwGIBwHX404Uz1vN4GFVR0L3re1p8PN-  |
|            | FtLBfe3gsMXqhDgIBNGY9KA6NfeS6GhWvmVD-cBYNnEaaayy7PLnrpUI0urHTruI                                                         |
| project_id | 0730847528eb4e00936856e0d25eaac8                                                                                         |
| user_id    | 61f0b5fd56754b899dbea4bce6917946                                                                                         |
+------------+--------------------------------------------------------------------------------------------------------------------------+
root@controller:~# source ~/admin-rc
root@controller:~# openstack token issue
+------------+--------------------------------------------------------------------------------------------------------------------------+
| Field      | Value                                                                                                                    |
+------------+--------------------------------------------------------------------------------------------------------------------------+
| expires    | 2017-10-26T07:18:27+0000                                                                                                 |
| id         | gAAAAABZ8X4z7I4bV19uzAobdxBiT2rxFk6CBsID06yG-CRAYzRmgu5-xMxhDul0EBjPZNx7wJVcxUC1b-                                       |
|            | BFN5wkfz7i31UHGPTVgDlpLZszrysFFtuAAnNhd7SenMKRw_0Oqf3kt05b6jAJKpFzDZkByx8qBZqUxQ26v_0GkN7a92U-mBBmP2g                    |
| project_id | bca1181f63a14dfd8a32deafa48a150b                                                                                         |
| user_id    | a834ecd61c0049dc92a3f0c9ba93685c                                                                                         |
+------------+--------------------------------------------------------------------------------------------------------------------------+
root@controller:~#
```

Configuring the image service (Glance)

Glance is the image service for OpenStack. It is responsible for storing images and snapshots of instances and for providing images for computing nodes when instances are created.

Before we start the actual installation of the image service, load the admin OpenRC file to gain access to admin-only CLI commands:

```
source ~/admin-rc
```

Step 1 - defining the Glance service and API endpoints in KeyStone

Each OpenStack service should have the dedicated user account, service, and endpoints defined in the identity service.

Follow the instructions to create a dedicated user for the Glance service called `glance` user followed by mapping the `admin` role to the `glance` user on the `service` project:

```
openstack user create --domain default --password-prompt glance
openstack role add --project service --user glance admin
```

Then, we need to create a service entity for the Glance service, followed by API endpoints for the image service using the following commands:

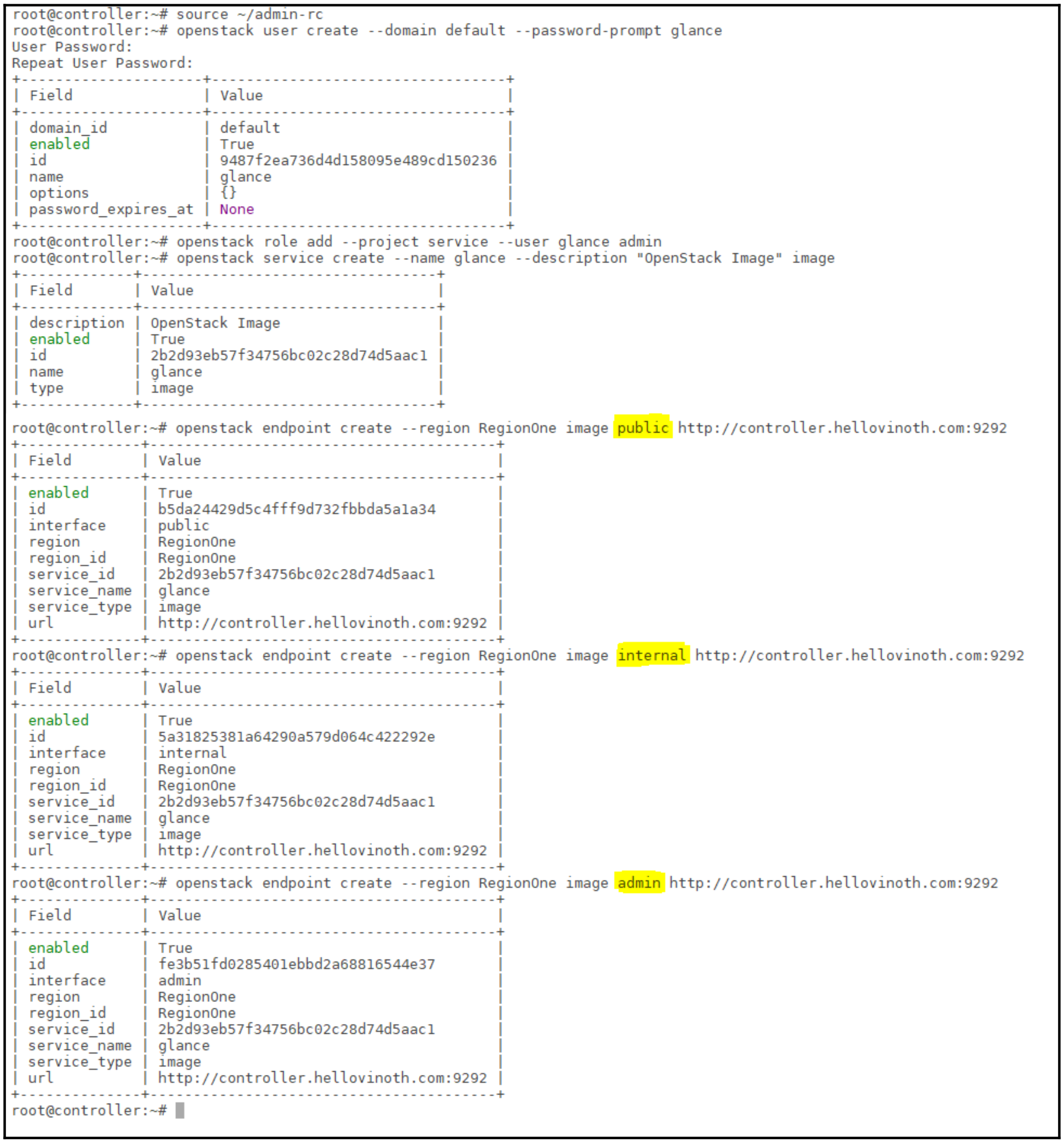
```
openstack service create --name glance \
  --description "OpenStack Image" image
openstack endpoint create --region RegionOne \
  image public http://controller.hellovinoth.com:9292
openstack endpoint create --region RegionOne \
  image internal http://controller.hellovinoth.com:9292
openstack endpoint create --region RegionOne \
  image admin http://controller.hellovinoth.com:9292
```

From the preceding command, you will have noticed that we have created three different endpoints for the same image service. The purpose of the three different endpoints are to server public, internal, and admin API requests.

Here:

- Public URL handles the API communication for the non-privileged task
- Internal URL handles the API communication between the OpenStack components
- Admin URL handles the API communication for the admin only functionality. So, APIs available in admin URL are not available in public/internal URLs

In response to the preceding command, you will see the output similar to the one in the following figure:



```
root@controller:~# source ~/admin-rc
root@controller:~# openstack user create --domain default --password-prompt glance
User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 9487f2ea736d4d158095e489cd150236 |
| name                | glance                           |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
root@controller:~# openstack role add --project service --user glance admin
root@controller:~# openstack service create --name glance --description "OpenStack Image" image
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Image                  |
| enabled     | True                             |
| id          | 2b2d93eb57f34756bc02c28d74d5aac1 |
| name        | glance                           |
| type        | image                            |
+-------------+----------------------------------+
root@controller:~# openstack endpoint create --region RegionOne image public http://controller.hellovinoth.com:9292
+--------------+--------------------------------------+
| Field        | Value                                |
+--------------+--------------------------------------+
| enabled      | True                                 |
| id           | b5da24429d5c4fff9d732fbbda5a1a34     |
| interface    | public                               |
| region       | RegionOne                            |
| region_id    | RegionOne                            |
| service_id   | 2b2d93eb57f34756bc02c28d74d5aac1     |
| service_name | glance                               |
| service_type | image                                |
| url          | http://controller.hellovinoth.com:9292 |
+--------------+--------------------------------------+
root@controller:~# openstack endpoint create --region RegionOne image internal http://controller.hellovinoth.com:9292
+--------------+--------------------------------------+
| Field        | Value                                |
+--------------+--------------------------------------+
| enabled      | True                                 |
| id           | 5a31825381a64290a579d064c422292e     |
| interface    | internal                             |
| region       | RegionOne                            |
| region_id    | RegionOne                            |
| service_id   | 2b2d93eb57f34756bc02c28d74d5aac1     |
| service_name | glance                               |
| service_type | image                                |
| url          | http://controller.hellovinoth.com:9292 |
+--------------+--------------------------------------+
root@controller:~# openstack endpoint create --region RegionOne image admin http://controller.hellovinoth.com:9292
+--------------+--------------------------------------+
| Field        | Value                                |
+--------------+--------------------------------------+
| enabled      | True                                 |
| id           | fe3b51fd0285401ebbd2a68816544e37     |
| interface    | admin                                |
| region       | RegionOne                            |
| region_id    | RegionOne                            |
| service_id   | 2b2d93eb57f34756bc02c28d74d5aac1     |
| service_name | glance                               |
| service_type | image                                |
| url          | http://controller.hellovinoth.com:9292 |
+--------------+--------------------------------------+
root@controller:~#
```

Step 2 - installing and configuring the Glance components

To install Glance binaries, run the following command on the controller node:

```
apt install glance wget -y
```

At this point, you have Glance packages installed in your controller. Now, follow the below instructions to configure the Glance components.

Use the `crudini` command to set the SQL connection string in the Glance configuration files:

```
crudini --set /etc/glance/glance-api.conf database connection
mysql+pymysql://glance:bootcamp@controller/glance
crudini --set /etc/glance/glance-registry.conf database connection
mysql+pymysql://glance:bootcamp@controller/glance
```

Let's set the attributes in the `/etc/glance/glance-api.conf` configuration files by copy and paste the following commands in the terminal:

```
crudini --set /etc/glance/glance-api.conf keystone_authtoken auth_uri
http://controller:5000
crudini --set /etc/glance/glance-api.conf keystone_authtoken auth_url
http://controller:35357
crudini --set /etc/glance/glance-api.conf keystone_authtoken
memcached_servers controller:11211
crudini --set /etc/glance/glance-api.conf keystone_authtoken
auth_type password
crudini --set /etc/glance/glance-api.conf keystone_authtoken
project_domain_name default
crudini --set /etc/glance/glance-api.conf keystone_authtoken
user_domain_name default
crudini --set /etc/glance/glance-api.conf keystone_authtoken
project_name service
crudini --set /etc/glance/glance-api.conf keystone_authtoken
username glance
crudini --set /etc/glance/glance-api.conf keystone_authtoken
password bootcamp
crudini --set /etc/glance/glance-api.conf paste_deploy flavor
keystone
crudini --set /etc/glance/glance-api.conf glance_store stores
file,http
crudini --set /etc/glance/glance-api.conf glance_store default_store
file
crudini --set /etc/glance/glance-api.conf glance_store
filesystem_store_datadir /var/lib/glance/images/
```

The `crudini` command-line tool in the preceding command will take care of replacing the attribute at the appropriate place in the specified configuration file.

Now, set the attributes in the `/etc/glance/glance-registry.conf` configuration files by copy and pasting the following commands in to the terminal:

```
crudini --set /etc/glance/glance-registry.conf keystone_authtoken auth_uri
http://controller:5000
crudini --set /etc/glance/glance-registry.conf keystone_authtoken auth_url
http://controller:35357
crudini --set /etc/glance/glance-registry.conf keystone_authtoken
memcached_servers controller:11211
crudini --set /etc/glance/glance-registry.conf keystone_authtoken
auth_type password
crudini --set /etc/glance/glance-registry.conf keystone_authtoken
project_domain_name default
crudini --set /etc/glance/glance-registry.conf keystone_authtoken
user_domain_name default
crudini --set /etc/glance/glance-registry.conf keystone_authtoken
project_name service
crudini --set /etc/glance/glance-registry.conf keystone_authtoken
username glance
crudini --set /etc/glance/glance-registry.conf keystone_authtoken
password bootcamp
crudini --set /etc/glance/glance-registry.conf paste_deploy flavor
keystone
```

Now, populate the image service database and then finalize the installation:

```
su -s /bin/sh -c "glance-manage db_sync" glance
service glance-registry restart
service glance-api restart
```

Step 3 - verifying the Glance operation

To verify that Glance was installed and configured correctly, download a test image from the internet, and verify that it can be uploaded to the image server.

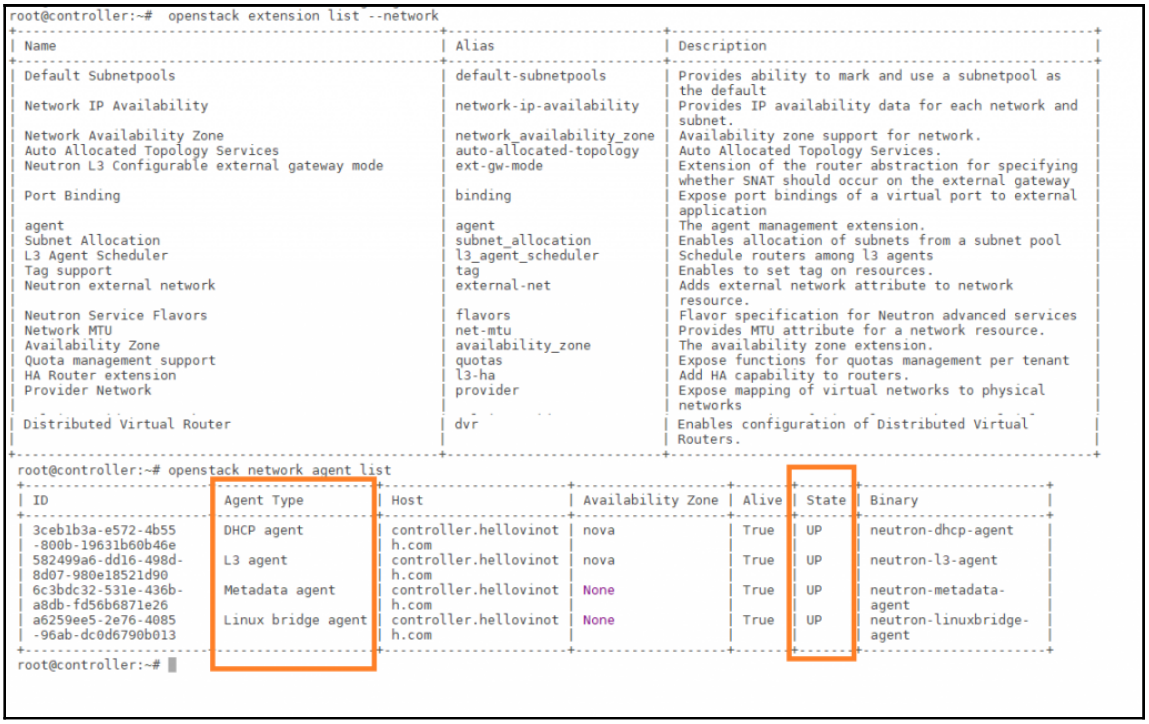
Follow the instructions to download the minimal cirros image from the internet and upload the image to the Glance service:

