Kevin Jackson, Cody Bunch, Egle Sigler, James Denton

OpenStack Cloud Computing Cookbook

Fourth Edition

Over 100 practical recipes to help you build and operate OpenStack cloud computing, storage, networking, and automation

Packt

OpenStack Cloud Computing Cookbook

Fourth Edition

Over 100 practical recipes to help you build and operate OpenStack cloud computing, storage, networking, and automation

Kevin Jackson

Cody Bunch

Egle Sigler

James Denton

Packt>

BIRMINGHAM - MUMBAI

OpenStack Cloud Computing Cookbook

Fourth Edition

Commissioning Editor: Dominic Shakeshaft
Acquisition Editor: Dominic Shakeshaft
Project Editor: Radhika Atitkar
Technical Editor: Nidhisha Shetty
Proofreader: Tom Jacob
Indexer: Pratik Shirodkar
Graphics: Tania Dutta
Production Coordinator: Arvindkumar Gupta

First published: September 2012

Second edition: October 2013

Third edition: August 2015

Fourth edition: January 2018

Production reference: 2230118

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

ISBN 978-1-78839-876-3

www.packtpub.com

To my mum, who is stronger than cancer

– Kevin Jackson

To Kevin's mum, who is stronger than cancer

– Cody Bunch

To Kevin's mum, who is stronger than cancer

– Egle Sigler

mapt.io

Mapt is an online digital library that gives you full access to over 5,000 books and videos as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Learn better with Skill Plans built especially for you
- Get a free eBook or video every month
- Mapt is fully searchable
- Copy and paste, print, and bookmark content

PacktPub.com

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com, and as a print book customer, you are entitled to a discount on the eBook copy. For more details, get in touch with us at service@packtpub.com.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Contributors

About the authors

Kevin Jackson is married and has three children. He is an IT professional with over 20 years of experience, working with businesses and enterprises of all sizes at Rackspace, as an OpenStack and private cloud specialist. Kevin has been working with OpenStack since the first release, and he has extensive experience of various flavors of Linux, Unix, and hosting environments. Kevin authored the first edition of this book, and coauthored the second and third editions. Kevin also coauthored *OpenStack Architecture Design Guide*, *OpenStack Foundation*, during a 5-day book sprint in California. He also regularly speaks at OpenStack community events.

Cody Bunch is a principal architect in the Rackspace Private Cloud group, based out of San Antonio, Texas. Cody has been working with OpenStack since early 2012, coauthored the second edition of this book, and also coauthored *OpenStack Security Guide*, *OpenStack Foundation*. Cody has extensive experience with virtualized and cloud environments in variously sized enterprises and hosting environments.

Egle Sigler is an OpenStack Foundation board member and a principal architect in the Rackspace Private Cloud group, based out of San Antonio, Texas. Egle holds an MS degree in computer science. She started her career as a software developer and still has a soft spot for all the people who write, test, and deploy code, since she has had the chance to do all of these tasks throughout her career. Egle dreams about a day when writing, testing, and deploying code will be a seamless and easy process—bug and frustration free for all. Egle believes that knowledge should be shared and has tried to do this by writing this book, giving talks and workshops at conferences, and blogging.

I would like to thank my husband and daughter for their unwavering support and love. Roy and Elena, you make everything better! I ask for forgiveness from my friends and family, who didn't get to talk to me very much while I was working on this book.

OpenStack developers, quality engineers, operators, users, and documentation writers, thank you all for making OpenStack better each day!

Kevin, Cody, and James, thank you for bringing me along on this adventure! I cannot believe how much quality work was put into this book.

Technical reviewers, thank you for volunteering hundreds of hours to review everything.

Reviewers and editors from Packt, thank you for your prompt communication and constant feedback. Rackers, thank you for your advice and guidance. Lastly, thanks to Rackspace for supporting my writing endeavors.

James Denton is a principal architect at Rackspace with over 15 years of experience in systems administration and networking. He has a bachelor's degree in Business Management with a focus in Computer Information Systems from Texas State University in San Marcos, Texas. He is currently focused on OpenStack operations and support within the Rackspace Private Cloud team. James is the author of the *Learning OpenStack Networking (Neutron)*, first and second editions, as well as *OpenStack Networking Essentials* both by *Packt Publishing*.

I'd like to thank my wife, Amanda, for providing time, encouragement, and patience throughout the writing of this book. I would also like to thank my team at Rackspace for providing valuable feedback and experience.

Without NASA, Rackspace, and countless other contributors to OpenStack, none of this would have been possible. To the developers, operators, and users of OpenStack I say, "Thank you!"

About the reviewers

Christian Ashby has a broad range of experience working with the architecture of Cloud environments - with every definition from "someone else's data center" to serverless / no-ops public cloud and PaaS environments. With a background in large IT outsourcing designing scale-up compute and storage solutions, Christian has been focusing more and more on Cloud architectures and assisting his customers find their route to the Cloud and modern IT practices. OpenStack has always been one of the components of this customer journey, and Christian is pleased to see it move deeper into the Enterprise space with later releases, and hopes that this trend continues.

I would like to thank *Kevin Jackson* for recommending me to review the book; it's been a good experience and it's been great working with the team.

Stefano Canepa is an information and communication technology professional with more then 20 years of various experiences. Stefano's main interest now is cloud computing with a special focus on OpenStack, its management, and centralized logging solutions. He started working as system administrator in one of the first internet service providers established in Italy. At that time, it was natural to install applications you needed from source code, so he learnt programming on Unix. He started looking after Windows systems and, at his new company, Stefano was involved in the deployment and administration of a huge network of Windows NT computers spread all over Italy. He continued his career moving to network design and testing. He then turned into a full time software developer, and for a while Stefano wrote applications to control medical devices. He moved to a new company and switched to write railways' centralized traffic control GUI applications. After that experience, Stefano was hired as the system software engineer to manage clouds in a team of professionals coming from all over the world.

Ricky Donato is an NFV/SDN solutions architect, with a passion for open source and an unhealthy love for Costa coffee. In his 23 years in the industry, his technical experience has covered networking design, development, network security, and OpenStack.

Geoff Higginbottom is an OpenStack specialist architect at Rackspace, where he is responsible for designing OpenStack Clouds for customers. Prior to Rackspace, Geoff was the technical cofounder, CTO, and cloud architect at a boutique cloud consultancy, where he designed numerous public and private cloud deployments. Geoff's technical career started with real-time computer systems while serving in the Royal Navy as a computer engineer in the 90s, and it has now spanned to almost every facet of IT. Geoff is a keen sailor, scuba diver, motorcyclist, and amateur photographer, and his ambition is to one day sail around the world!

Andrew McCrae works as a software developer at Rackspace, in the Rackspace Private Cloud team, working specifically on Ceph storage solutions. He was the project technical lead for the OpenStack-Ansible project for the Ocata and Pike cycles. Andy began his career in 2007 at Rackspace, as a Linux systems administrator, after completing a masters in Engineering, majoring Computer Science at **University College London** (**UCL**). Andy specializes in distributed storage, specifically Swift (OpenStack Object Storage), and Ceph, as well as in deployment automation using tools such as Ansible and Chef. He was previously a core reviewer on the Chef-OpenStack project. Andy was also a reviewer on the third edition of *OpenStack Cloud Computing Cookbook*, *Packt Publishing*.

Wojciech Sciesinski is an IT professional with 15 years of experience in a broad range of technologies, namely Microsoft Windows, Microsoft Exchange Server, PowerShell, Linux, automation, virtualization, and cloud-related tools, and roles, particularly, a support engineer, architect, and developer. Wojciech is curious about new things, especially when they are open source and complicated enough to not be boring.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, helping them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Preface

OpenStack is the open source software for building public and private clouds, and privately hosted software-defined infrastructure services. It is a global open source success and is developed and supported by thousands of people around the globe and backed by leading players in the cloud space today. This book is specifically designed to quickly help you get up to speed with OpenStack and give you the confidence and understanding to roll it out into your own datacenters. This book covers an installation of OpenStack using Ansible, into your datacenters, in addition to providing steps for running under so that you can quickly gain the knowledge needed to install and operate OpenStack today. This book has been written by four principal level OpenStack engineers and architects from Rackspace, and it covers a wide range of topics that help you install and configure your OpenStack environments. This book will guide you in the following things:

- How to install and configure all of the core components of OpenStack using OpenStack-Ansible
- How to master the complete private cloud stack, from scaling out compute resources to managing object storage services for highly redundant, highly available storage
- Examples of using Heat and Ansible to orchestrate workloads running on OpenStack
- Practical, real-world examples of each service built upon in each chapter, allowing you to progress with the confidence that they will work in your own environments

This book gives you clear, step-by-step instructions to install and run your own private cloud successfully. It is full of practical and applicable recipes that enable you to use the latest capabilities of OpenStack and implement them.

Who this book is for

This book is aimed at system administrators and technical architects moving from a virtualized environment to cloud environments who are familiar with cloud computing platforms. Knowledge of virtualization and managing Linux environments is expected. Prior knowledge or experience of OpenStack is not required, although beneficial.

What this book covers

Chapter 1, *Installing OpenStack with Ansible*, walks you through a practical installation of OpenStack using OpenStack-Ansible.

Chapter 2, *The OpenStack Client*, shows you how to install the tools needed to operate OpenStack, as well as providing a quick reference to commonly used commands.

Chapter 3, *Keystone – OpenStack Identity Service*, explains how to use Keystone to configure and maintain services and projects within OpenStack.

Chapter 4, *Neutron – OpenStack Networking*, covers the introduction and use of the software-defined networking service and its plugins.

Chapter 5, *Nova – OpenStack Compute*, works through a number of examples of running and operating virtual machine instances to run your applications.

Chapter 6, *Glance – OpenStack Image Service*, explores the use of the machine images that are used to spawn instances.

Chapter 7, *Cinder – OpenStack Block Storage*, presents how to work with block storage volumes to attach to your instances.

Chapter 8, *Swift – OpenStack Object Storage*, depicts how to use the highly available and redundant object storage service.

Chapter 9, *OpenStack Orchestration Using Heat and Ansible*, discusses how to begin orchestrating your workloads within OpenStack using both Heat and Ansible.

Chapter 10, *Using OpenStack Dashboard*, shows you how to navigate the Horizon user-interface dashboard.

To get the most out of this book

To use this book, you will need access to computers or servers that have hardware virtualization capabilities. Familiarity of Linux and accessing Linux servers using SSH is expected.

To set up the lab environment described at the end of *Chapter 1*, *Installing OpenStack with Ansible*, you will need to install and use Oracle's VirtualBox and Vagrant. You can access details of how to set up your computer using VirtualBox and Vagrant by visiting `https://github.com/OpenStackCookbook/vagrant-openstack`.

There are additional recipes to get you started with the lab environment available at `http://www.openstackcookbook.com`. Refer to this website for information on installation of supporting software such as MariaDB/MySQL. More information can be found at `http://bit.ly/OpenStackCookbookPreReqs`.

Download the example code files

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 emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `http://www.packtpub.com`.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the on-screen instructions.

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

- 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-Cloud-Computing-Cookbook-Fourth-Edition`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://www.packtpub.com/sites/default/files/downloads/OpenStackCloudComputingCookbookFourthEdition_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: Configuration of the host's networking, on a Ubuntu system, is performed by editing the `/etc/network/interfaces` file.A block of code is set as follows:

```
# Shared infrastructure parts
shared-infra_hosts:
  controller-01:
    ip: 172.29.236.110
  controller-02:
    ip: 172.29.236.111
  controller-03:
    ip: 172.29.236.112
# Compute Hosts
compute_hosts:
  compute-01:
    ip: 172.29.236.113
  compute-02:
    ip: 172.29.236.114
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
# Shared infrastructure parts
shared-infra_hosts:
  controller-01:
    ip: 172.29.236.110
  controller-02:
    ip: 172.29.236.111
  controller-03:
    ip: 172.29.236.112
# Compute Hosts
compute_hosts:
  compute-01:
    ip: 172.29.236.113
  compute-02:
    ip: 172.29.236.114
```

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

```
cd /opt/openstack-ansible/scripts
```

Bold: Indicates a new term, an important word, or words that you see on the screen, for example, in menus or dialog boxes, also appear in the text like this. Here is an example: "Next choose **Advanced system settings** from the menu on the left."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Sections

In this book, you will find several headings that appear frequently (*Getting ready*, *How to do it...*, *How it works...*, *There's more...*, and *See also*).

To give clear instructions on how to complete a recipe, use these sections as follows:

Getting ready

This section tells you what to expect in the recipe and describes how to set up any software or any preliminary settings required for the recipe.

How to do it...

This section contains the steps required to follow the recipe.

How it works...

This section usually consists of a detailed explanation of what happened in the previous section.

There's more...

This section consists of additional information about the recipe in order to make you more knowledgeable about the recipe.

See also

This section provides helpful links to other useful information for the recipe.

Get in touch

Feedback from our readers is always welcome.

General feedback: Email `feedback@packtpub.com` and mention the book's title in the subject of your message. If you have questions about any aspect of this book, please email us at `questions@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book we would be grateful if you would report this to us. Please visit, `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packtpub.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `http://authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packtpub.com`.

1
Installing OpenStack with Ansible

In this chapter, we will cover the following topics:

- Introduction – the OpenStack architecture
- Host network configuration
- Root SSH keys configuration
- Installing Ansible, playbooks, and dependencies
- Configuring the installation
- Running the OpenStack-Ansible playbooks
- Troubleshooting the installation
- Manually testing the installation
- Modifying the OpenStack configuration
- Virtual lab - vagrant up!

Introduction – the OpenStack architecture

OpenStack is a suite of projects that combine into a software-defined environment to be consumed using cloud friendly tools and techniques. The popular open source software allows users to easily consume compute, network, and storage resources that have been traditionally controlled by disparate methods and tools by various teams in IT departments, big and small. While consistency of APIs can be achieved between versions of OpenStack, an administrator is free to choose which features of OpenStack to install, and as such there is no single method or architecture to install the software. This flexibility can lead to confusion when choosing how to deploy OpenStack. That said, it is universally agreed that the services that the end users interact with—the OpenStack services, supporting software (such as the databases), and APIs—must be highly available.

A very popular method for installing OpenStack is the OpenStack-Ansible project (`https://github.com/openstack/openstack-ansible`). This method of installation allows an administrator to define highly available controllers together with arrays of compute and storage, and through the use of Ansible, deploy OpenStack in a very consistent way with a small amount of dependencies. Ansible is a tool that allows for system configuration and management that operates over standard SSH connections. Ansible itself has very few dependencies, and as it uses SSH to communicate, most Linux distributions and networks are well-catered for when it comes to using this tool. It is also very popular with many system administrators around the globe, so installing OpenStack on top of what they already know lowers the barrier to entry for setting up a cloud environment for their enterprise users.