```
source ~/admin-rc
wget http://download.cirros-cloud.net/0.3.5/cirros-0.3.5-x86_64-disk.img
openstack image create "cirros" \
  --file cirros-0.3.5-x86_64-disk.img \
  --disk-format qcow2 --container-format bare \
  --public
```

Now, confirm the successful upload of the cirros image and validate the attributes using the following command:

```
openstack image list
```

You may refer to the following figure as the reference for the output of preceding commands:



```
root@controller:~#  openstack extension list --network
+-------------------------------------------------+-----------------------------+------------------------------------------------------+
| Name                                            | Alias                       | Description                                          |
+-------------------------------------------------+-----------------------------+------------------------------------------------------+
| Default Subnetpools                             | default-subnetpools         | Provides ability to mark and use a subnetpool as     |
|                                                 |                             | the default                                          |
| Network IP Availability                         | network-ip-availability     | Provides IP availability data for each network and   |
|                                                 |                             | subnet.                                              |
| Network Availability Zone                       | network_availability_zone   | Availability zone support for network.               |
| Auto Allocated Topology Services                | auto-allocated-topology     | Auto Allocated Topology Services.                    |
| Neutron L3 Configurable external gateway mode   | ext-gw-mode                 | Extension of the router abstraction for specifying   |
|                                                 |                             | whether SNAT should occur on the external gateway    |
| Port Binding                                    | binding                     | Expose port bindings of a virtual port to external   |
|                                                 |                             | application                                          |
| agent                                           | agent                       | The agent management extension.                      |
| Subnet Allocation                               | subnet_allocation           | Enables allocation of subnets from a subnet pool     |
| L3 Agent Scheduler                              | l3_agent_scheduler          | Schedule routers among l3 agents                     |
| Tag support                                     | tag                         | Enables to set tag on resources.                     |
| Neutron external network                        | external-net                | Adds external network attribute to network           |
|                                                 |                             | resource.                                            |
| Neutron Service Flavors                         | flavors                     | Flavor specification for Neutron advanced services   |
| Network MTU                                     | net-mtu                     | Provides MTU attribute for a network resource.       |
| Availability Zone                               | availability_zone           | The availability zone extension.                     |
| Quota management support                        | quotas                      | Expose functions for quotas management per tenant    |
| HA Router extension                             | l3-ha                       | Add HA capability to routers.                        |
| Provider Network                                | provider                    | Expose mapping of virtual networks to physical       |
|                                                 |                             | networks                                             |
| Distributed Virtual Router                      | dvr                         | Enables configuration of Distributed Virtual         |
|                                                 |                             | Routers.                                             |
+-------------------------------------------------+-----------------------------+------------------------------------------------------+
root@controller:~# openstack network agent list
+----------------------+--------------------+----------------------+-------------------+-------+-------+----------------------+
| ID                   | Agent Type         | Host                 | Availability Zone | Alive | State | Binary               |
+----------------------+--------------------+----------------------+-------------------+-------+-------+----------------------+
| 3ceb1b3a-e572-4b55   | DHCP agent         | controller.hellovinot | nova             | True  | UP    | neutron-dhcp-agent   |
| -800b-19631b60b46e   |                    | h.com                |                   |       |       |                      |
| 582499a6-dd16-498d-  | L3 agent           | controller.hellovinot | nova             | True  | UP    | neutron-l3-agent     |
| 8d07-980e18521d90    |                    | h.com                |                   |       |       |                      |
| 6c3bdc32-531e-436b-  | Metadata agent     | controller.hellovinot | None             | True  | UP    | neutron-metadata-    |
| a8db-fd56b6871e26    |                    | h.com                |                   |       |       | agent                |
| a6259ee5-2e76-4085   | Linux bridge agent | controller.hellovinot | None             | True  | UP    | neutron-linuxbridge- |
| -96ab-dc0d6790b013   |                    | h.com                |                   |       |       | agent                |
+----------------------+--------------------+----------------------+-------------------+-------+-------+----------------------+
root@controller:~# ▌
```

Configuring the Compute service (Nova)

OpenStack compute is a collection of services that enable cloud operators to launch virtual machine instances. Most services run on the controller node, except for the OpenStack `nova-compute` service, which runs on the compute nodes and is responsible for launching the virtual machine instances. As I mentioned earlier, in our case, for all-in-one node setup, we will be running the `nova-compute` service on the same node.

Before we start the actual installation of Nova service, load the admin OpenRC file to gain access to admin-only CLI commands:

```
source ~/admin-rc
```

Step 1 - defining the Nova service and API endpoints in KeyStone

Follow the instructions to create a dedicated user for Nova service called `nova` user, followed by mapping the `admin` role to the `nova` user on the `service` project:

```
openstack user create --domain default --password-prompt nova
openstack role add --project service --user nova admin
```

Then, we need to create a service entity for the Nova service, followed by API endpoints for the Nova service, using the following commands:

```
openstack service create --name nova --description "OpenStack
Compute" compute
openstack endpoint create --region RegionOne compute public
http://controller.hellovinoth.com:8774/v2.1
openstack endpoint create --region RegionOne compute internal
http://controller.hellovinoth.com:8774/v2.1
openstack endpoint create --region RegionOne compute admin
http://controller.hellovinoth.com:8774/v2.1
```

Starting from the Newton release of OpenStack, the Nova service has introduced a new subcomponent called Nova placement service with a separate REST API stack and data model used to track the resource provider such as compute node, an IP allocation pool, or a shared storage pool.

For example, a virtual machine provisioned on the compute node will be recorded as a consumer of RAM and CPU resources on a compute node resource provider. Similarly, IP addresses created from an external IP pool is a consumer of the IP pool resource provider and the disk created from an externally shared storage pool will be the consumer of the storage resource provider.

As the placement service has separate REST API stack, we need to create a dedicated user for the placement service called `placement` user and followed by assigning the `admin` role to the `placement` user for the project `service`.

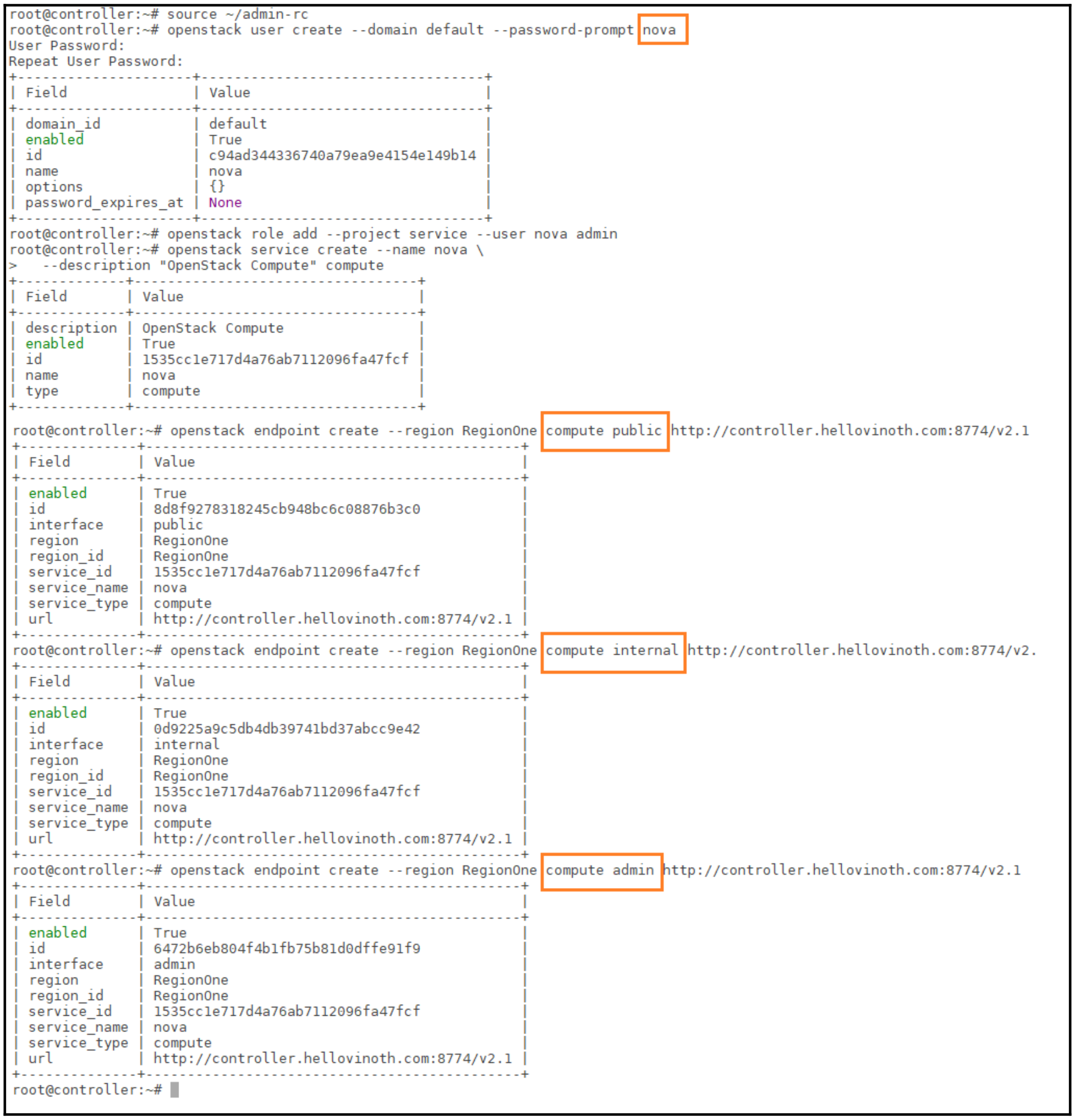
Execute the following commands to do the same:

```
openstack user create --domain default --password-prompt placement
openstack role add --project service --user placement admin
```

We should also create a service entity for the placement service, followed by API endpoints for the placement service, using the following commands:

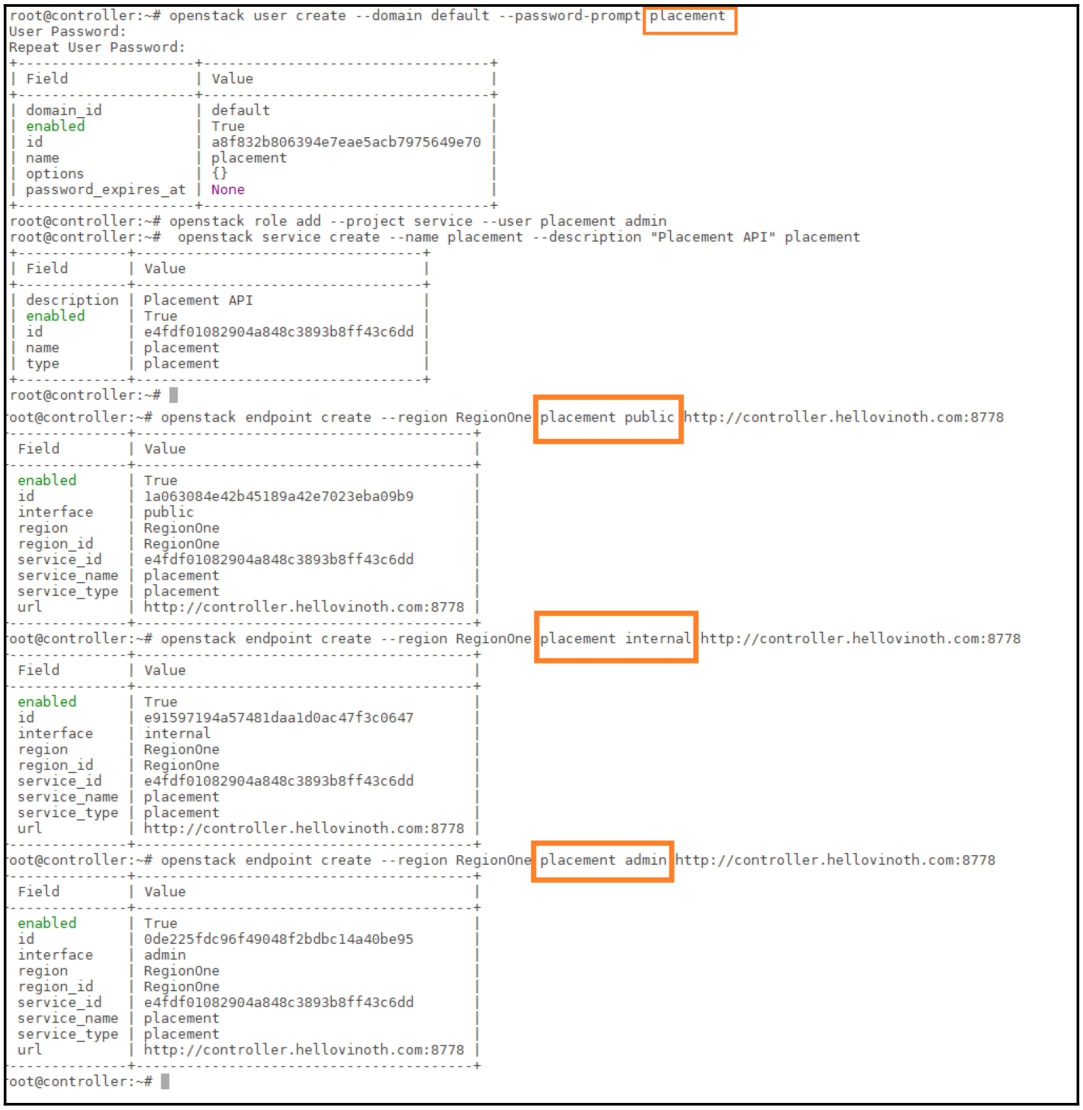
```
openstack service create --name placement --description "Placement API"
placement
openstack endpoint create --region RegionOne placement public
http://controller.hellovinoth.com:8778
openstack endpoint create --region RegionOne placement internal
http://controller.hellovinoth.com:8778
openstack endpoint create --region RegionOne placement admin
http://controller.hellovinoth.com:8778
```

You may refer to the following figure as the reference for the output of previous commands:



```
root@controller:~# source ~/admin-rc
root@controller:~# openstack user create --domain default --password-prompt nova
User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | c94ad344336740a79ea9e4154e149b14 |
| name                | nova                             |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
root@controller:~# openstack role add --project service --user nova admin
root@controller:~# openstack service create --name nova \
>   --description "OpenStack Compute" compute
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Compute                |
| enabled     | True                             |
| id          | 1535cc1e717d4a76ab7112096fa47fcf |
| name        | nova                             |
| type        | compute                          |
+-------------+----------------------------------+
root@controller:~# openstack endpoint create --region RegionOne compute public http://controller.hellovinoth.com:8774/v2.1
+--------------+---------------------------------------------+
| Field        | Value                                       |
+--------------+---------------------------------------------+
| enabled      | True                                        |
| id           | 8d8f9278318245cb948bc6c08876b3c0            |
| interface    | public                                      |
| region       | RegionOne                                   |
| region_id    | RegionOne                                   |
| service_id   | 1535cc1e717d4a76ab7112096fa47fcf            |
| service_name | nova                                        |
| service_type | compute                                     |
| url          | http://controller.hellovinoth.com:8774/v2.1 |
+--------------+---------------------------------------------+
root@controller:~# openstack endpoint create --region RegionOne compute internal http://controller.hellovinoth.com:8774/v2.
+--------------+---------------------------------------------+
| Field        | Value                                       |
+--------------+---------------------------------------------+
| enabled      | True                                        |
| id           | 0d9225a9c5db4db39741bd37abcc9e42            |
| interface    | internal                                    |
| region       | RegionOne                                   |
| region_id    | RegionOne                                   |
| service_id   | 1535cc1e717d4a76ab7112096fa47fcf            |
| service_name | nova                                        |
| service_type | compute                                     |
| url          | http://controller.hellovinoth.com:8774/v2.1 |
+--------------+---------------------------------------------+
root@controller:~# openstack endpoint create --region RegionOne compute admin http://controller.hellovinoth.com:8774/v2.1
+--------------+---------------------------------------------+
| Field        | Value                                       |
+--------------+---------------------------------------------+
| enabled      | True                                        |
| id           | 6472b6eb804f4b1fb75b81d0dffe91f9            |
| interface    | admin                                       |
| region       | RegionOne                                   |
| region_id    | RegionOne                                   |
| service_id   | 1535cc1e717d4a76ab7112096fa47fcf            |
| service_name | nova                                        |
| service_type | compute                                     |
| url          | http://controller.hellovinoth.com:8774/v2.1 |
+--------------+---------------------------------------------+
root@controller:~#
```

The following output reference is with respect to the placement service stack:



```
root@controller:~# openstack user create --domain default --password-prompt placement
User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | a8f832b806394e7eae5acb7975649e70 |
| name                | placement                        |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
root@controller:~# openstack role add --project service --user placement admin
root@controller:~#  openstack service create --name placement --description "Placement API" placement
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Placement API                    |
| enabled     | True                             |
| id          | e4fdf01082904a848c3893b8ff43c6dd |
| name        | placement                        |
| type        | placement                        |
+-------------+----------------------------------+
root@controller:~# 
oot@controller:~# openstack endpoint create --region RegionOne placement public http://controller.hellovinoth.com:8778
--------------+----------------------------------+
 Field        | Value                            |
--------------+----------------------------------+
 enabled      | True                             |
 id           | 1a063084e42b45189a42e7023eba09b9 |
 interface    | public                           |
 region       | RegionOne                        |
 region_id    | RegionOne                        |
 service_id   | e4fdf01082904a848c3893b8ff43c6dd |
 service_name | placement                        |
 service_type | placement                        |
 url          | http://controller.hellovinoth.com:8778 |
--------------+----------------------------------+
oot@controller:~# openstack endpoint create --region RegionOne placement internal http://controller.hellovinoth.com:8778
--------------+----------------------------------+
 Field        | Value                            |
--------------+----------------------------------+
 enabled      | True                             |
 id           | e91597194a57481daa1d0ac47f3c0647 |
 interface    | internal                         |
 region       | RegionOne                        |
 region_id    | RegionOne                        |
 service_id   | e4fdf01082904a848c3893b8ff43c6dd |
 service_name | placement                        |
 service_type | placement                        |
 url          | http://controller.hellovinoth.com:8778 |
--------------+----------------------------------+
oot@controller:~# openstack endpoint create --region RegionOne placement admin http://controller.hellovinoth.com:8778
--------------+----------------------------------+
 Field        | Value                            |
--------------+----------------------------------+
 enabled      | True                             |
 id           | 0de225fdc96f49048f2bdbc14a40be95 |
 interface    | admin                            |
 region       | RegionOne                        |
 region_id    | RegionOne                        |
 service_id   | e4fdf01082904a848c3893b8ff43c6dd |
 service_name | placement                        |
 service_type | placement                        |
 url          | http://controller.hellovinoth.com:8778 |
--------------+----------------------------------+
oot@controller:~# 
```

The IDs in the preceding figures are randomly generated. So, the IDs display in your output may differ from the ones shown here.

Step 2 - installing and configuring the Nova components

To install Nova binaries, run the following command on the controller node:

```
apt install nova-api nova-conductor nova-consoleauth \
  nova-novncproxy nova-scheduler nova-placement-api -y
```

At this point, you have Nova packages installed in your controller. Now, simply copy and paste the below `crudini` commands into the terminal to configure the Nova components:

```
crudini --set /etc/nova/nova.conf api_database connection
mysql+pymysql://nova:bootcamp@controller/nova_api

crudini --set /etc/nova/nova.conf database connection
mysql+pymysql://nova:bootcamp@controller/nova

crudini --set /etc/nova/nova.conf DEFAULT transport_url
rabbit://openstack:bootcamp@controller

crudini --set /etc/nova/nova.conf api auth_strategy keystone

crudini --set /etc/nova/nova.conf keystone_authtoken
auth_uri http://controller:5000

crudini --set /etc/nova/nova.conf keystone_authtoken
auth_url http://controller:35357

crudini --set /etc/nova/nova.conf keystone_authtoken
memcached_servers controller:11211

crudini --set /etc/nova/nova.conf keystone_authtoken
auth_type password

crudini --set /etc/nova/nova.conf keystone_authtoken
project_domain_name default

crudini --set /etc/nova/nova.conf keystone_authtoken
user_domain_name default

crudini --set /etc/nova/nova.conf keystone_authtoken project_name
service
```

```
crudini --set /etc/nova/nova.conf keystone_authtoken username
nova

crudini --set /etc/nova/nova.conf keystone_authtoken password
bootcamp

crudini --set /etc/nova/nova.conf DEFAULT use_neutron True

crudini --set /etc/nova/nova.conf DEFAULT firewall_driver
nova.virt.firewall.NoopFirewallDriver

crudini --set /etc/nova/nova.conf glance api_servers
http://controller:9292

crudini --set /etc/nova/nova.conf oslo_concurrency
lock_path /var/lib/nova/tmp

crudini --set /etc/nova/nova.conf placement os_region_name
RegionOne

crudini --set /etc/nova/nova.conf placement project_domain_name
Default

crudini --set /etc/nova/nova.conf placement project_name
service

crudini --set /etc/nova/nova.conf placement auth_type
password

crudini --set /etc/nova/nova.conf placement user_domain_name
Default

crudini --set /etc/nova/nova.conf placement auth_url
http://controller:35357/v3

crudini --set /etc/nova/nova.conf placement username
placement

crudini --set /etc/nova/nova.conf placement password
bootcamp

crudini --set /etc/nova/nova.conf scheduler
discover_hosts_in_cells_interval 300
```

Note that I have given `bootcamp` as the password for all of the authentication parameters. Please replace `bootcamp` with the password you chose for the respective authentication in the identity service.

We are yet to finalize the Nova configuration; copy and paste the following commands to configure the VNC settings in Nova service.

CAUTION! Please replace the given IP address with the management IP address of your environment.

```
crudini --set /etc/nova/nova.conf DEFAULT my_ip
192.168.1.7
crudini --set /etc/nova/nova.conf vnc enabled true
crudini --set /etc/nova/nova.conf vnc vncserver_listen
192.168.1.7
crudini --set /etc/nova/nova.conf vnc vncserver_proxyclient_address
192.168.1.7
```

Now, populate the Nova service database and then finalize the installation:

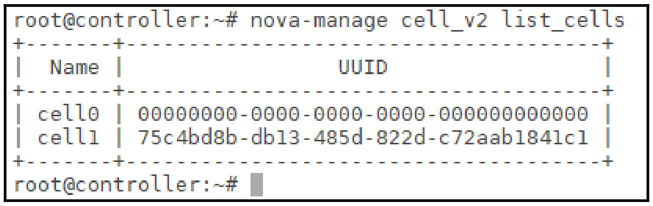
```
su -s /bin/sh -c "nova-manage api_db sync" nova
su -s /bin/sh -c "nova-manage cell_v2 map_cell0"
nova
su -s /bin/sh -c "nova-manage cell_v2 create_cell --name=cell1
--verbose" nova
su -s /bin/sh -c "nova-manage db sync"
nova
```

At this point, you may ignore any deprecation messages in this output.

To verify the successful registration of Nova `cell0` and `cell1`, run the following commands:

```
nova-manage cell_v2 list_cells
```

In response, you will receive an output similar to the one here:



```
root@controller:~# nova-manage cell_v2 list_cells
+-------+--------------------------------------+
|  Name |                 UUID                 |
+-------+--------------------------------------+
| cell0 | 00000000-0000-0000-0000-000000000000 |
| cell1 | 75c4bd8b-db13-485d-822d-c72aab1841c1 |
+-------+--------------------------------------+
root@controller:~#
```

Now, let's finalize the Nova installation by restarting all of the Nova components:

```
service nova-api restart
service nova-consoleauth restart
service nova-scheduler restart
service nova-conductor restart
service nova-novncproxy restart
```

Installing and configuring a compute node (nova-compute)

Once the Nova services have been configured on the controller node, another host must be configured as a compute node to receive requests from the controller node to host virtual machines. Separating the services by running dedicated compute nodes means that Nova (compute) can be scaled horizontally by adding additional compute nodes once all available resources have been utilized.

In case you would like to add an additional compute node to this existing OpenStack setup, you should repeat the steps from this section on the other compute node with the appropriate IP address modification in the configuration file.

As I mentioned earlier, for all-in-one node setup, we can merge the controller and compute node components in a single node. So, do follow the below instructions on the same node to install and configure the compute node components:

```
apt install nova-compute -y
```

At this point, you have `nova-compute` packages installed in your node. Now, simply copy and paste the following `crudini` commands in the terminal to configure the `nova-compute` components:

```
crudini --set /etc/nova/nova.conf DEFAULT transport_url
rabbit://openstack:bootcamp@controller
crudini --set /etc/nova/nova.conf api auth_strategy
```

```
keystone
crudini --set /etc/nova/nova.conf keystone_authtoken auth_uri
http://controller:5000
crudini --set /etc/nova/nova.conf keystone_authtoken auth_url
http://controller:35357
crudini --set /etc/nova/nova.conf keystone_authtoken memcached_servers
controller:11211
crudini --set /etc/nova/nova.conf keystone_authtoken auth_type
password
crudini --set /etc/nova/nova.conf keystone_authtoken
project_domain_name default
crudini --set /etc/nova/nova.conf keystone_authtoken user_domain_name
default
crudini --set /etc/nova/nova.conf keystone_authtoken project_name
service
crudini --set /etc/nova/nova.conf keystone_authtoken username
nova
crudini --set /etc/nova/nova.conf keystone_authtoken password
bootcamp
crudini --set /etc/nova/nova.conf DEFAULT use_neutron True
crudini --set /etc/nova/nova.conf DEFAULT firewall_driver
nova.virt.firewall.NoopFirewallDriver
crudini --set /etc/nova/nova.conf glance api_servers
http://controller:9292
crudini --set /etc/nova/nova.conf oslo_concurrency lock_path
/var/lib/nova/tmp
crudini --set /etc/nova/nova.conf placement os_region_name
RegionOne
crudini --set /etc/nova/nova.conf placement project_domain_name
Default

crudini --set /etc/nova/nova.conf placement project_name
service
crudini --set /etc/nova/nova.conf placement auth_type password
crudini --set /etc/nova/nova.conf placement user_domain_name
Default
crudini --set /etc/nova/nova.conf placement auth_url
http://controller:35357/v3
crudini --set /etc/nova/nova.conf placement username
placement
crudini --set /etc/nova/nova.conf placement password
bootcamp

crudini --set /etc/nova/nova.conf scheduler
discover_hosts_in_cells_interval 300
```

In the preceding commands, I have given `bootcamp` as the password for all of the authentication parameter. Please replace the password `bootcamp` with the password you chose for your respective user in the identity service.

To configure the VNC settings in `nova-compute` service, do copy and paste the following commands into the terminal.

CAUTION! Please replace the given IP address with the management IP address of your node.

```
crudini --set /etc/nova/nova.conf DEFAULT my_ip 192.168.1.7
crudini --set /etc/nova/nova.conf vnc enabled true
crudini --set /etc/nova/nova.conf vnc vncserver_listen 0.0.0.0
crudini --set /etc/nova/nova.conf vnc vncserver_proxyclient_address
192.168.1.7
crudini --set /etc/nova/nova.conf vnc novncproxy_base_url
http://192.168.1.7:6080/vnc_auto.html
crudini --set /etc/nova/nova-compute.conf libvirt virt_type qemu
```

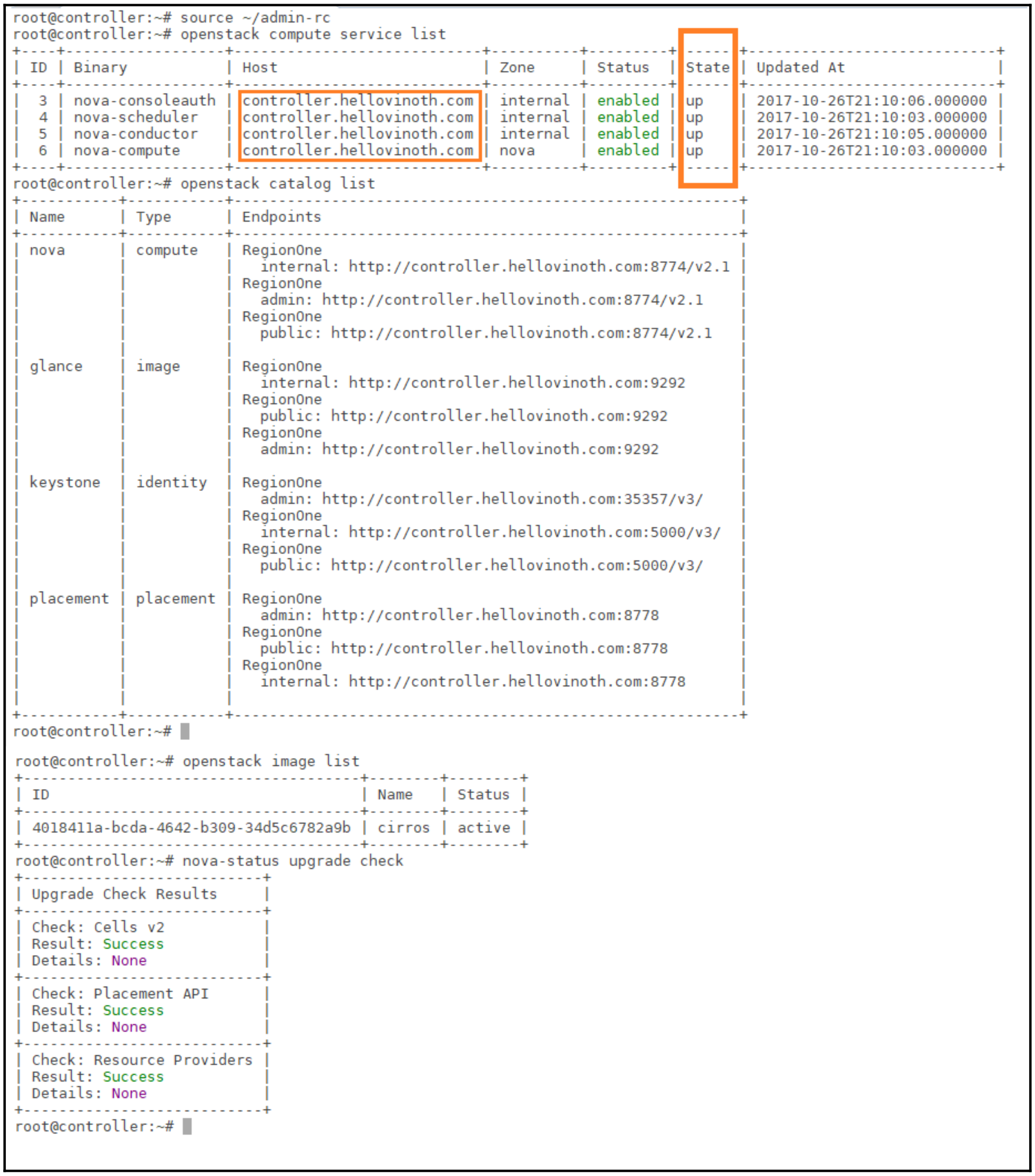
Let's finalize the installation by restarting the `nova-compute` service:

```
service nova-compute restart
```

Now, verify the operation of the compute service by following the following instructions from the controller node. Well! In our case, from the all-in-one node:

```
source ~/admin-rc
openstack compute service list
openstack catalog list
openstack image list
nova-status upgrade check
```

In response to the preceding verification commands, you will receive an output similar to the one here:



```
root@controller:~# source ~/admin-rc
root@controller:~# openstack compute service list
+----+------------------+---------------------------+----------+---------+-------+----------------------------+
| ID | Binary           | Host                      | Zone     | Status  | State | Updated At                 |
+----+------------------+---------------------------+----------+---------+-------+----------------------------+
|  3 | nova-consoleauth | controller.hellovinoth.com | internal | enabled | up    | 2017-10-26T21:10:06.000000 |
|  4 | nova-scheduler   | controller.hellovinoth.com | internal | enabled | up    | 2017-10-26T21:10:03.000000 |
|  5 | nova-conductor   | controller.hellovinoth.com | internal | enabled | up    | 2017-10-26T21:10:05.000000 |
|  6 | nova-compute     | controller.hellovinoth.com | nova     | enabled | up    | 2017-10-26T21:10:03.000000 |
+----+------------------+---------------------------+----------+---------+-------+----------------------------+
root@controller:~# openstack catalog list
+-----------+-----------+-----------------------------------------------------------+
| Name      | Type      | Endpoints                                                 |
+-----------+-----------+-----------------------------------------------------------+
| nova      | compute   | RegionOne                                                 |
|           |           |   internal: http://controller.hellovinoth.com:8774/v2.1  |
|           |           | RegionOne                                                 |
|           |           |   admin: http://controller.hellovinoth.com:8774/v2.1     |
|           |           | RegionOne                                                 |
|           |           |   public: http://controller.hellovinoth.com:8774/v2.1    |
|           |           |                                                           |
| glance    | image     | RegionOne                                                 |
|           |           |   internal: http://controller.hellovinoth.com:9292       |
|           |           | RegionOne                                                 |
|           |           |   public: http://controller.hellovinoth.com:9292         |
|           |           | RegionOne                                                 |
|           |           |   admin: http://controller.hellovinoth.com:9292          |
|           |           |                                                           |
| keystone  | identity  | RegionOne                                                 |
|           |           |   admin: http://controller.hellovinoth.com:35357/v3/     |
|           |           | RegionOne                                                 |
|           |           |   internal: http://controller.hellovinoth.com:5000/v3/   |
|           |           | RegionOne                                                 |
|           |           |   public: http://controller.hellovinoth.com:5000/v3/     |
|           |           |                                                           |
| placement | placement | RegionOne                                                 |
|           |           |   admin: http://controller.hellovinoth.com:8778          |
|           |           | RegionOne                                                 |
|           |           |   public: http://controller.hellovinoth.com:8778         |
|           |           | RegionOne                                                 |
|           |           |   internal: http://controller.hellovinoth.com:8778       |
|           |           |                                                           |
+-----------+-----------+-----------------------------------------------------------+
root@controller:~#

root@controller:~# openstack image list
+--------------------------------------+--------+--------+
| ID                                   | Name   | Status |
+--------------------------------------+--------+--------+
| 4018411a-bcda-4642-b309-34d5c6782a9b | cirros | active |
+--------------------------------------+--------+--------+
root@controller:~# nova-status upgrade check
+---------------------------+
| Upgrade Check Results     |
+---------------------------+
| Check: Cells v2           |
| Result: Success           |
| Details: None             |
+---------------------------+
| Check: Placement API      |
| Result: Success           |
| Details: None             |
+---------------------------+
| Check: Resource Providers |
| Result: Success           |
| Details: None             |
+---------------------------+
root@controller:~#
```

Configuring the networking service (neutron)

In this installation, the various services that power the OpenStack networking will be installed on the controller node. However, some necessary neutron agent configuration files must exist on the compute node as well. In our case, we will be installing and configuring the controller and compute node changes in the single node.

Before we start the actual installation of neutron service, load the admin OpenRC file to gain access to admin-only CLI commands:

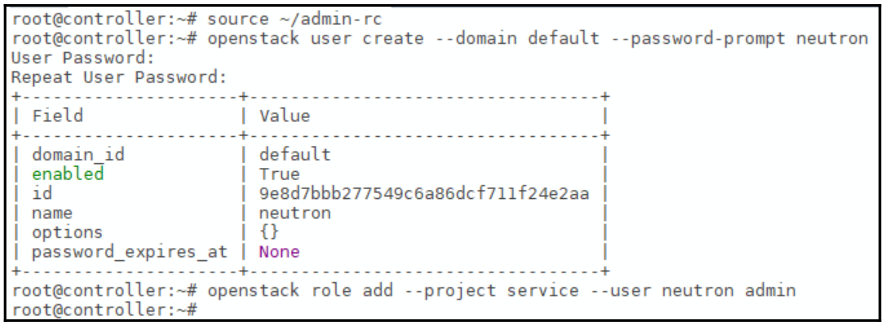
```
source ~/admin-rc
```

Step 1 - defining the neutron service and API endpoints in KeyStone

Follow the instructions to create a dedicated user for neutron service called the `neutron` user, followed by mapping the `admin` role to the `neutron` user on the `service` project:

```
openstack user create --domain default --password-prompt neutron
openstack role add --project service --user neutron admin
```

In response to the preceding commands, you will get an output like the one shown here:

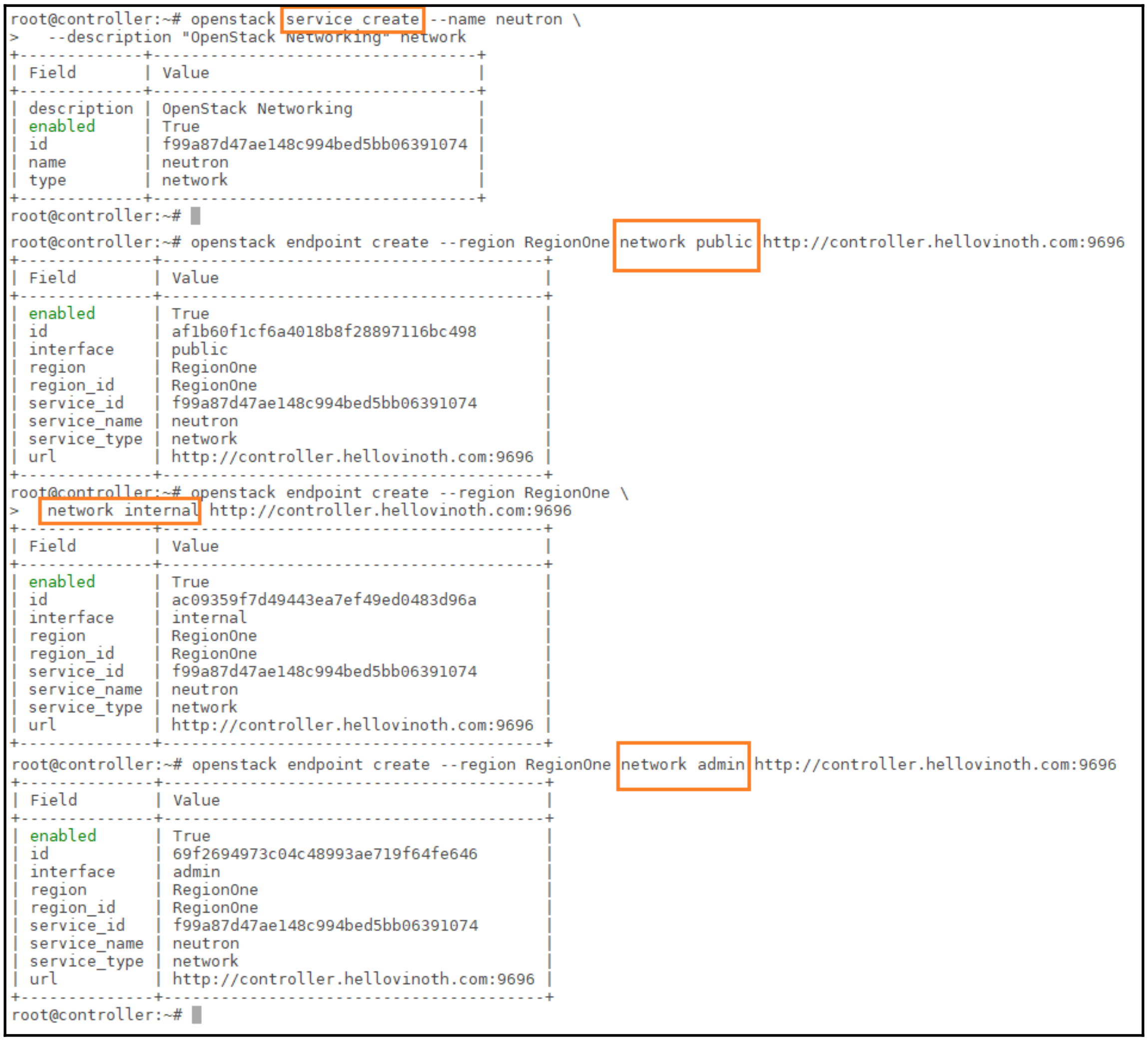


```
root@controller:~# source ~/admin-rc
root@controller:~# openstack user create --domain default --password-prompt neutron
User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 9e8d7bbb277549c6a86dcf711f24e2aa |
| name                | neutron                          |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
root@controller:~# openstack role add --project service --user neutron admin
root@controller:~#
```

Then, we need to create a service entity for the Nova service, followed by API endpoints for the Nova service using the following commands:

```
openstack service create --name neutron \
  --description "OpenStack Networking" network
openstack endpoint create --region RegionOne \
  network public http://controller.hellovinoth.com:9696
openstack endpoint create --region RegionOne \
  network internal http://controller.hellovinoth.com:9696
openstack endpoint create --region RegionOne \
  network admin http://controller.hellovinoth.com:9696
```

In the response output, you will get the information similar to the one in the following figure:



```
root@controller:~# openstack service create --name neutron \
>   --description "OpenStack Networking" network
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Networking             |
| enabled     | True                             |
| id          | f99a87d47ae148c994bed5bb06391074 |
| name        | neutron                          |
| type        | network                          |
+-------------+----------------------------------+
root@controller:~#
root@controller:~# openstack endpoint create --region RegionOne network public http://controller.hellovinoth.com:9696
+--------------+----------------------------------------+
| Field        | Value                                  |
+--------------+----------------------------------------+
| enabled      | True                                   |
| id           | af1b60f1cf6a4018b8f28897116bc498       |
| interface    | public                                 |
| region       | RegionOne                              |
| region_id    | RegionOne                              |
| service_id   | f99a87d47ae148c994bed5bb06391074       |
| service_name | neutron                                |
| service_type | network                                |
| url          | http://controller.hellovinoth.com:9696 |
+--------------+----------------------------------------+
root@controller:~# openstack endpoint create --region RegionOne \
>   network internal http://controller.hellovinoth.com:9696
+--------------+----------------------------------------+
| Field        | Value                                  |
+--------------+----------------------------------------+
| enabled      | True                                   |
| id           | ac09359f7d49443ea7ef49ed0483d96a       |
| interface    | internal                               |
| region       | RegionOne                              |
| region_id    | RegionOne                              |
| service_id   | f99a87d47ae148c994bed5bb06391074       |
| service_name | neutron                                |
| service_type | network                                |
| url          | http://controller.hellovinoth.com:9696 |
+--------------+----------------------------------------+
root@controller:~# openstack endpoint create --region RegionOne network admin http://controller.hellovinoth.com:9696
+--------------+----------------------------------------+
| Field        | Value                                  |
+--------------+----------------------------------------+
| enabled      | True                                   |
| id           | 69f2694973c04c48993ae719f64fe646       |
| interface    | admin                                  |
| region       | RegionOne                              |
| region_id    | RegionOne                              |
| service_id   | f99a87d47ae148c994bed5bb06391074       |
| service_name | neutron                                |
| service_type | network                                |
| url          | http://controller.hellovinoth.com:9696 |
+--------------+----------------------------------------+
root@controller:~#
```

Step 2 - configuring the self-service networks

To install neutron binaries, run the following command on the controller node:

```
apt install neutron-server neutron-plugin-ml2 \
  neutron-linuxbridge-agent neutron-l3-agent neutron-dhcp-agent \
  neutron-metadata-agent -y
```

The neutron configuration file `/etc/neutron/neutron.conf` has loads of settings that should be altered to meet the needs of the OpenStack cloud administrator.

By now, you should have neutron packages installed in your controller. Now, simply copy and paste the following `crudini` commands in the terminal to configure the neutron components:

```
crudini --set /etc/neutron/neutron.conf database connection
mysql+pymysql://neutron:bootcamp@controller/neutron
crudini --set /etc/neutron/neutron.conf DEFAULT core_plugin ml2
crudini --set /etc/neutron/neutron.conf DEFAULT service_plugins
router
crudini --set /etc/neutron/neutron.conf DEFAULT allow_overlapping_ips
true
crudini --set /etc/neutron/neutron.conf DEFAULT transport_url
rabbit://openstack:bootcamp@controller
crudini --set /etc/neutron/neutron.conf DEFAULT auth_strategy
keystone
crudini --set /etc/neutron/neutron.conf keystone_authtoken auth_uri
http://controller:5000
crudini --set /etc/neutron/neutron.conf keystone_authtoken auth_url
http://controller:35357
crudini --set /etc/neutron/neutron.conf keystone_authtoken
memcached_servers
controller:11211

crudini --set /etc/neutron/neutron.conf keystone_authtoken auth_type
password
crudini --set /etc/neutron/neutron.conf keystone_authtoken
project_domain_name
default
crudini --set /etc/neutron/neutron.conf keystone_authtoken user_domain_name
default
crudini --set /etc/neutron/neutron.conf keystone_authtoken project_name
service
crudini --set /etc/neutron/neutron.conf keystone_authtoken username
neutron
crudini --set /etc/neutron/neutron.conf keystone_authtoken password
bootcamp
crudini --set /etc/neutron/neutron.conf DEFAULT
notify_nova_on_port_status_changes
true
crudini --set /etc/neutron/neutron.conf DEFAULT
notify_nova_on_port_data_changes
true
crudini --set /etc/neutron/neutron.conf nova auth_url
http://controller:35357
```

```
crudini --set /etc/neutron/neutron.conf nova auth_type
password
crudini --set /etc/neutron/neutron.conf nova project_domain_name
default
crudini --set /etc/neutron/neutron.conf nova user_domain_name
default
crudini --set /etc/neutron/neutron.conf nova region_name
RegionOne
crudini --set /etc/neutron/neutron.conf nova project_name
service
crudini --set /etc/neutron/neutron.conf nova username
nova
crudini --set /etc/neutron/neutron.conf nova password
bootcamp
```

Note that I have given `bootcamp` as the password for KeyStone authentication. You should replace the password that you chose for the respective user in the identity service.

Step 3 - configuring the Modular Layer 2 (ML2) plugin

The ML2 plug-in uses the Linux bridge mechanism to build layer-2 virtual networking infrastructure for instances.

Copy and paste the following `crudini` commands to configure the ML2 plug-in to work with neutron components:

```
crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2 type_drivers
flat,vlan,vxlan

crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2
tenant_network_types vxlan
crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2
mechanism_drivers linuxbridge,l2population
crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini ml2
extension_drivers port_security
crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini
ml2_type_flat flat_networks provider
crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini
ml2_type_vxlan vni_ranges 1:1000
crudini --set /etc/neutron/plugins/ml2/ml2_conf.ini
securitygroup enable_ipset true
```

Step 4 - Configuring the Linux bridge agent

Copy and paste the following `crudini` commands to configure the Linux bridge agent to work with ML2 components:

```
crudini --set /etc/neutron/plugins/ml2/linuxbridge_agent.ini
linux_bridge physical_interface_mappings provider:enp0s8
```

CAUTION! Replace `enp0s8` with the name of the provider network interface for your environment. Please refer to the *Configuring network interfaces* section to know your provider network interface name.