OpenStack can be architected in any number of ways; OpenStack-Ansible doesn't address the architecture problem directly: users are free to define any number of controller services (such as Horizon, Neutron Server, Nova Server, and MySQL). Through experience at Rackspace and feedback from users, a popular architecture is defined, which is shown here:

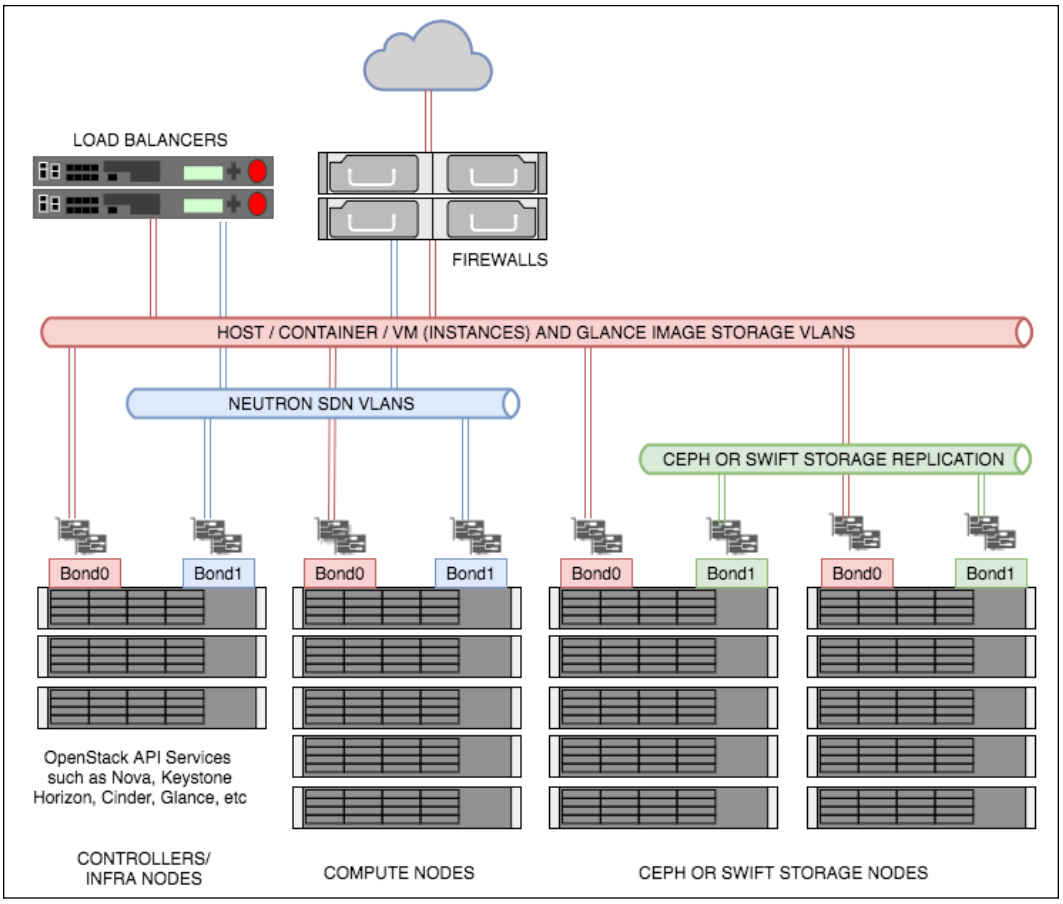


Figure 1: Recommended OpenStack architecture used in this book

As shown in the preceding diagram (*Figure 1*), there are a few concepts to first understand. These are described as follows.

Controllers

The *controllers* (also referred to as *infrastructure or infra nodes*) run the heart of the OpenStack services and are the only servers exposed (via load balanced pools) to your end users. The *controllers* run the API services, such as Nova API, Keystone API, and Neutron API, as well as the core supporting services such as **MariaDB** for the database required to run OpenStack, and RabbitMQ for messaging. It is this reason why, in a production setting, these servers are set up as highly available as required. This means that these are deployed as clusters behind (highly available) load balancers, starting with a minimum of 3 in the cluster. Using odd numbers starting from 3 allows clusters to lose a single server without and affecting service and still remain with quorum (minimum numbers of votes needed). This means that when the unhealthy server comes back online, the data can be replicated from the remaining 2 servers (which are, between them, consistent), thus ensuring data consistency.

Networking is recommended to be highly resilient, so ensure that Linux has been configured to bond or aggregate the network interfaces so that in the event of a faulty switch port, or broken cable, your services remain available. An example networking configuration for Ubuntu can be found in *Appendix A*.

Computes

These are the servers that run the hypervisor or container service that OpenStack schedules workloads to when a user requests a Nova resource (such as a virtual machine). These are not too dissimilar to hosts running a hypervisor, such as ESXi or Hyper-V, and OpenStack Compute servers can be configured in a very similar way, optionally using shared storage. However, most installations of OpenStack forgo the need for the use of shared storage in the architecture. This small detail of not using shared storage, which implies the virtual machines run from the hard disks of the compute host itself, can have a large impact on the users of your OpenStack environment when it comes to discussing the resiliency of the applications in that environment. An environment set up like this pushes most of the responsibility for application uptime to developers, which gives the greatest flexibility of a long-term cloud strategy. When an application relies on the underlying infrastructure to be 100% available, the gravity imposed by the infrastructure ties applications to specific data center technology to keep it running. However, OpenStack can be configured to introduce shared storage such as Ceph (`http://ceph.com/`) to allow for operational features such as live-migration (the ability to move running instances from one hypervisor to another with no downtime), allowing enterprise users move their applications to a cloud environment in a very safe way. These concepts will be discussed in more detail in later chapters on compute and storage. As such, the reference architecture for a compute node is to expect virtual machines to run locally on the hard drives in the server itself.

With regard to networking, like the *controllers*, the network must also be configured to be highly available. A compute node that has no network available might be very secure, but it would be equally useless to a cloud environment! Configure bonded interfaces in the same way as the controllers. Further information for configuring bonded interfaces under Ubuntu can be found in *Appendix A*.

Storage

Storage in OpenStack refers to block storage and object storage. Block storage (providing LUNs or *hard drives* to virtual machines) is provided by the Cinder service, while object storage (API driven object or blobs of storage) is provided by Swift or Ceph. Swift and Ceph manage each individual drive in a server designated as an object storage node, very much like a RAID card manages individual drives in a typical server. Each drive is an independent entity that Swift or Ceph uses to write data to. For example, if a storage node has 24 x 2.5in SAS disks in, Swift or Ceph will be configured to write to any one of those 24 disks. Cinder, however, can use a multitude of backends to store data. For example, Cinder can be configured to communicate with third-party vendors such as NetApp or Solidfire arrays, or it can be configured to talk to Sheepdog or Ceph, as well as the simplest of services such as LVM. In fact, OpenStack can be configured in such a way that Cinder uses multiple backends so that a user is able to choose the storage applicable to the service they require. This gives great flexibility to both end users and operators as it means workloads can be targeted at specific backends suitable for that workload or storage requirement.

This book briefly covers Ceph as the backend storage engine for Cinder. Ceph is a very popular, highly available open source storage service. Ceph has its own disk requirements to give the best performance. Each of the Ceph storage nodes in the preceding diagram are referred to as **Ceph OSDs** (**Ceph Object Storage Daemons**). We recommend starting with 5 of these nodes, although this is not a hard and fast rule. Performance tuning of Ceph is beyond the scope of this book, but at a minimum, we would highly recommend having SSDs for Ceph journaling and either SSD or SAS drives for the OSDs (the physical storage units).

The differences between a Swift node and a Ceph node in this architecture are very minimal. Both require an interface (bonded for resilience) for replication of data in the storage cluster, as well as an interface (bonded for resilience) used for data reads and writes from the client or service consuming the storage.

The primary difference is the recommendation to use SSDs (or NVMe) as the journaling disks.

Load balancing

The end users of the OpenStack environment expect services to be highly available, and OpenStack provides REST API services to all of its features. This makes the REST API services very suitable for placing behind a load balancer. In most deployments, load balancers would usually be highly available hardware appliances such as F5. For the purpose of this book, we will be using HAProxy. The premise behind this is the same though—to ensure that the services are available so your end users can continue working in the event of a failed *controller* node.