```
crudini --set /etc/neutron/plugins/ml2/linuxbridge_agent.ini vxlan
local_ip 192.168.1.7
```

In the preceding command, replace `192.168.1.7` with the IP address of your management IP address to handle the VM tunnel traffic:

```
crudini --set /etc/neutron/plugins/ml2/linuxbridge_agent.ini vxlan
enable_vxlan true
crudini --set /etc/neutron/plugins/ml2/linuxbridge_agent.ini vxlan
l2_population true
crudini --set /etc/neutron/plugins/ml2/linuxbridge_agent.ini
securitygroup enable_security_group true
crudini --set /etc/neutron/plugins/ml2/linuxbridge_agent.ini \
  securitygroup firewall_driver \
  neutron.agent.linux.iptables_firewall.IptablesFirewallDriver
```

Step 5 - configuring the layer-3, DHCP agent, and metadata agent

The Layer-3 (L3) agent provides routing and NAT services for self-service virtual networks.

The DHCP agent provides DHCP services for the virtual networks.

The metadata agent provides configuration information, such as credentials to instances.

Copy and paste the following `crudini` commands to configure the L3 agent, DHCP agent, and metadata agent to work with neutron components:

```
crudini --set /etc/neutron/l3_agent.ini DEFAULT interface_driver
linuxbridge
crudini --set /etc/neutron/dhcp_agent.ini DEFAULT interface_driver
linuxbridge
crudini --set /etc/neutron/dhcp_agent.ini DEFAULT dhcp_driver
neutron.agent.linux.dhcp.Dnsmasq
crudini --set /etc/neutron/dhcp_agent.ini DEFAULT enable_isolated_metadata
true
crudini --set /etc/neutron/metadata_agent.ini DEFAULT nova_metadata_ip
controller
crudini --set /etc/neutron/metadata_agent.ini DEFAULT
metadata_proxy_shared_secret METADATA_SECRET
```

Step 6 - configuring the Nova service to use the neutron service

Copy and paste the following `crudini` commands to configure the `nova-service` to work with neutron components:

```
crudini --set /etc/nova/nova.conf neutron url
http://controller:9696
crudini --set /etc/nova/nova.conf neutron auth_url
http://controller:35357
crudini --set /etc/nova/nova.conf neutron auth_type
password
crudini --set /etc/nova/nova.conf neutron project_domain_name
default
crudini --set /etc/nova/nova.conf neutron user_domain_name
default
crudini --set /etc/nova/nova.conf neutron region_name
RegionOne
crudini --set /etc/nova/nova.conf neutron project_name
service
crudini --set /etc/nova/nova.conf neutron username neutron
crudini --set /etc/nova/nova.conf neutron password bootcamp
crudini --set /etc/nova/nova.conf neutron service_metadata_proxy true
crudini --set /etc/nova/nova.conf neutron
metadata_proxy_shared_secret METADATA_SECRET
```

Note that I have given `bootcamp` as the password for KeyStone authentication. You should replace the password that you chose for the neutron user in the identity service.

Now, populate the neutron service database and then finalize the installations:

```
su -s /bin/sh -c "neutron-db-manage \
 --config-file /etc/neutron/neutron.conf \
 --config-file /etc/neutron/plugins/ml2/ml2_conf.ini \
 upgrade head" neutron
```

Let's finalize the installation by restarting the neutron and Nova service:

```
service nova-api restart
service neutron-server restart
service neutron-linuxbridge-agent restart
service neutron-dhcp-agent restart
service neutron-metadata-agent restart
service neutron-l3-agent restart
```

Installing and configuring a compute node (neutron)

In case of multi-node setup, we should configure the networking agent in the compute node to work with the neutron server on the controller. You should repeat the steps in this session when you decide to add an additional compute node to the existing OpenStack environment.

As I mentioned earlier, for all-in-one node setup, we can merge the controller and compute node components in a single node. So, do follow the instructions on the same node to install and configure the neutron compute node components:

```
apt install neutron-linuxbridge-agent -y
```

By now, you should have neutron agent packages installed in your compute node. Now, simply copy and paste the `crudini` commands in the terminal to configure the neutron agent:

```
crudini --set /etc/neutron/neutron.conf DEFAULT transport_url
rabbit://openstack:bootcamp@controller
crudini --set /etc/neutron/neutron.conf DEFAULT auth_strategy
keystone
crudini --set /etc/neutron/neutron.conf keystone_authtoken
auth_uri http://controller:5000
crudini --set /etc/neutron/neutron.conf keystone_authtoken
auth_url http://controller:35357
crudini --set /etc/neutron/neutron.conf keystone_authtoken
memcached_servers controller:11211
crudini --set /etc/neutron/neutron.conf keystone_authtoken
auth_type password
crudini --set /etc/neutron/neutron.conf keystone_authtoken
project_domain_name default
crudini --set /etc/neutron/neutron.conf keystone_authtoken
user_domain_name default
crudini --set /etc/neutron/neutron.conf keystone_authtoken
project_name service
crudini --set /etc/neutron/neutron.conf keystone_authtoken
username neutron
crudini --set /etc/neutron/neutron.conf keystone_authtoken
password bootcamp
```

Note that I have given `bootcamp` as the password for KeyStone authentication. You should replace the password that you chose for the respective user.

Now, copy and paste the `crudini` commands to configure the compute node's Linux bridge agent to work with ML2 components:

```
crudini --set /etc/neutron/plugins/ml2/linuxbridge_agent.ini
linux_bridge physical_interface_mappings provider:enp0s8
```

CAUTION! Replace `enp0s8` with the name of the provider network interface for your compute node environment. Please refer to the *Configuring network interfaces* section to know your provider network interface name.

```
crudini --set /etc/neutron/plugins/ml2/linuxbridge_agent.ini vxlan local_ip
192.168.1.7
```

In the preceding command, replace `192.168.1.7` with the IP address of your management network of compute node to handle the VM tunnel traffic. In the all-in-one node setup, the IP address of the management network is going to remain same for the controller and compute nodes:

```
crudini --set /etc/neutron/plugins/ml2/linuxbridge_agent.ini vxlan
enable_vxlan true
crudini --set /etc/neutron/plugins/ml2/linuxbridge_agent.ini vxlan
l2_population true
crudini --set /etc/neutron/plugins/ml2/linuxbridge_agent.ini
securitygroup enable_security_group true
crudini --set /etc/neutron/plugins/ml2/linuxbridge_agent.ini \
  securitygroup firewall_driver \
  neutron.agent.linux.iptables_firewall.IptablesFirewallDriver
```

Then, copy and paste the `crudini` commands to configure the `nova-compute` service to work with neutron components:

```
crudini --set /etc/nova/nova.conf neutron url http://controller:9696
crudini --set /etc/nova/nova.conf neutron auth_url http://controller:35357
crudini --set /etc/nova/nova.conf neutron auth_type password
crudini --set /etc/nova/nova.conf neutron project_domain_name default
crudini --set /etc/nova/nova.conf neutron user_domain_name default
crudini --set /etc/nova/nova.conf neutron region_name RegionOne
crudini --set /etc/nova/nova.conf neutron project_name service
crudini --set /etc/nova/nova.conf neutron username neutron
crudini --set /etc/nova/nova.conf neutron password bootcamp
```

Note that I have given `bootcamp` as the password for KeyStone authentication. You should replace the password that you chose for the neutron user in the identity service.

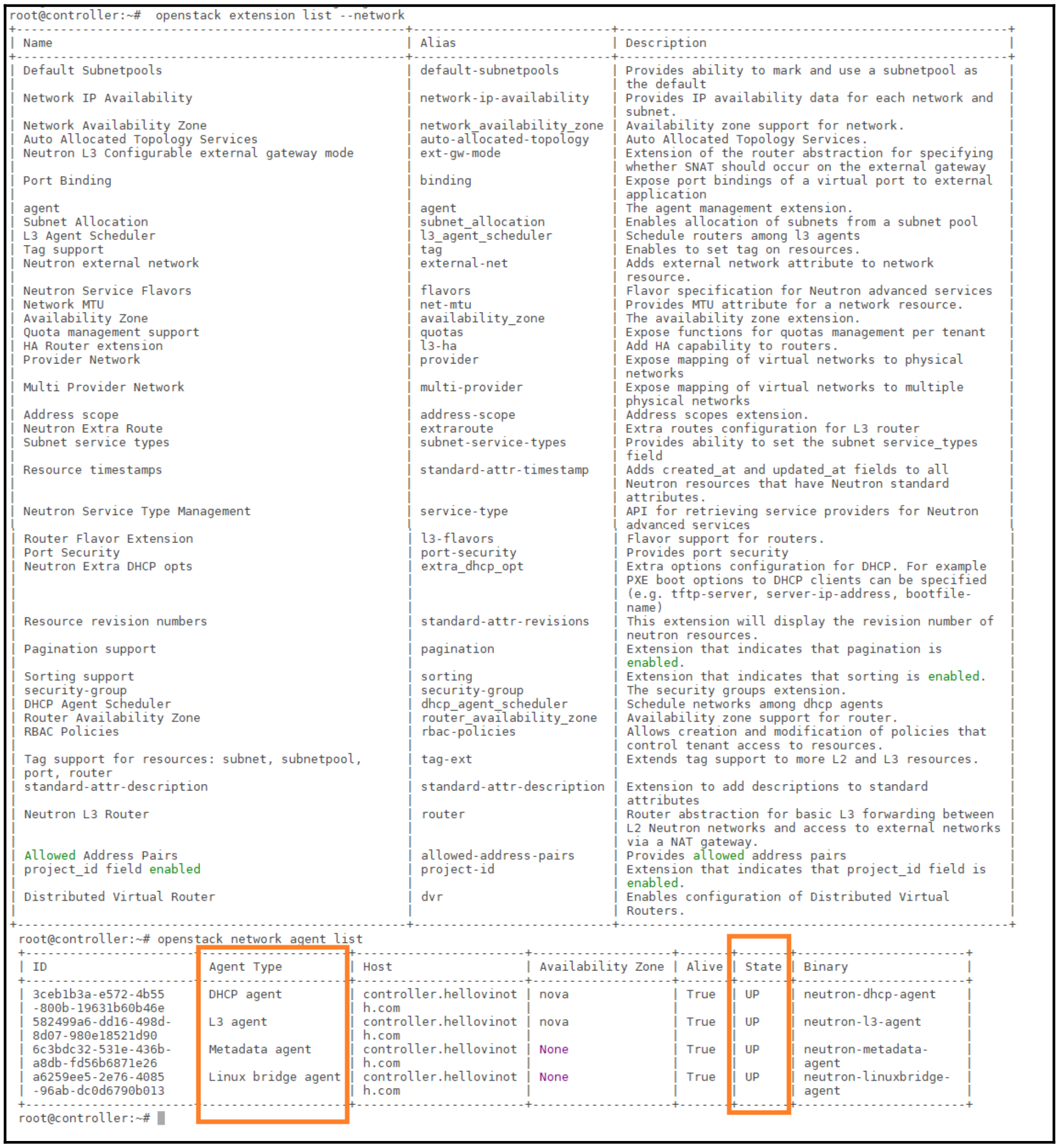
Let's finalize the installation by restarting the neutron and Nova services:

```
service nova-compute restart
service neutron-linuxbridge-agent restart
```

Now, verify the operation of the neutron service by following the instructions from the controller node. Well! In our case, from the all-in-one node:

```
source ~/admin-rc
openstack extension list -network
openstack network agent list
```

You may refer to the following figure as the reference for the output of the preceding commands:



```
root@controller:~#  openstack extension list --network
+-----------------------------------------------------+---------------------------+----------------------------------------------------+
| Name                                                | Alias                     | Description                                        |
+-----------------------------------------------------+---------------------------+----------------------------------------------------+
| Default Subnetpools                                 | default-subnetpools       | Provides ability to mark and use a subnetpool as   |
|                                                     |                           | the default                                        |
| Network IP Availability                             | network-ip-availability   | Provides IP availability data for each network and |
|                                                     |                           | subnet.                                            |
| Network Availability Zone                           | network_availability_zone | Availability zone support for network.             |
| Auto Allocated Topology Services                    | auto-allocated-topology   | Auto Allocated Topology Services.                  |
| Neutron L3 Configurable external gateway mode       | ext-gw-mode               | Extension of the router abstraction for specifying |
|                                                     |                           | whether SNAT should occur on the external gateway  |
| Port Binding                                        | binding                   | Expose port bindings of a virtual port to external |
|                                                     |                           | application                                        |
| agent                                               | agent                     | The agent management extension.                    |
| Subnet Allocation                                   | subnet_allocation         | Enables allocation of subnets from a subnet pool   |
| L3 Agent Scheduler                                  | l3_agent_scheduler        | Schedule routers among l3 agents                   |
| Tag support                                         | tag                       | Enables to set tag on resources.                   |
| Neutron external network                            | external-net              | Adds external network attribute to network         |
|                                                     |                           | resource.                                          |
| Neutron Service Flavors                             | flavors                   | Flavor specification for Neutron advanced services |
| Network MTU                                         | net-mtu                   | Provides MTU attribute for a network resource.     |
| Availability Zone                                   | availability_zone         | The availability zone extension.                   |
| Quota management support                            | quotas                    | Expose functions for quotas management per tenant  |
| HA Router extension                                 | l3-ha                     | Add HA capability to routers.                      |
| Provider Network                                    | provider                  | Expose mapping of virtual networks to physical     |
|                                                     |                           | networks                                           |
| Multi Provider Network                              | multi-provider            | Expose mapping of virtual networks to multiple     |
|                                                     |                           | physical networks                                  |
| Address scope                                       | address-scope             | Address scopes extension.                          |
| Neutron Extra Route                                 | extraroute                | Extra routes configuration for L3 router           |
| Subnet service types                                | subnet-service-types      | Provides ability to set the subnet service_types   |
|                                                     |                           | field                                              |
| Resource timestamps                                 | standard-attr-timestamp   | Adds created_at and updated_at fields to all       |
|                                                     |                           | Neutron resources that have Neutron standard       |
|                                                     |                           | attributes.                                        |
| Neutron Service Type Management                     | service-type              | API for retrieving service providers for Neutron   |
|                                                     |                           | advanced services                                  |
| Router Flavor Extension                             | l3-flavors                | Flavor support for routers.                        |
| Port Security                                       | port-security             | Provides port security                             |
| Neutron Extra DHCP opts                             | extra_dhcp_opt            | Extra options configuration for DHCP. For example  |
|                                                     |                           | PXE boot options to DHCP clients can be specified  |
|                                                     |                           | (e.g. tftp-server, server-ip-address, bootfile-    |
|                                                     |                           | name)                                              |
| Resource revision numbers                           | standard-attr-revisions   | This extension will display the revision number of |
|                                                     |                           | neutron resources.                                 |
| Pagination support                                  | pagination                | Extension that indicates that pagination is        |
|                                                     |                           | enabled.                                           |
| Sorting support                                     | sorting                   | Extension that indicates that sorting is enabled.  |
| security-group                                      | security-group            | The security groups extension.                     |
| DHCP Agent Scheduler                                | dhcp_agent_scheduler      | Schedule networks among dhcp agents                |
| Router Availability Zone                            | router_availability_zone  | Availability zone support for router.              |
| RBAC Policies                                       | rbac-policies             | Allows creation and modification of policies that  |
|                                                     |                           | control tenant access to resources.                |
| Tag support for resources: subnet, subnetpool,      | tag-ext                   | Extends tag support to more L2 and L3 resources.   |
| port, router                                        |                           |                                                    |
| standard-attr-description                           | standard-attr-description | Extension to add descriptions to standard          |
|                                                     |                           | attributes                                         |
| Neutron L3 Router                                   | router                    | Router abstraction for basic L3 forwarding between |
|                                                     |                           | L2 Neutron networks and access to external networks|
|                                                     |                           | via a NAT gateway.                                 |
| Allowed Address Pairs                               | allowed-address-pairs     | Provides allowed address pairs                     |
| project_id field enabled                            | project-id                | Extension that indicates that project_id field is  |
|                                                     |                           | enabled.                                           |
| Distributed Virtual Router                          | dvr                       | Enables configuration of Distributed Virtual       |
|                                                     |                           | Routers.                                           |
+-----------------------------------------------------+---------------------------+----------------------------------------------------+
root@controller:~# openstack network agent list
+----------------------+--------------------+----------------------+-------------------+-------+-------+------------------------+
| ID                   | Agent Type         | Host                 | Availability Zone | Alive | State | Binary                 |
+----------------------+--------------------+----------------------+-------------------+-------+-------+------------------------+
| 3ceb1b3a-e572-4b55   | DHCP agent         | controller.hellovinot | nova             | True  | UP    | neutron-dhcp-agent     |
| -800b-19631b60b46e   |                    | h.com                |                   |       |       |                        |
| 582499a6-dd16-498d-  | L3 agent           | controller.hellovinot | nova             | True  | UP    | neutron-l3-agent       |
| 8d07-980e18521d90    |                    | h.com                |                   |       |       |                        |
| 6c3bdc32-531e-436b-  | Metadata agent     | controller.hellovinot | None             | True  | UP    | neutron-metadata-      |
| a8db-fd56b6871e26    |                    | h.com                |                   |       |       | agent                  |
| a6259ee5-2e76-4085   | Linux bridge agent | controller.hellovinot | None             | True  | UP    | neutron-linuxbridge-   |
| -96ab-dc0d6790b013   |                    | h.com                |                   |       |       | agent                  |
+----------------------+--------------------+----------------------+-------------------+-------+-------+------------------------+
root@controller:~# █
```

Note that the actual output may differ slightly from the preceding figure.

Installing the OpenStack dashboard

By now, your OpenStack environment has all of the core components that are necessary to provision a basic virtual machine. You may use the OpenStack CLI to launch the one by referring to `Chapter 6`, *Day 6 - Field Training Exercise*.

The OpenStack dashboard, also known as Horizon, provides a web-based user interface to OpenStack services, including compute, networking, storage, and identity, among others.

Follow the steps to install and configure the OpenStack Horizon on the controller node:

```
apt install openstack-dashboard -y
```

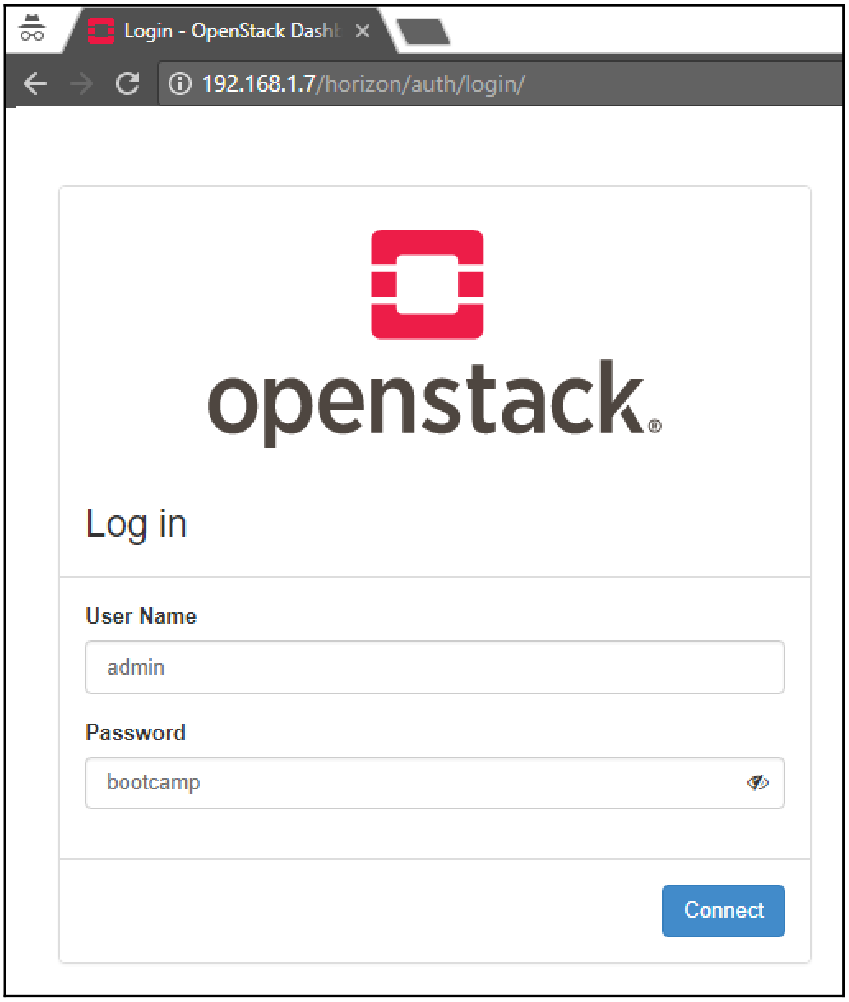
Copy and paste the snippet on the controller terminal to configure the OpenStack dashboard:

```
sed -i 's/ubuntu/default/' /etc/openstack-dashboard/local_settings.py
sed -i 's/v2.0/v3/' /etc/openstack-dashboard/local_settings.py
sed -i 's/_member_/user/' /etc/openstack-dashboard/local_settings.py
sed -i 's/127.0.0.1/controller/' /etc/openstack-
dashboard/local_settings.py
cat >> /etc/openstack-dashboard/local_settings.py <<EOF
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
OPENSTACK_KEYSTONE_DEFAULT_DOMAIN = "Default"
OPENSTACK_API_VERSIONS = {
    "identity": 3,
    "image": 2,
}
EOF
service apache2 restart
```

You need to restart the apache service for the new configuration changes to take effect in the web UI. Alternatively, you could also refer to:
`https://docs.openstack.org/ocata/install-guide-ubuntu/horizon-install.html` to know more about an available horizon configuration.

Now, verify the Horizon operation by accessing the Horizon dashboard using a web browser at: `http://<Management_IP_address>/horizon`.

In my case, I could access my OpenStack web UI at: `http://192.168.1.7/horizon` from my laptop web browser. In response, you will see the dashboard login page like the one shown here:



Authenticate the dashboard using admin or demo user credentials to log in and manage the OpenStack resources.

Adding compute node to the OpenStack cluster

By now, the OpenStack identity, image, networking, dashboard, and compute services have been installed and configured in the single node to build an all-in-one node OpenStack setup.

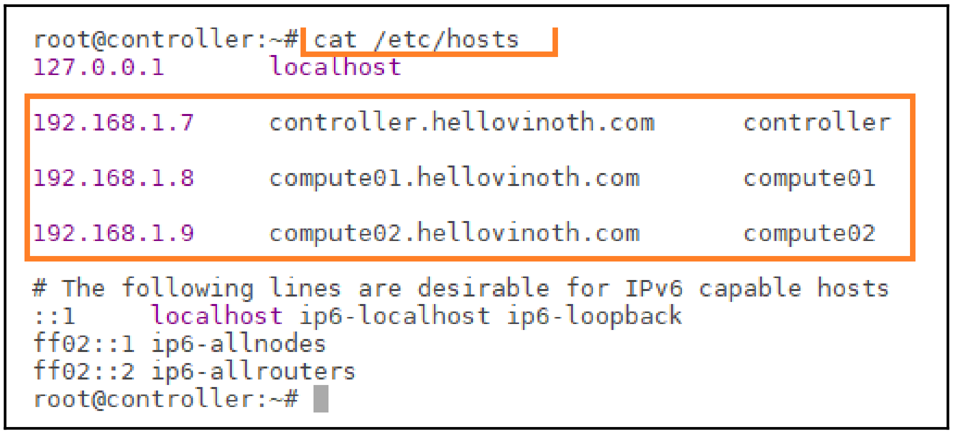
If you wish to add an additional compute node to the existing OpenStack setup, do repeat the steps from the following-mentioned sessions on the newly chosen compute node:

- *Before we begin*
- *Installing and configuring a compute node (nova-compute)*
- *Installing and configuring a compute node (neutron)*

Note that, while repeating the steps from the preceding sections, you should replace the management IP address of the new compute node and hostname appropriately in the configuration files and `crudini` commands.

Also, you need to update the `/etc/hosts` file with a newly added compute hostname, and sync the entries across all of the nodes in the OpenStack cluster.

For example, after adding two compute nodes to my OpenStack cluster, my `/etc/hosts` file across all of the nodes in my OpenStack cluster would look like the one here:



```
root@controller:~# cat /etc/hosts
127.0.0.1       localhost

192.168.1.7     controller.hellovinoth.com      controller

192.168.1.8     compute01.hellovinoth.com       compute01

192.168.1.9     compute02.hellovinoth.com       compute02

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
root@controller:~#
```

Summary

We have gone through a step-by-step package-based installation of the Ocata release of OpenStack on the Ubuntu operating system. At this point, you have the OpenStack setup built on your own from scratch. Now, you could start playing with your own private cloud by redoing the exercise from the previous chapter, `Chapter 6`, *Day 6 - Field Training Exercise*.

In the next chapter, we will walk through the most commonly faced issues in OpenStack and the steps to troubleshoot it.

9

Day 9 - Repairing OpenStack

Troubleshooting OpenStack is not always upfront because OpenStack is distributed across several different projects, all working together in providing public/private cloud functionalities. Adding to this, with OpenStack's flexibility to support external open source technology, comes the challenge of identifying the root cause of errors and problems.

In this chapter, we will walk through OpenStack troubleshooting methodologies that will help you learn the tips, and the right approach to handle OpenStack issues that may come your way while installing and managing the cloud.

Structured troubleshooting approaches

You must thoroughly understand each OpenStack component's functionality and its interrelation between the services to nail down issues and fix them. The more you understand the OpenStack architecture, the more successful you will be at troubleshooting it.

In my experience, I have found that the below-structured troubleshooting order works well when troubleshooting OpenStack issues to identify their root cause and resolve them:

- **Service status**: Check that the required services are up and running
- **Authentication response**: Confirm that the authentication is configured correctly
- **CLI debug mode**: Run the CLI commands in the debug mode while looking for error messages
- **Check service logs**: Check log files for errors messages

If you find any error messages pop up in the Horizon dashboard or in response to the OSC CLI, you must have some idea about the service that the error message is related to.

For example, if you see an error message stating `Unable to retrieve image list`. By now, you will have guessed that the error message is something related to the Glance service. Yes, there could be other reasons behind the error messages. However, as the first step, we need to focus on the Glance services to ensure that everything is working fine from the image service end, before digging deeper into other services logs.

To start with, let's quickly walk through, how to follow the previous discussed troubleshooting approach to analyze the root cause for the previous error message `Unable to retrieve image list`.

Level 1 - Service status check

By now, we know that the error message is something related to the Glance services. So, as an initial step, we need to confirm that all the components in the Glance services are up and running, namely the `glance-api` service and the `glance-registry` service. You can use the following service status commands to do this:

```
service glance-api status
service glance-registry status
```

Level 2 - Authentication response

In most cases, when working with the OpenStack client CLI, you will receive an error response asking to provide proper authentication information or to set an environment variable in the CLI parameter.

This error typically occurs when you haven't provided all the necessary attributes to the OSC CLI. Usually, sourcing an OpenRC file that contains the attributes required for the command-line client solves this problem.

Level 3 - CLI debug mode

Executing the command may have an organized workflow behind the screen that will end up in displaying the expected output in response. However, in case of issues, we will receive an error message that will have only minimal information about the issue. To trace the workflow and know more about the error, we can use the `--debug` parameter added to the actual command.

For example, execute the following two commands and see the difference in the output:

```
glance image-list
glance --debug image-list
```

Level 4 - Service log check

In most cases, we will end up searching for an `Error` and `Trace` keyword in the respective log files to nail down the root cause. For examining the log files, you should be aware of the path of every log file that each OpenStack services write to.

I have tabulated the OpenStack service name and the respective log location in the following table:

Service name	**Log files location**

Horizon	Controller node:
	`/var/log/apache2/horizon_access.log`
	`/var/log/apache2/horizon_error.log`

KeyStone	Controller node:
	`/var/log/apache2/error.log`
	`/var/log/apache2/access.log`
	`/var/log/apache2/keystone_wsgi_admin_access.log`
	`/var/log/apache2/keystone_wsgi_admin_error.log`
	`/var/log/apache2/keystone_wsgi_main_access.log`
	`/var/log/apache2/keystone_wsgi_main_error.log`

MySQL	Controller node:
	`/var/log/syslog`
	`/var/log/mysql/error.log`

RabbitMQ	Controller node: `/var/log/rabbitmq/`
Nova	Controller and compute node: `/var/log/nova/`
Glance	Controller node: `/var/log/glance/`
Neutron	Controller and compute node: `/var/log/neutron/`

Service name	Log files location
Cinder	Storage node: `/var/log/cinder/`

In the preceding table, I have mentioned the home path for the log files of the OpenStack services. In the latter part of this chapter, we will cover the log files location of each OpenStack component in detail.

The KeyStone service

Though the KeyStone project does not depend on any other OpenStack projects in OpenStack, all other OpenStack services depend on KeyStone for identity, token management, a service catalog, and policy functionality. This dependency on the KeyStone service means that problems with your KeyStone services may cause problems for many of the other OpenStack services as well. So, it is always a good practice to ensure that KeyStone is operating as expected before we start focusing on other projects logs.

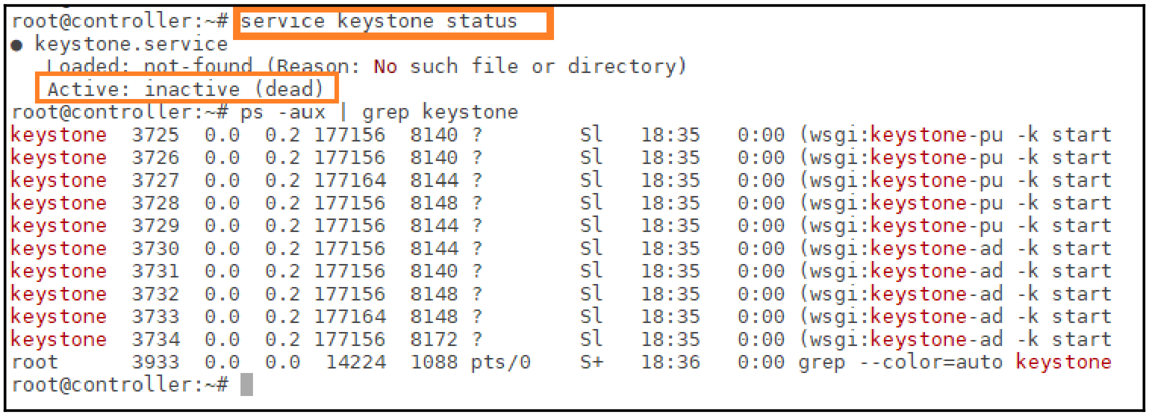
Checking the KeyStone service

Unlike other services in OpenStack, the KeyStone service is available through the Apache server. To check the status of the KeyStone service, we need to check if the Apache server is running. Notably, the eventlet-based KeyStone service will be set in stopped state by default.

Use the following command to verify if the KeyStone service is running:

```
ps -aux | grep keystone
service apache2 status
```

The output should look similar to the one here:

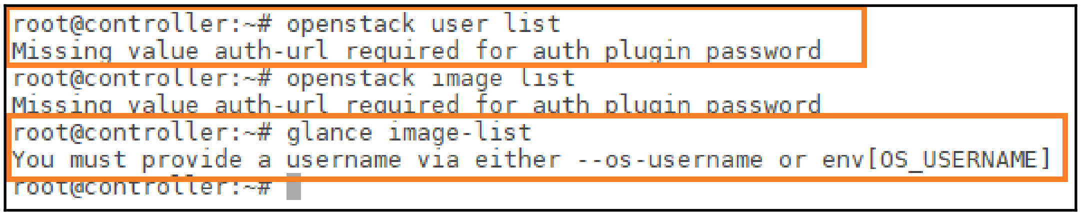


```
root@controller:~# service keystone status
● keystone.service
   Loaded: not-found (Reason: No such file or directory)
   Active: inactive (dead)
root@controller:~# ps -aux | grep keystone
keystone   3725  0.0  0.2 177156  8140 ?        Sl   18:35   0:00 (wsgi:keystone-pu -k start
keystone   3726  0.0  0.2 177156  8140 ?        Sl   18:35   0:00 (wsgi:keystone-pu -k start
keystone   3727  0.0  0.2 177164  8144 ?        Sl   18:35   0:00 (wsgi:keystone-pu -k start
keystone   3728  0.0  0.2 177156  8148 ?        Sl   18:35   0:00 (wsgi:keystone-pu -k start
keystone   3729  0.0  0.2 177156  8144 ?        Sl   18:35   0:00 (wsgi:keystone-pu -k start
keystone   3730  0.0  0.2 177156  8144 ?        Sl   18:35   0:00 (wsgi:keystone-ad -k start
keystone   3731  0.0  0.2 177156  8140 ?        Sl   18:35   0:00 (wsgi:keystone-ad -k start
keystone   3732  0.0  0.2 177156  8148 ?        Sl   18:35   0:00 (wsgi:keystone-ad -k start
keystone   3733  0.0  0.2 177164  8148 ?        Sl   18:35   0:00 (wsgi:keystone-ad -k start
keystone   3734  0.0  0.2 177156  8172 ?        Sl   18:35   0:00 (wsgi:keystone-ad -k start
root       3933  0.0  0.0  14224  1088 pts/0    S+   18:36   0:00 grep --color=auto keystone
root@controller:~#
```

You will have noticed from the preceding output that the eventlet-based KeyStone process should be in the stopped status, as the KeyStone service is available through the Apache server.

Checking the KeyStone client

Before you use any OSC CLI, make sure that you have sourced the credentials from your OpenRC file or alternatively, you need to pass the required AUTH attributes in all the commands. If you have forgotten to take one of these steps, you might see an error as follows:



```
root@controller:~# openstack user list
Missing value auth-url required for auth plugin password
root@controller:~# openstack image list
Missing value auth-url required for auth plugin password
root@controller:~# glance image-list
You must provide a username via either --os-username or env[OS_USERNAME]
root@controller:~#
```

For more information on how to create and source OpenRC files, refer to *Configuring the identity service (KeyStone)* section in `Chapter 8`, *Day 8 - Build Your OpenStack*.

The CLI debug mode

You can run an OSC CLI in the debug mode by merely adding the `--debug` parameter to the command. Take the following command as an example:

```
openstack --debug user list
openstack --debug project list
```

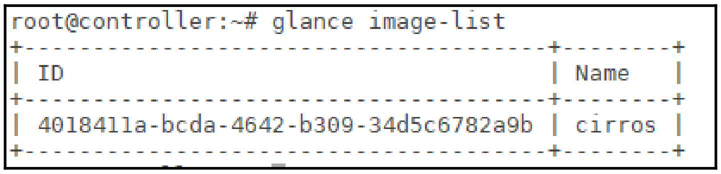
Like I mentioned earlier, running any commands in the `--debug` mode will enable the debug lines to be printed to the output console. The debug mode enabled output may contain the details of the API request sent by the command-line tool and the response body sent back from the API.

The Glance service

When troubleshooting the Glance service, the first thing is to ensure that the Glance service is up and running. You can confirm this by running any simple Glance commands from the command line as follows:

```
service glance-api status
service glance-registry status
glance image-list
```

When Glance functions correctly, the preceding command should return a list of images in your Glance repository, like the following output:



```
root@controller:~# glance image-list
+--------------------------------------+--------+
| ID                                   | Name   |
+--------------------------------------+--------+
| 4018411a-bcda-4642-b309-34d5c6782a9b | cirros |
+--------------------------------------+--------+
```

The Glance log files

The corresponding log file of each Glance service is stored in the `/var/log/glance/` folder on the node in which the respective service runs:

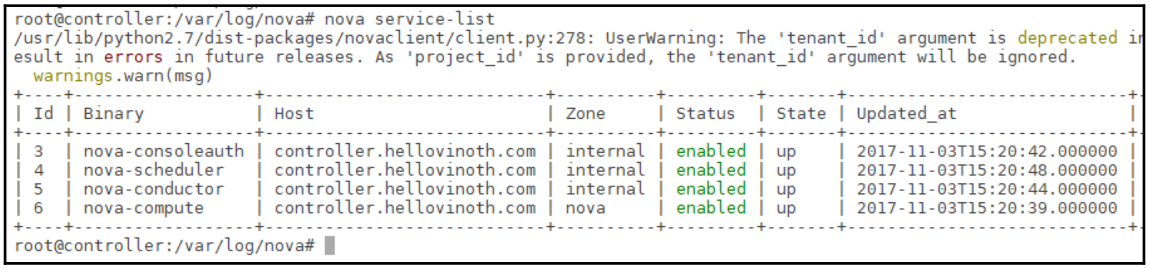
Service name	**Log file name**
`glance-api`	`/var/log/glance/glance-api.log`
`glance-registry`	`/var/log/glance/glance-registry.log`

The Nova service

Nova is one of the core projects in OpenStack. Notably, it is one of the largest in terms of lines of code and also the oldest OpenStack projects. When troubleshooting the Nova service, the first thing is to ensure that the Nova service is up and running. You can confirm this by running any Nova command from the controller node as follows:

```
service nova-api status
service nova-consoleauth status
service nova-scheduler status
service nova-conductor status
service nova-novncproxy status
nova service-list
```

If the Nova service operates correctly, the preceding command should return the status of Nova sub-services, similar to the following output:



```
root@controller:/var/log/nova# nova service-list
/usr/lib/python2.7/dist-packages/novaclient/client.py:278: UserWarning: The 'tenant_id' argument is deprecated ir
esult in errors in future releases. As 'project_id' is provided, the 'tenant_id' argument will be ignored.
  warnings.warn(msg)
+----+------------------+---------------------------+----------+---------+-------+----------------------------+
| Id | Binary           | Host                      | Zone     | Status  | State | Updated_at                 |
+----+------------------+---------------------------+----------+---------+-------+----------------------------+
| 3  | nova-consoleauth | controller.hellovinoth.com | internal | enabled | up    | 2017-11-03T15:20:42.000000 |
| 4  | nova-scheduler   | controller.hellovinoth.com | internal | enabled | up    | 2017-11-03T15:20:48.000000 |
| 5  | nova-conductor   | controller.hellovinoth.com | internal | enabled | up    | 2017-11-03T15:20:44.000000 |
| 6  | nova-compute     | controller.hellovinoth.com | nova     | enabled | up    | 2017-11-03T15:20:39.000000 |
+----+------------------+---------------------------+----------+---------+-------+----------------------------+
root@controller:/var/log/nova#
```

In the compute node, you can use the following command to ensure that the `nova-compute` service is up and running:

```
service nova-compute status
```

The Nova log files

The corresponding log file of each Nova service is located in the `/var/log/nova/` folder on the node in which the respective service is running:

Service name	Log file name
`nova-api`	`/var/log/nova/nova-api.log`
`nova-consoleauth`	`/var/log/nova/nova-consoleauth.log`
`nova-scheduler`	`/var/log/nova/nova-scheduler.log`
`nova-conductor`	`/var/log/nova/nova-conductor.log`
`nova-novncproxy`	`/var/log/nova/nova-novncproxy.log`
`nova-compute`	`/var/log/nova/nova-compute.log`
`nova-placement-api`	`/var/log/nova/nova-placement-api.log`

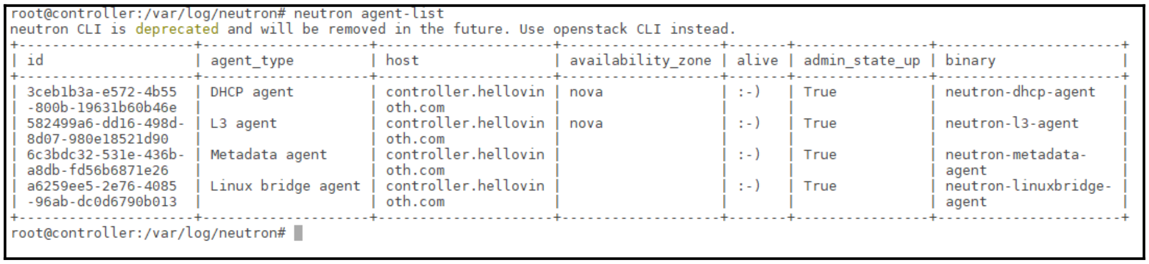
The Neutron service

The Neutron service is one of the most complex OpenStack projects to troubleshoot. Due to its flexible nature of allowing various configurations and plugins to support different networking vendors, it is more complicated to pinpoint the root cause of the issue. However, in-depth understanding of the OpenStack architecture and the fundamental knowledge of networking will help identify and isolate the problems as they arise.

When you are about to troubleshoot the networking service for issues, the first thing is to ensure that all the Neutron services are up and running. You can confirm this by running any Nova command from the controller node, as follows:

```
service neutron-server status
service neutron-linuxbridge-agent status
service neutron-dhcp-agent status
service neutron-metadata-agent status
service neutron-l3-agent status
neutron agent-list
```

If the Neutron service operates correctly, the preceding command should return the status of neutron agents, like the one in the following output:



```
root@controller:/var/log/neutron# neutron agent-list
neutron CLI is deprecated and will be removed in the future. Use openstack CLI instead.
+--------------------+--------------------+---------------------+-------------------+-------+----------------+----------------------+
| id                 | agent_type         | host                | availability_zone | alive | admin_state_up | binary               |
+--------------------+--------------------+---------------------+-------------------+-------+----------------+----------------------+
| 3ceb1b3a-e572-4b55 | DHCP agent         | controller.hellovin | nova              | :-)   | True           | neutron-dhcp-agent   |
| -800b-19631b60b46e |                    | oth.com             |                   |       |                |                      |
| 582499a6-dd16-498d-| L3 agent           | controller.hellovin | nova              | :-)   | True           | neutron-l3-agent     |
| 8d07-980e18521d90  |                    | oth.com             |                   |       |                |                      |
| 6c3bdc32-531e-436b-| Metadata agent     | controller.hellovin |                   | :-)   | True           | neutron-metadata-    |
| a8db-fd56b6871e26  |                    | oth.com             |                   |       |                | agent                |
| a6259ee5-2e76-4085 | Linux bridge agent | controller.hellovin |                   | :-)   | True           | neutron-linuxbridge- |
| -96ab-dc0d6790b013 |                    | oth.com             |                   |       |                | agent                |
+--------------------+--------------------+---------------------+-------------------+-------+----------------+----------------------+
root@controller:/var/log/neutron#
```

Neutron log files

The corresponding log files for each neutron service is in the `/var/log/neutron/` directory of the host on which each service runs:

Service name	**Log file name**
`neutron-server`	`/var/log/neutron/neutron-server.log`
`neutron-linuxbridge-agent`	`/var/log/neutron/neutron-linuxbridge-agent.log`
`neutron-dhcp-agent`	`/var/log/neutron/neutron-dhcp-agent.log`
`neutron-metadata-agent`	`/var/log/neutron/neutron-metadata-agent.log`
`neutron-l3-agent`	`/var/log/neutron/neutron-l3-agent.log`

Database issues

In many cases, one will spend more time on troubleshooting the specific OpenStack services. But after a long effort, they would discover that the database is indeed the culprit. So, it is better to cross-check and verify that the database is configured and is operating correctly soon after you have confirmed that the OpenStack service is up and running.

You can verify whether MySQL is running by executing the following command:

```
service mysql status
```

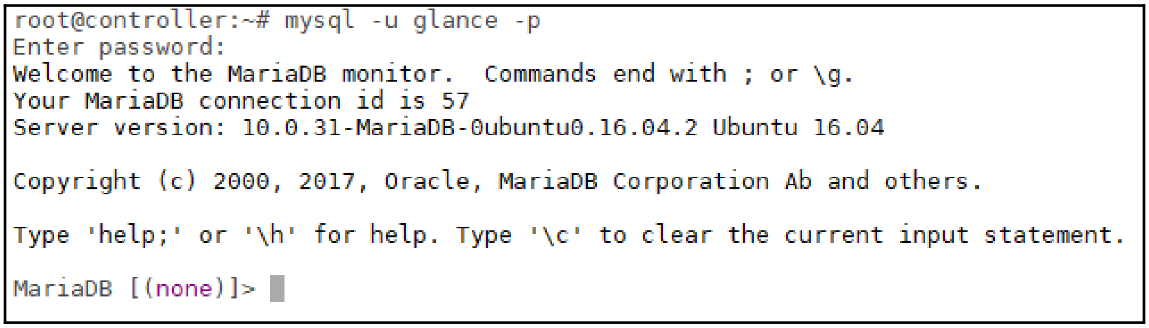
If the preceding command does not return a message indicating that the MySQL process is running, you will need to troubleshoot your database server to get it back to the running state.

If the preceding command shows that the MySQL server is running, then as a next step, you should check that you can connect to the database server. Do follow the following instructions to connect to the MySQL server:

```
mysql -u <db_user > -p
```

Replace `db_user` with the appropriate service username for your database. In my case, I used `glance` as the user and `bootcamp` as the Glance DB password.

If you are successfully connected to MySQL, you will see an output similar to the one shown here:



```
root@controller:~# mysql -u glance -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 57
Server version: 10.0.31-MariaDB-0ubuntu0.16.04.2 Ubuntu 16.04

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> 
```

If you are unsuccessful when trying to connect to the database manually, verify the username and password of the database user you are operating. You should also check whether the user has appropriate privileges to access this database.

Once the credentials and privileges are verified, you should cross-check the configuration file of the appropriate OpenStack service to ensure that the same credentials have been configured under the database session.

How to troubleshoot your database server is outside the scope of this book. However, there are a lot of resources available online that could help.

Getting help from OpenStack community

The spirit of open source and a community-driven development approach has spawned OpenStack as one of the fastest-growing and active open source communities in the world. So, whenever you get stuck with any issues in the OpenStack, you can make use of this significant global open source community.

To connect with the OpenStack community for getting assistance, you may get used to the following ways.

Questions and answers forum (`https://ask.openstack.org/en/questions/`)

Wiki (`https://wiki.openstack.org/wiki/Main_Page`)

Participate in chats on IRC `#openstack`

Join the general mailing list (`http://lists.openstack.org/cgi-bin/mailman/listinfo`)

I would strongly recommend every OpenStack beginner to register at `https://ask.openstack.org/en/questions/` to get started with OpenStack.

Summary

The essence of open source and a community-driven development approach results in a lot of new features getting added to the OpenStack project with each release. When working with such a complex project, it is certain that you will face problems, bugs, errors, issues, and plain old trouble. By now, you will have learned about how to approach these issues, along with knowledge of OpenStack troubleshooting techniques and being able to trace the issues in the respective log files.

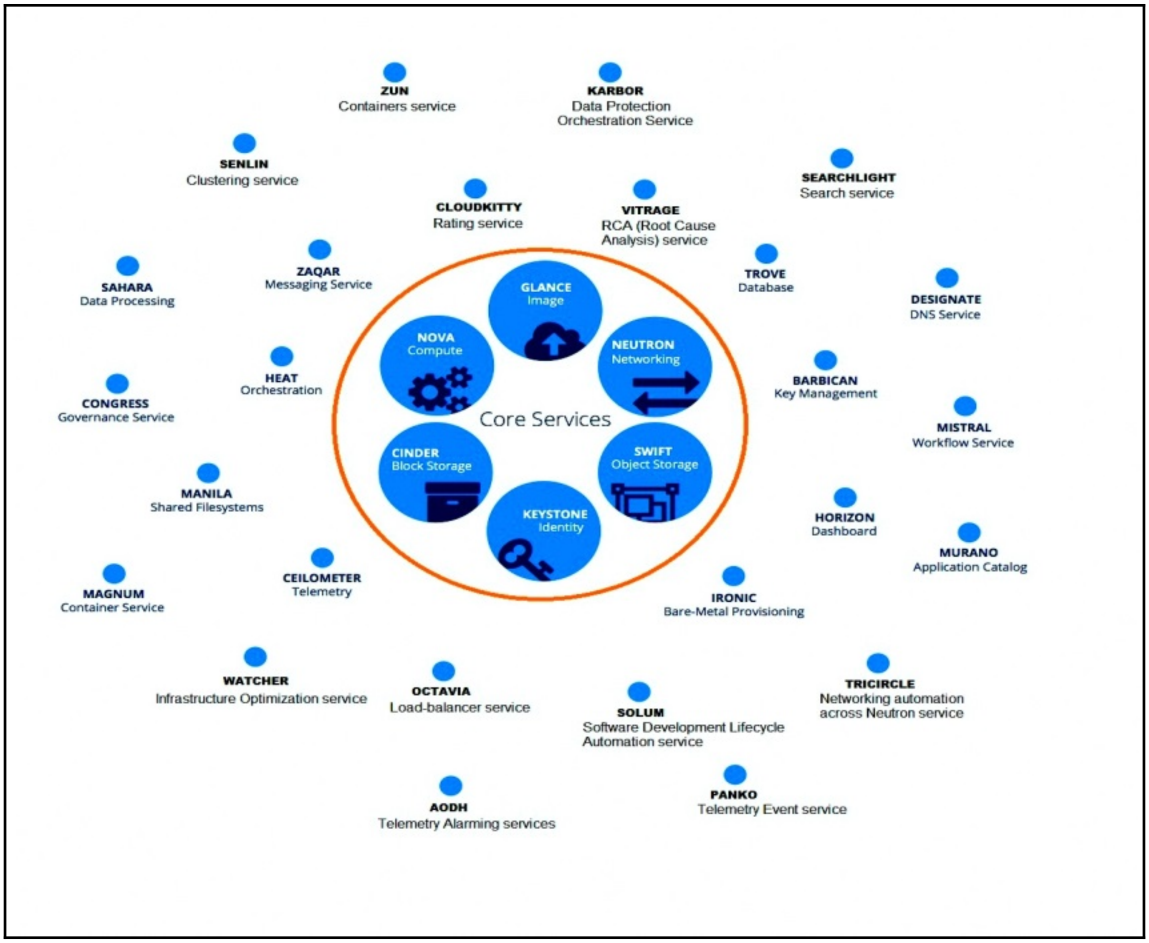
In the next chapter, we will walk through an overview of the optional services available in OpenStack.

10
Day 10 - Side Arms of OpenStack

In this chapter, we will explore non-core but special OpenStack services such as how Trove provides **Database-as-a-Service**, Magnum Container Service, and Murano. We will end the chapter by giving you enough momentum to explore all the other exciting OpenStack services that are being added so frequently.

Bells and whistles of OpenStack

The reason for the success of OpenStack are the 32 components available that are easily pluggable and configurable. Let's quickly explore the most used ones to give you an idea of what each one of them can do with OpenStack:



OpenStack Umbrella

As on OpenStack - Ocata release, the following mentioned 32 projects are officially added under the OpenStack Umbrella:

1. Application Catalog Service (Murano)
2. Bare Metal Service (Ironic)
3. Block Storage Service (Cinder)
4. Clustering Service (Senlin)
5. Compute Service (Nova)

6. Container Infrastructure Management Service (Magnum)
7. Containers Service (Zun)
8. Dashboard (Horizon)
9. Data Processing Service (Sahara)
10. Data Protection Orchestration Service (Karbor)
11. Database Service (Trove)
12. DNS Service (Designate)
13. Governance Service (Congress)
14. Identity Service (Keystone)
15. Image Service (Glance)
16. Infrastructure Optimization Service (Watcher)
17. Key Manager Service (Barbican)
18. Load-balancer Service (Octavia)
19. Messaging Service (Zaqar)
20. Networking Automation Across Neutron Service (Tricircle)
21. Networking Service (Neutron)
22. Object Storage Service (Swift)
23. Orchestration Service (Heat)
24. Rating Service (Cloudkitty)
25. Root Cause Analysis (RCA) Service (Vitrage)
26. Search Service (Searchlight)
27. Shared Filesystems service (Manila)
28. Software Development Lifecycle Automation Service (Solum)
29. Telemetry Alarming Services (Aodh)
30. Telemetry Data Collection Service (Ceilometer)
31. Telemetry Event Service (Panko)
32. Workflow Service (Mistral)

Ironic

Ironic is a code name for the bare metal as a service in OpenStack. The bare metal service enables OpenStack to provide direct access to the physical machines to the end user. As we know, the Nova service will take care of provisioning and manage the virtual machines to the end user. However, in some cases the user may require a bare physical server to run their cloud application. The Ironic project enables the support for OpenStack to provide a bare metal.

The Nova compute service includes a virtualization driver that makes calls to the Ironic service to provide a bare metal node. With the help of Ironic virt driver, the end users can launch a bare metal server instance in the same way that they usually launch a virtual machine instance.

Manila

Manila is a code name for the shared filesystems service in OpenStack. The OpenStack Manila service provides shared file storage to a virtual machine using NFS and CIFS protocols. The shared filesystems service offers an abstraction for managing and provisioning of file shares.

Manila (shared filesystems) - **Amazon Elastic File System** (**EFS**)

Remember that we cannot mount the same Cinder volume on two virtual machines at the same time. However, using manila shared filesystems service, we can mount the same NFS files storage on multiple virtual machines at the same time.

Designate

Designate is a code name for DNS as a service in OpenStack. Designate enables OpenStack with multi-tenant **Domain Name Server** (**DNS**) as a service that provides a REST API with a KeyStone authentication.

Designate (DNS as a service) - AWS Route 53

It is designed to enable end users to configure a route for cloud applications within a specific tenant by translating names such as `http://www.bootcamp.com/` to numeric IP addresses such as `192.168.2.3` that computers use to connect to each other.

Designate supports a variety of DNS servers, including **Bind9** and **PowerDNS**.

Trove

Trove is a code name for **Database-as-a-Service** in OpenStack. The Trove service provides scalable and reliable cloud provisioning functionality for both relational and non-relational database engines. Users can feasibly use database features without the burden of handling complex administrative tasks. Cloud users and database administrators can provide and manage multiple database instances as needed.

For example, to configure a database to work with the application, typically, the end user would launch a new virtual machine and install the database engine, such as MySQL or MongoDB based on their application requirement. Then, the database engine's endpoint and credentials should be configured in the application configuration files. But the Trove service enables the end user to obtain the endpoint for the database directly without handling the complex administrative task of installing and configuring the database in the virtual machine. From the endpoint obtained using the Trove service, the end user will configure their application to work with the database.

Trove (Database-as-a-Service) - **Amazon Relational Database Service** (**RDS**)

The Trove service also offers the resource isolation at high-performance levels and automates complex administrative tasks, such as deployment, configuration, backups, restores, and monitoring.

Sahara

Sahara is a code name for data processing service in OpenStack. Previously known as the Savanna project, then renamed as Sahara, the Sahara project enables the user to provide a data-intensive application cluster, such as Hadoop, Storm, or Spark, on top of the OpenStack cloud.

Barbican

Barbican is a code name for **Key Manager Service** that allows users to create and control the encryption keys used for encrypting your data and also provides secure storage, provisioning, and management of secret data, such as passwords and encryption keys.

Barbican (Key Manager Service) - **AWS Key Management Service** (**KMS**)

Zaqar

Zaqar is a code name for **Messaging Queue as a Service** in OpenStack. Zaqar is a fully managed message queuing service that allows users to decouple and scale microservices, distributed applications, and serverless applications with ease.

Zaqar (Messaging Queue as a service) - **AWS Simple Queue Service** (**SQS**)

Zaqar enables developers to share data between distributed applications performing different tasks, without losing messages or requiring each component to be always available.

Aodh

Aodh is a code name for **Telemetry Alarming Services** in OpenStack. The Telemetry Alarming Services enable the ability to trigger actions based on defined rules against metric or event data collected by the ceilometer service in OpenStack.

Congress

Congress is a code name for the Governance service in OpenStack. Using Congress, a cloud operator can declare, monitor, enforce, and audit policy in a cloud environment. The Congress service collects information about VMs from Nova and network state from Neutron. Congress then pushes this collected input data into its policy engine. Now, Congress verifies that the cloud's actual state abides by the cloud operator's policies.

For example, if the cloud operator defines a new policy that restricts any instance to have a security group rule that allows port 22 access to any public IP address, then the Congress service will collect all the instance information from the Nova service at a regular time interval, and feed the collected data to the policy engine to verify if the cloud state abides by the policy. Congress will trigger the notification if the cloud state does not satisfy the policy.

Mistral

Mistral is a code name for workflow service in OpenStack. Most processes consist of multiple distinct interconnected steps that need to be executed in a specific order in a distributed environment. With Mistral, the user can describe such process as a set of tasks and its task relation, then; upload such description to Mistral so that it takes care of the correct execution order. Mistral also provides flexible task scheduling so that we can run a process according to a specified schedule instead of running it immediately.

For example, a user can use the Mistral service to schedule a task to execute a bash script on specified virtual machines, and then post the output response of the first task as the input for the next task in the workflow through the REST API call, to execute another bash script in other virtual machines. It is essential that several tasks can be combined in a single workflow and run in an on-schedule time. Mistral will take care of their workflow execution.

Murano

Murano is a code name for **Application Catalog Service** in OpenStack. The Murano project enables the cloud operator to publish various cloud-ready applications in a catalog that can be browsed. With Murano, even an inexperienced cloud user can deploy a reliable cloud application in the OpenStack environment with ease.

Magnum

Magnum is the code name for **Container Infrastructure Management Service** in OpenStack. The magnum service is a collection of components that provides support in OpenStack to manage different **Container Orchestration Engines** (**COE**), such as Kubernetes, Apache Mesos, and Docker Swarm.

Magnum uses heat orchestration service to provide a new instance either as a virtual machine or bare metal based on the cluster configuration using the OS image that contains Docker and Kubernetes.

Zun

Zun is a code name for the container service in OpenStack. Zun project provides an OpenStack API for launching and managing containers backed by different container technologies such as Docker.

Don't confuse between container service and container infrastructure management service in OpenStack. Here, the container service (Zun) allows OpenStack to launch and manage the containers directly, where the magnum project enables support for the container orchestration engine in OpenStack.

Panko

Panko is the code name for **Telemetry Event Service** in OpenStack. The Panko project provides the ability to store and query the event data generated by ceilometer. Thus, the Panko service enables users to capture the state information of OpenStack resources at a given time.

Vitrage

Vitrage is a code name for the **Root Cause Analysis** (**RCA**) service in OpenStack. The Vitrage service is used for analyzing, organizing, and expanding the OpenStack alarms and events, yielding insights regarding the root cause of problems and deducing their existence before they are directly detected.

Watcher

Watcher is a code name for the **Infrastructure Optimization Service** in OpenStack.Watcher provides complete optimization solutions, including metrics receiver, optimization processor, and an action plan applier. The infrastructure optimization service framework enables a way for a wide range of cloud optimization goals, including the decrease of data center operating costs, improved system performance via intelligent virtual machine migration, and increased energy efficiency.

To know more about all the optional projects available and the maturity age, visit `https://www.openstack.org/software/project-navigator`.

Summary

In this chapter, we have seen an overview of commonly adopted optional OpenStack services. New projects may get added to the OpenStack Umbrella during every six-month release cycle. The OpenStack project is at a size now where no one can truly know the details of each service. By now, you will have gained an idea of what each optional service can do with OpenStack.

Having looked at the bells and whistles of OpenStack, we have come to the end of this book. The course outline covered in this book is the core concept of OpenStack, which includes all the fundamental methodologies that are baked into the OpenStack project. New projects and features may get added to OpenStack quickly in the upcoming release cycle, but because the information here is central to a base installation of OpenStack, you should be able to reference this book for many releases to come from the OpenStack project.

Index