OpenStack-Ansible installation requirements

Operating installing System: Ubuntu 16.04 x86_64

Minimal data center deployment requirements

For a physical installation, the following will be needed:

- Controller servers (also known as infrastructure nodes)
  - At least 64 GB RAM
  - At least 300 GB disk (RAID)
  - 4 Network Interface Cards (for creating two sets of bonded interfaces; one would be used for infrastructure and all API communication, including client, and the other would be dedicated to OpenStack networking: Neutron)
  - Shared storage, or object storage service, to provide backend storage for the base OS images used
- Compute servers
  - At least 64 GB RAM
  - At least 600 GB disk (RAID)
  - 4 Network Interface Cards (for creating two sets of bonded interfaces, used in the same way as the controller servers)
- Optional (if using Ceph for Cinder) 5 Ceph Servers (Ceph OSD nodes)
  - At least 64 GB RAM
  - 2 x SSD (RAID1) 400 GB for journaling
  - 8 x SAS or SSD 300 GB (No RAID) for OSD (size up requirements and adjust accordingly)
  - 4 Network Interface Cards (for creating two sets of bonded interfaces; one for replication and the other for data transfer in and out of Ceph)

- Optional (if using Swift) 5 Swift Servers
  - At least 64 GB RAM
  - 8 x SAS 300 GB (No RAID) (size up requirements and adjust accordingly)
  - 4 Network Interface Cards (for creating two sets of bonded interfaces; one for replication and the other for data transfer in and out of Swift)
- Load balancers
  - 2 physical load balancers configured as a pair
  - Or 2 servers running HAProxy with a Keepalived VIP to provide as the API endpoint IP address:
    - At least 16 GB RAM
    - HAProxy + Keepalived
    - 2 Network Interface Cards (bonded)

Tip: Setting up a physical home lab? Ensure you have a managed switch so that interfaces can have VLANs tagged.

Host network configuration

Installation of OpenStack using an orchestration and configuration tool such as Ansible performs a lot of tasks that would otherwise have to be undertaken manually. However, we can only use an orchestration tool if the servers we are deploying to are configured in a consistent way and described to Ansible.

The following section will describe a typical server setup that uses two sets of active/passive bonded interfaces for use by OpenStack. Ensure that these are cabled appropriately.

We assume that the following physical network cards are installed in each of the servers; adjust them to suit your environment:

- `p2p1` and `p2p2`
- `p4p1` and `p4p2`

We assume that the *host* network is currently using `p2p1`. The *host* network is the basic network that each of the servers currently resides on, and it allows you to access each one over SSH. It is assumed that this network also has a default gateway configured, and allows internet access. There should be no other networks required at this point as the servers are currently unconfigured and are not running OpenStack services.

At the end of this section, we will have created the following bonded interfaces:

- `bond0`: This consists of the physical interfaces `p2p1` and `p4p1`. The `bond0` interface will be used for host, OpenStack management, and storage traffic.
- `bond1`: This consists of the physical interfaces `p2p2` and `p4p2`. The `bond1` interface will be used for Neutron networking within OpenStack.

We will have created the following VLAN tagged interfaces:

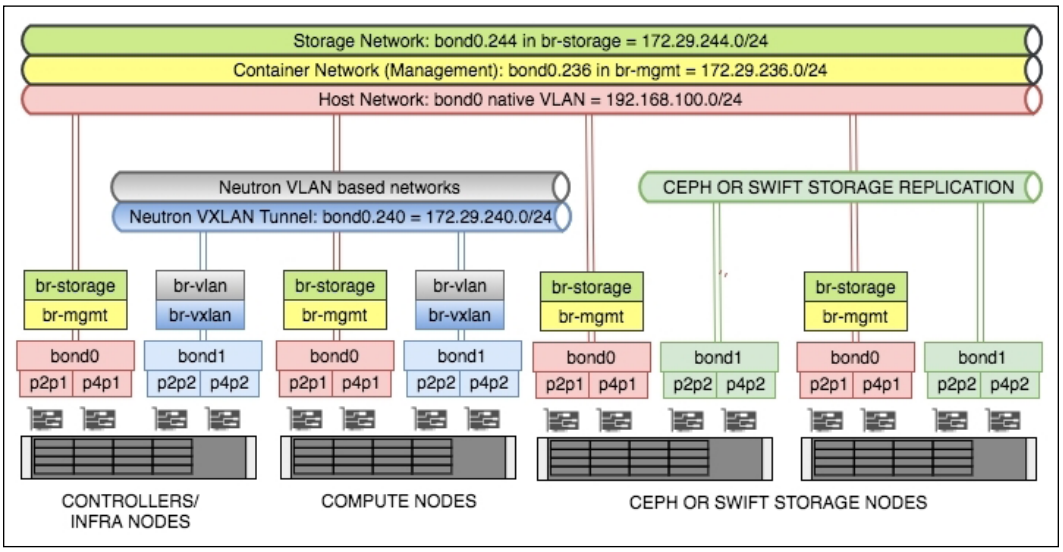
- `bond0.236`: This will be used for the *container network*
- `bond0.244`: This will be used for the *storage network*
- `bond1.240`: This will be used for the *VXLAN tunnel network*

And the following bridges:

- `br-mgmt`: This will use the `bond0.236` VLAN interface, and will be configured with an IP address from the `172.29.236.0/24` range.
- `br-storage`: This will use the `bond0.244` VLAN interface, and will be configured with an IP address from the `172.29.244.0/24` range.
- `br-vxlan`: This will use the `bond1.240 VLAN` interface, and will be configured with an IP address from the `172.29.240.0/24` range.
- `br-vlan`: This will use the untagged `bond1` interface, and will not have an IP address configured.

Tip: Ensure that your subnets are large enough to support your current requirements as well as future growth!

The following diagram shows the networks, interfaces, and bridges set up before we begin our installation of OpenStack:



Getting ready

We assume that each server has Ubuntu 16.04 installed.

Log in, as root, onto each server that will have OpenStack installed.

How to do it...

Configuration of the host's networking, on a Ubuntu system, is performed by editing the `/etc/network/interfaces` file.

Tip: We will be moving the host network IP from a single interface to the bonded interface, `bond0`. When we follow the steps given here, it is possible that you will lose network connectivity when the interfaces get restarted. As such, it is recommended that you make modifications to the networking through an out-of-band interface, such as iLO or DRAC, or some other KVM-based (Keyboard, Video, Monitor) system (not to be confused with the hypervisor technology by the same acronym).

1. First of all, ensure that we have the right network packages installed on each server. As we are using VLANs and Bridges, the following packages must be installed:

```
apt update
apt install vlan bridge-utils
```

2. Now edit the `/etc/network/interfaces` file on the first server using your preferred editor:

```
vi /etc/network/interfaces
```

3. We will first configure the bonded interfaces. The first part of the file will describe this. Edit this file so that it looks like the following to begin with:

```
# p2p1 + p4p1 = bond0 (used for host, container and
storage)
auto p2p1
iface p2p1 inet manual
  bond-master bond0
  bond-primary p2p1
auto p4p1
iface p4p1 inet manual
  bond-master bond0
# p2p2 + p4p2 = bond1 (used for Neutron and Storage
Replication)
auto p2p2
iface p2p2 inet manual
  bond-master bond1
  bond-primary p2p2
auto p4p2
iface p4p2 inet manual
  bond-master bond1
```

4. Now we will configure the VLAN interfaces that are tagged against these bonds. Continue editing the file to add in the following tagged interfaces. Note that we are not assigning IP addresses to the OpenStack bonds just yet:

```
# We're using bond0 on a native VLAN for the 'host'
network.
# This bonded interface is likely to replace the
address you
# are currently using to connect to this host.
auto bond0
iface bond0 inet static
  address 192.168.100.11
  netmask 255.255.255.0
  gateway 192.168.100.1
  dns-nameserver 192.168.100.1 # Update to suit/ensure
you can resolve DNS
auto bond0.236  # Container VLAN
iface bond0.236  inet manual
```

```
auto bond1.240  # VXLAN Tunnel VLAN
iface bond1.240  inet manual
auto bond0.244  # Storage (Instance to Storage) VLAN
iface bond0.244  inet manual
```

Tip: Use appropriate VLANs as required in your own environment. The VLAN tags used here are for reference only.

Ensure that the correct VLAN tag is configured against the correct bonded interface. `bond0` is for host-type traffic, `bond1` is predominantly for Neutron-based traffic, except for storage nodes, where it is then used for storage replication.

5. We will now create the bridges, and place IP addresses on here as necessary (note that `br-vlan` does not have an IP address assigned). Continue editing the same file and add in the following lines:

```
# Container bridge (br-mgmt)
auto br-mgmt
iface br-mgmt inet static
  address 172.29.236.11
  netmask 255.255.255.0
  bridge_ports bond0.236
  bridge_stp off
# Neutron's VXLAN bridge (br-vxlan)
auto br-vxlan
iface br-vxlan inet static
  address 172.29.240.11
  netmask 255.255.255.0
  bridge_ports bond1.240
  bridge_stp off
# Neutron's VLAN bridge (br-vlan)
auto br-vlan
iface br-vlan inet manual
  bridge_ports bond1
  bridge_stp off
# Storage Bridge (br-storage)
auto br-storage
iface br-storage inet static
  address 172.29.244.11
  netmask 255.255.255.0
  bridge_ports bond0.244
  bridge_stp off
```

These bridge names are referenced in the OpenStack-Ansible configuration file, so ensure you name them correctly.

Be careful in ensuring that the correct bridge is assigned to the correct bonded interface.

6. Save and exit the file, then issue the following command:

```
restart networking
```

7. As we are configuring our OpenStack environment to be as highly available as possible, it is suggested that you also reboot your server at this point to ensure the basic server, with redundant networking in place, comes back up as expected:

```
reboot
```

8. Now repeat this for each server on your network.

9. Once all the servers are done, ensure that your servers can communicate with each other over these newly created interfaces and subnets. A test like the following might be convenient:

```
apt install fping
fping -a -g 172.29.236.0/24
fping -a -g 172.29.240.0/24
fping -a -g 172.29.244.0/24
```

Tip: We also recommend that you perform a network cable unplugging exercise to ensure that the failover from one active interface to another is working as expected.

How it works...

We have configured the physical networking of our hosts to ensure a good known state and configuration for running OpenStack. Each of the interfaces configured here is specific to OpenStack—either directly managed by OpenStack (for example, `br-vlan`) or used for inter-service communication (for example, `br-mgmt`). In the former case, OpenStack utilizes the `br-vlan` bridge and configures tagged interfaces on `bond1` directly.

Note that the convention used here, of VLAN tag ID using a portion of the subnet, is only to highlight a separation of VLANs to specific subnets (for example, `bond0.236` is used by the `172.29.236.0/24` subnet). This VLAN tag ID is arbitrary, but must be set up in accordance with your specific networking requirements.

Finally, we performed a fairly rudimentary test of the network. This gives you the confidence that the network configuration that will be used throughout the life of your OpenStack cloud is fit for purpose and gives assurances in the event of a failure of a cable or network card.

Root SSH keys configuration

Ansible is designed to help system administrators drive greater efficiency in the datacenter by being able to configure and operate many servers using orchestration playbooks. In order for Ansible to be able to fulfill its duties, it needs an SSH connection on the Linux systems it is managing. Furthermore, in order to have a greater degree of freedom and flexibility, a hands-off approach using SSH public private key pairs is required.

As the installation of OpenStack is expected to run as root, this stage expects the deployment host's root public key to be propagated across all servers.

Getting ready

Ensure that you are `root` on the deployment host. In most cases, this is the first infrastructure *controller* node that we have named for the purposes of this book to be called `infra01`. We will be assuming that all Ansible commands will be run from this host, and that it expects to be able to connect to the rest of the servers on this network over the host network via SSH.

How to do it...

In order to allow a hands-free, orchestrated OpenStack-Ansible deployment, follow these steps to create and propagate root SSH public key of `infra01` across all servers required of the installation:

1. As root, execute the following command to create an SSH key pair:

   ```
   ssh-keygen -t rsa -f ~/.ssh/id_rsa -N ""
   ```

   The output should look similar to this:

   ```
   Generating public/private rsa key pair.
   Your identification has been saved in /root/.ssh/id_rsa.
   Your public key has been saved in /root/.ssh/id_rsa.pub.
   The key fingerprint is:
   SHA256:q0mdqJI3TTFaiLrMaPABBboTsyr3pRnCaylLU5WEDCw root@infra01
   The key's randomart image is:
   +---[RSA 2048]----+
   |ooo ..           |
   ```

```
|E..o. .          |
|=. . +           |
|.=. o +          |
|+o . o oS        |
|+oo . .o        |
|B=++.o+ +        |
|*=B+oB.o         |
|o+.o=.o          |
+----[SHA256]-----+
```

2. This has created two files in `/root/.ssh`, called `id_rsa` and `id_rsa.pub`. The file, `id_rsa` is the private key, and must not be copied across the network. It is not required to be anywhere other than on this server. The file, `id_rsa.pub`, is the public key and can be shared to other servers on the network. If you have other nodes (for example, named infra02), use the following to copy this key to that node in your environment:

   ```
   ssh-copy-id root@infra02
   ```

Tip: Ensure that you can resolve `infra02` and the other servers, else amend the preceding command to use its host IP address instead.

3. Now repeat step 2 for all servers on your network.
4. Important: finally, ensure that you execute the following command to be able to SSH to itself:

   ```
   ssh-copy-id root@infra01
   ```

5. Test that you can `ssh`, as the root user, from `infra01` to other servers on your network. You should be presented with a Terminal ready to accept commands if successful, without being prompted for a passphrase. Consult `/var/log/auth.log` on the remote server if this behavior is incorrect.

How it works...

We first generated a key pair file for use by SSH. The `-t` option specified the `rsa` type encryption, `-f` specified the output of the private key, where the public portion will get `.pub` appended to its name, and `-N ""` specified that no passphrase is to be used on this key. Consult your own security standards if the presented options differ from your company's requirements.

Installing Ansible, playbooks, and dependencies

In order for us to successfully install OpenStack using Ansible, we need to ensure that Ansible and any expected dependencies are installed on the deployment host. The OpenStack-Ansible project provides a handy script to do this for us, which is part of the version of OpenStack-Ansible we will be deploying.

Getting ready

Ensure that you are `root` on the deployment host. In most cases, this is the first infrastructure controller node, `infra01`.

At this point, we will be checking out the version of OpenStack-Ansible from GitHub.

How to do it...

To set up Ansible and its dependencies, follow these steps:

1. We first need to use `git` to check out the OpenStack-Ansible code from GitHub, so ensure that the following packages are installed (among other needed dependencies):

   ```
   apt update

   apt install git python-dev bridge-tools lsof lvm2 tcpdump build-
   essential ntp ntpdate python-dev libyaml-dev libpython2.7-dev
   libffi-dev libssl-dev python-crypto python-yaml
   ```

2. We then need to grab the OpenStack-Ansible code from GitHub. At the time of writing, the Pike release branch (16.X) is described as follows, but the steps remain the same for the foreseeable future. It is recommended that you use the latest stable tag by visiting `https://github.com/openstack/openstack-ansible/tags`. Here we're using the latest 16 (Pike) tag denoted by `16.0.5`:

   Tip: To use a branch of the Queens release, use the following: `-b 17.0.0`. When the Rocky release is available, use `-b 18.0.0`.

   ```
   git clone -b 16.0.5 https://github.com/openstack/openstack-
   ansible.git /opt/openstack-ansible
   ```

3. Ansible and the needed dependencies to successfully install OpenStack can be found in the `/opt/openstack-ansible/scripts` directory. Issue the following command to bootstrap the environment:

```
cd /opt/openstack-ansible
scripts/bootstrap-ansible.sh
```

How it works...

The OpenStack-Ansible project provides a handy script to ensure that Ansible and the right dependencies are installed on the deployment host. This script (`bootstrap-ansible.sh`) lives in the `scripts/` directory of the checked out OpenStack-Ansible code, so at this stage we need to grab the version we want to deploy using Git. Once we have the code, we can execute the script and wait for it to complete.

There's more...

Visit `https://docs.openstack.org/project-deploy-guide/openstack-ansible/latest` for more information.

Configuring the installation

OpenStack-Ansible is a set of official Ansible playbooks and roles that lay down OpenStack with minimal prerequisites. Like any orchestration tool, most effort is done up front with configuration, followed by a hands-free experience when the playbooks are running. The result is a tried and tested OpenStack installation suitable for any size environment, from testing to production environments.

When we use OpenStack-Ansible, we are basically downloading the playbooks from GitHub onto a nominated *deployment server*. A deployment server is the host that has access to all the machines in the environment via SSH (and for convenience, and for the most seamless experience without hiccups, via keys). This deployment server can be one of the machines you've nominated as a part of your OpenStack environment as Ansible isn't anything that takes up any ongoing resources once run.

Tip: Remember to back up the relevant configuration directories related to OpenStack-Ansible before you rekick an install of Ubuntu on this server!

Getting ready

Ensure that you are `root` on the *deployment host*. In most cases, this is the first infrastructure controller node, `infra01`.

How to do it...

Let's assume that you're using the first infrastructure node, `infra01`, as the deployment server.

If you have not followed the preceding *Installing Ansible, playbooks, and dependencies* recipe review, then as `root`, carry out the following if necessary:

```
git clone -b 16.05 https://github.com/openstack/openstack-ansible.git /
opt/openstack-ansible
```

This downloads the OpenStack-Ansible playbooks to the `/opt/openstack-ansible` directory.

To configure our OpenStack deployment, carry out the following steps:

1. We first copy the `etc/openstack_deploy` folder out of the downloaded repository to `/etc`:

   ```
   cd /opt/openstack-ansible
   cp -R /etc/openstack_deploy /etc
   ```

2. We now have to tell Ansible which servers will do which OpenStack function, by editing the `/etc/openstack_deploy/openstack_user_config.yml` file, as shown here:

   ```
   cp /etc/openstack_deploy/openstack_user_variables.yml.example /
   etc/openstack_deploy_openstack_user_variables.yml
   vi /etc/openstack_deploy/openstack_user_variables.yml
   ```

3. The first section, `cidr_networks`, describes the subnets used by OpenStack in this installation. Here we describe the *container* network (each of the OpenStack services are run inside a container, and this has its own network so that each service can communicate with each other). We describe the tunnel network (when a user creates a tenant network in this installation of OpenStack, this will create a segregated VXLAN network over this physical network). Finally, we describe the storage network subnet. Edit this file so that it looks like the following:

   ```
   cidr_networks:
   container: 172.29.236.0/24
   tunnel: 172.29.240.0/24
   storage: 172.29.244.0/24
   ```

4. Continue editing the file to include any IP addresses that are already used by existing physical hosts in the environment where OpenStack will be deployed (and ensuring that you've included any reserved IP addresses for physical growth too). Include the addresses we have already configured leading up to this section. Single IP addresses or ranges (start and end placed either side of a ',') can be placed here. Edit this section to look like the following, adjust as per your environment and any reserved IPs:

```
used_ips:
  - "172.29.236.20"
  - "172.29.240.20"
  - "172.29.244.20"
  - "172.29.236.101,172.29.236.117"
  - "172.29.240.101,172.29.240.117"
  - "172.29.244.101,172.29.244.117"
  - "172.29.248.101,172.29.248.117"
```

5. The `global_overrides` section describes the bridges and other specific details of the interfaces used environment—particularly pertaining to how the container network attaches to the physical network interfaces. For the example architecture used in this book, the following output can be used. In most cases, the content in this section doesn't need to be edited apart from the load balancer information at the start, so edit to suit:

```
global_overrides:
  internal_lb_vip_address: 172.29.236.117
  external_lb_vip_address: 192.168.100.117
  lb_name: haproxy
  tunnel_bridge: "br-vxlan"
  management_bridge: "br-mgmt"
  provider_networks:
    - network:
        group_binds:
          - all_containers
          - hosts
        type: "raw"
        container_bridge: "br-mgmt"
        container_interface: "eth1"
        container_type: "veth"
        ip_from_q: "management"
        is_container_address: true
        is_ssh_address: true
    - network:
        group_binds:
          - neutron_linuxbridge_agent
        container_bridge: "br-vxlan"
```

```
        container_type: "veth"
        container_interface: "eth10"
        ip_from_q: "tunnel"
        type: "vxlan"
        range: "1:1000"
        net_name: "vxlan"
    - network:
        group_binds:
          - neutron_linuxbridge_agent
        container_bridge: "br-vlan"
        container_type: "veth"
        container_interface: "eth11"
        type: "vlan"
        range: "1:1"
        net_name: "vlan"
```

6. The remaining section of this file describes which server each service runs from. Most of the sections repeat, differing only in the name of the service. This is fine as the intention here is to tell OpenStack-Ansible which server (we give it a name so that Ansible can refer to it by name, and reference the IP associated with it) runs the Nova API, RabbitMQ, or the Glance service, for example. As these particular example services run on our controller nodes, and in a production setting there are at least three controllers, you can quickly see why this information repeats. Other sections refer specifically to other services, such as OpenStack compute. For brevity, a couple of sections are shown here, but continue editing the file to match your networking:

```
# Shared infrastructure parts
shared-infra_hosts:
  controller-01:
    ip: 172.29.236.110
  controller-02:
    ip: 172.29.236.111
  controller-03:
    ip: 172.29.236.112
# Compute Hosts
compute_hosts:
  compute-01:
    ip: 172.29.236.113
  compute-02:
    ip: 172.29.236.114
```

7. Save and exit the file. We will now need to generate some random passphrases for the various services that run in OpenStack. In OpenStack, each service—such as Nova, Glance, and Neutron (which are described through the book)—themselves have to authenticate with Keystone, and be authorized to act as a service. To do so, their own user accounts need to have passphrases generated. Carry out the following command to generate the required passphrases, which would be later used when the OpenStack playbooks are executed:

   ```
   cd /opt/openstack-ansible/scripts

   python pw-token-gen.py --file /etc/openstack_deploy/user_secrets.
   yml
   ```

8. Finally, there is another file that allows you to fine-tune the parameters of the OpenStack services, such as which backing store Glance (the OpenStack Image service) will be using, as well as configure proxy services ahead of the installation. This file is called `/etc/openstack_deploy/user_variables.yml`. Let's view and edit this file:

   ```
   cd /etc/openstack_deploy

   vi user_variables.yml
   ```

9. In a typical, highly available deployment—one in which we have three controller nodes—we need to configure Glance to use a shared storage service so that each of the three controllers have the same view of a filesystem, and therefore the images used to spin up instances. A number of shared storage backend systems that Glance can use range from NFS to Swift. We can even allow a private cloud environment to connect out over a public network and connect to a public service like Rackspace Cloud Files. If you have Swift available, add the following lines to `user_variables.yml` to configure Glance to use Swift:

   ```
   glance_swift_store_auth_version: 3
   glance_default_store: swift
   glance_swift_store_auth_address: http://172.29.236.117:5000/v3
   glance_swift_store_container: glance_images
   glance_swift_store_endpoint_type: internalURL
   glance_swift_store_key: '{{ glance_service_password }}'
   glance_swift_store_region: RegionOne
   glance_swift_store_user: 'service:glance'
   ```

   Tip: Latest versions of OpenStack-Ansible are smart enough to discover if Swift is being used and will update their configuration accordingly.

10. View the other commented out details in the file to see if they need editing to suit your environment, then save and exit. You are now ready to start the installation of OpenStack!

How it works...

Ansible is a very popular server configuration tool that is well-suited to the task of installing OpenStack. Ansible takes a set of configuration files that *Playbooks* (a defined set of steps that get executed on the servers) use to control how they executed. For OpenStack-Ansible, configuration is split into two areas: describing the physical environment and describing how OpenStack is configured.

The first configuration file, `/etc/openstack_deploy/openstack_user_config.yml`, describes the physical environment. Each section is described here:

```
cidr_networks:
  container: 172.29.236.0/24
  tunnel: 172.29.240.0/24
  storage: 172.29.244.0/24
```

This section describes the networks required for an installation based on OpenStack-Ansible. Look at the following diagram to see the different networks and subnets.

- **Container**: Each container that gets deployed gets an IP address from this subnet. The load balancer also takes an IP address from this range.
- **Tunnel**: This is the subnet that forms the VXLAN tunnel mesh. Each container and compute host that participates in the VXLAN tunnel gets an IP from this range (the VXLAN tunnel is used when an operator creates a Neutron subnet that specifies the `vxlan` type, which creates a virtual network over this underlying subnet). Refer to *Chapter 4*, *Neutron – OpenStack Networking* for more details on OpenStack networking.
- **Storage**: This is the subnet that was used when a client instance spun up in OpenStack request Cinder block storage:

```
used_ips:
  - "172.29.236.20"
  - "172.29.240.20"
  - "172.29.244.20"
  - "172.29.236.101,172.29.236.117"
  - "172.29.240.101,172.29.240.117"
  - "172.29.244.101,172.29.244.117"
  - "172.29.248.101,172.29.248.117"
```

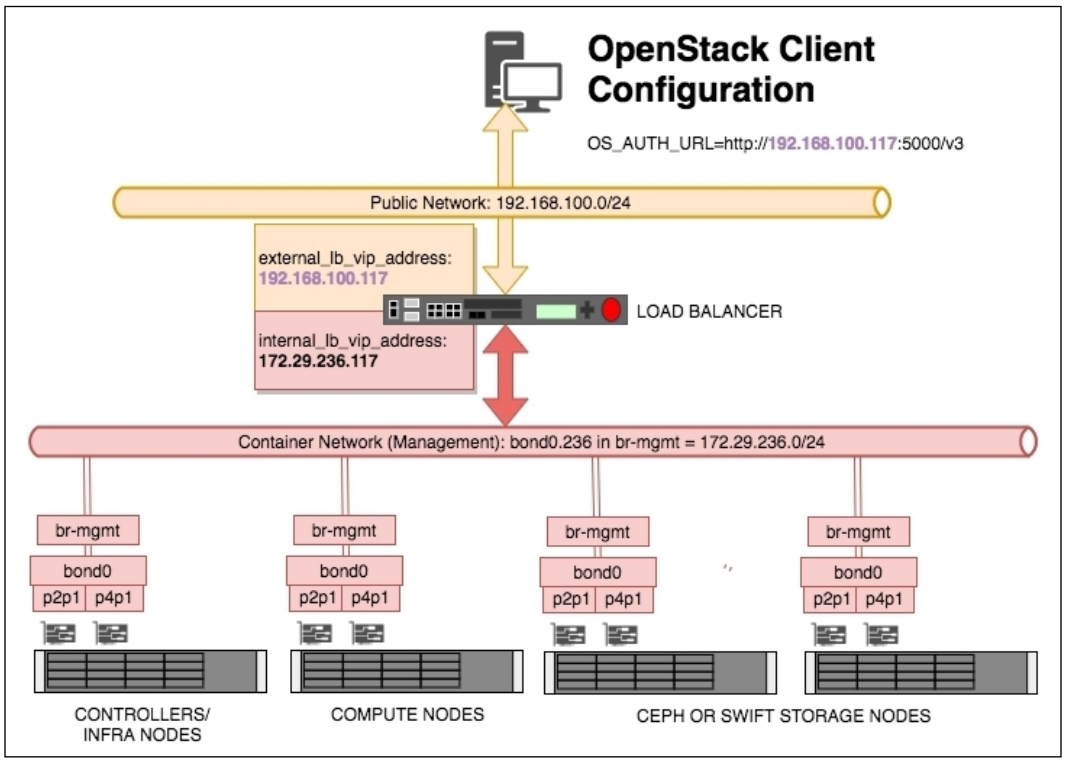
The `used_ips:` section refers to IP addresses that are already in use on that subnet, or reserved for use by static devices. Such devices are load balancers or other hosts that are part of the subnets that OpenStack-Ansible would otherwise have randomly allocated to containers:

```
global_overrides:
  internal_lb_vip_address: 172.29.236.117
```

```
external_lb_vip_address: 192.168.1.117
tunnel_bridge: "br-vxlan"
management_bridge: "br-mgmt"
storage_bridge: "br-storage"
```

The `global_overrides:` section describes the details around how the containers and bridged networking are set up. OpenStack-Ansible's default documentation expects Linux Bridge to be used; however, Open vSwitch can also be used. Refer to *Chapter 4*, *Neutron – OpenStack Networking*, for more details.

The `internal_lb_vip_address:` and `external_lb_vip_address:` sections refer to the *private* and *public* sides of a typical load balancer. The private (`internal_lb_vip_address`) is used by the services within OpenStack (for example, Nova calls communicating with the Neutron API would use `internal_lb_vip_address`, whereas a user communicating with the OpenStack environment once it has been installed would use `external_lb_vip_address`). See the following diagram:



A number of load balance pools will be created for a given **Virtual IP** (**VIP**) address, describing the IP addresses and ports associated with a particular service, and for each pool—one will be created on the public/external network (in the example, a VIP address of `192.168.100.117` has been created for this purpose), and another VIP for use internally by OpenStack (in the preceding example, the VIP address `172.29.236.117` has been created for this purpose).

The `tunnel_bridge:` section is the name given to the bridge that is used for attaching the physical interface that participates in the VXLAN tunnel network.

The `management_bridge:` section is the name given to the bridge that is used for all of the OpenStack services that get installed on the container network shown in the diagram.

The `storage_bridge:` section is the name given to the bridge that is used for traffic associated with attaching storage to instances or where Swift proxied traffic would flow.

Each of the preceding bridges must match the names you have configured in the `/etc/network/interfaces` file on each of your servers.

The next section, `provider_networks`, remains relatively static and untouched as it describes the relationship between container networking and the physical environment. Do not adjust this section.

Following the `provider_networks` section are the sections describing which server or group of servers run a particular service. Each block has the following syntax:

```
service_name:
  ansible_inventory_name_for_server:
    IP_ADDRESS
  ansible_inventory_name_for_server:
    IP_ADDRESS
```

Tip: Ensure the correct and consistent spelling of each server name (`ansible_inventory_name_for_server`) to ensure correct execution of your Ansible playbooks.

A number of sections and their use are listed here:

- `shared-infra_hosts`: This supports the shared infrastructure software, which is MariaDB/Galera and RabbitMQ
- `repo-infra_hosts`: This is the specific repository containers version of OpenStack-Ansible requested
- `haproxy_hosts`: When using HAProxy for load balancing, this tells the playbooks where to install and configure this service
- `os-infra_hosts`: These include OpenStack API services such as Nova API and Glance API

- `log_hosts`: This is where the rsyslog server runs from
- `identity_hosts`: These are the servers that run the Keystone (OpenStack Identity) Service
- `storage-infra_hosts`: These are the servers that run the Cinder API service
- `storage_hosts`: This is the section that describes Cinder LVM nodes
- `swift-proxy_hosts`: These are the hosts that would house the Swift Proxy service
- `swift_hosts`: These are the Swift storage nodes
- `compute_hosts`: This is the list of servers that make up your hypervisors
- `image_hosts`: These are the servers that run the Glance (OpenStack Image) Service
- `orchestration_hosts`: These are the servers that run the Heat API (OpenStack Orchestration) services
- `dashboard_hosts`: These are the servers that run the Horizon (OpenStack Dashboard) service
- `network_hosts`: These are the servers that run the Neutron (OpenStack Networking) agents and services
- `metering-infra_hosts`: These are the servers that run the Ceilometer (OpenStack Telemetry) service
- `metering-alarm_hosts`: These are the servers that run the Ceilometer (OpenStack Telemetry) service associated with alarms
- `metrics_hosts`: The servers that run the Gnocchi component of Ceilometer
- `metering-compute_hosts`: When using Ceilometer, these are the list of compute hosts that need the metering agent installed

Running the OpenStack-Ansible playbooks

To install OpenStack, we simply run the relevant playbooks. There are three main playbooks in total that we will be using:

- `setup-hosts.yml`
- `setup-infrastructure.yml`
- `setup-openstack.yml`

Getting ready

Ensure that you are the `root` user on the deployment host. In most cases, this is the first infrastructure controller node, `infra01`.

How to do it...

To install OpenStack using the OpenStack-Ansible playbooks, you navigate to the `playbooks` directory of the checked out Git repository, then execute each playbook in turn:

1. First change to the `playbooks` directory by executing the following command:

   ```
   cd /opt/openstack-ansible/playbooks
   ```

2. The first step is to run a syntax check on your scripts and configuration. As we will be executing three playbooks, we will execute the following against each:

   ```
   openstack-ansible setup-hosts.yml --syntax-check
   openstack-ansible setup-infrastructure.yml --syntax-check
   openstack-ansible setup-openstack.yml --syntax-check
   ```

3. Now we will run the first playbook using a special OpenStack-Ansible wrapper script to Ansible that configures each host that we described in the `/etc/openstack_deploy/openstack_user_config.yml` file:

   ```
   openstack-ansible setup-hosts.yml
   ```

4. After a short while, you should be greeted with a PLAY RECAP output that is all green (with yellow/blue lines indicating where any changes were made), with the output showing all changes were OK. If there are issues, review the output by scrolling back through the output and watch out for any output that was printed out in red. Refer to the *Troubleshooting the installation* recipe further on in this chapter. If all is OK, we can proceed to run the next playbook for setting up load balancing. At this stage, it is important that the load balancer gets configured. OpenStack-Ansible installs the OpenStack services in LXC containers on each server, and so far we have not explicitly stated which IP address on the container network will have that particular service installed. This is because we let Ansible manage this for us. So while it might seem counter-intuitive to set up load balancing at this stage before we know where each service will be installed—Ansible has already generated a dynamic inventory ahead of any future work, so Ansible already knows how many containers are involved and knows which container will have that service installed. If you are using an F5 LTM, Brocade, or similar enterprise load balancing kit, it is recommended that you use HAProxy temporarily and view the generated configuration to be manually transferred to a physical setup. To temporarily set up HAProxy to allow an installation of OpenStack to continue, modify your `openstack_user_config.yml` file to include a HAProxy host, then execute the following:

   ```
   openstack-ansible install-haproxy.yml
   ```

5. If all is OK, we can proceed to run the next Playbook that sets up the shared infrastructure services as follows:

   ```
   openstack-ansible setup-infrastructure.yml
   ```

6. This step takes a little longer than the first Playbook. As before, inspect the output for any failures. At this stage, we should have a number of containers running on each Infrastructure Node (also known and referred to as Controller Nodes). On some of these containers, such as the ones labelled Galera or RabbitMQ, we should see services running correctly on here, waiting for OpenStack to be configured against them. We can now continue the installation by running the largest of the playbooks—the installation of OpenStack itself. To do this, execute the following command:

   ```
   openstack-ansible setup-openstack.yml
   ```

7. This may take a while to run—running to hours—so be prepared for this duration by ensuring your SSH session to the deployment host will not be interrupted after a long time, and safeguard any disconnects by running the Playbook in something like `tmux` or `screen`. At the end of the Playbook run, if all is OK, congratulations, you have OpenStack installed!

How it works...

Installation of OpenStack using OpenStack-Ansible is conducted using a number of playbooks. The first playbook, `setup-hosts.yml`, sets up the hosts by laying down the container configurations. At this stage, Ansible knows where it will be placing all future services associated with OpenStack, so we use the dynamic inventory information to perform an installation of HAProxy and configure it for all the services used by OpenStack (that are yet to be installed). The next playbook, `setup-infrastructure.yml`, configures and installs the base Infrastructure services containers that OpenStack expects to be present, such as Galera. The final playbook is the main event—the playbook that installs all the required OpenStack services we specified in the configuration. This runs for quite a while—but at the end of the run, you are left with an installation of OpenStack.

The OpenStack-Ansible project provides a wrapper script to the `ansible` command that would ordinarily run to execute Playbooks. This is called `openstack-ansible`. In essence, this ensures that the correct inventory and configuration information is passed to the `ansible` command to ensure correct running of the OpenStack-Ansible playbooks.

Troubleshooting the installation

Ansible is a tool, written by people, that runs playbooks, written by people, to configure systems that would ordinarily be manually performed by people, and as such, errors can occur. The end result is only as good as the input.

Typical failures either occur quickly, such as connection problems, and will be relatively self-evident, or after long running jobs that may be as a result of load or network timeouts. In any case, the OpenStack-Ansible playbooks provide an efficient mechanism to rerun playbooks without having to repeat the tasks it has already completed.

On failure, Ansible produces a file in `/root` (as we're running these playbooks as `root`) called the *playbook* name, with the file extension of `.retry`. This file simply lists the hosts that had failed so this can be referenced when running the playbook again. This targets the single or small group of hosts, which is far more efficient than a large cluster of machines that successfully completed.

How to do it...

We will step through a problem that caused one of the playbooks to fail.

Note the failed playbook and then invoke it again with the following steps:

1. Ensure that you're in the `playbooks` directory as follows:

   ```
   cd /opt/openstack-ansible/playbooks
   ```

2. Now rerun that Playbook, but specify the `retry` file:

   ```
   ansible-openstack setup-openstack.yml --retry /root/setup-
   openstack.retry
   ```

3. In most situations, this will be enough to rectify the situation, however, OpenStack-Ansible has been written to be idempotent—meaning that the whole playbook can be run again, only modifying what it needs to. Therefore, you can run the Playbook again without specifying the `retry` file.

Should there be a failure at this first stage, execute the following:

1. First remove the generated `inventory` files:

   ```
   rm -f /etc/openstack_deploy/openstack_inventory.json
   rm -f /etc/openstack_deploy/openstack_hostnames_ips.yml
   ```

2. Now rerun the `setup-hosts.yml` playbook:

   ```
   cd /opt/openstack-ansible/playbooks
   openstack-ansible setup-hosts.yml
   ```

In some situations, it might be applicable to destroy the installation and begin again. As each service gets installed in LXC containers, it is very easy to wipe an installation and start from the beginning. To do so, carry out the following steps:

1. We first destroy all of the containers in the environment:

   ```
   cd /opt/openstack-ansible/playbooks
   openstack-ansible lxc-containers-destroy.yml
   ```

   You will be asked to confirm this action. Follow the ons-screen prompts.

2. We recommend you to uninstall the following package to avoid any conflicts with the future running of the playbooks, and also clear out any remnants of containers on each host:

```
ansible hosts -m shell -a "pip uninstall -y appdirs"
```

3. Finally, remove the inventory information:

```
rm -f /etc/openstack_deploy/openstack_inventory.json /etc/
openstack_deploy/openstack_hostnames_ips.yml
```

How it works...

Ansible is not perfect and so are computers. Sometimes failures occur in the environment due to SSH timeouts, or some other transient failure. Also, despite Ansible trying its best to retry the execution of a playbook, the result might be a failure. Failure in Ansible is quite obvious—it is usually predicated by outputs of red text on the screen. In most cases, rerunning the offending playbook may get over some transient problems. Each playbook runs a specific task, and Ansible will state which task has failed. Troubleshooting why that particular task had failed will eventually lead to a good outcome. Worst case, you can reset your installation from the beginning.

Manually testing the installation

Once the installation has completed successfully, the first step is to test the install. Testing OpenStack involves both automated and manual checks.

Manual tests verify user-journeys that may not normally be picked up through automated testing, such as ensuring horizon is displayed properly.

Automated tests can be invoked using a testing framework such as tempest or the OpenStack benchmarking tool—rally.

Getting ready

Ensure that you are `root` on the first infrastructure controller node, `infra01`.

How to do it...

The installation of OpenStack-Ansible creates several `utility` containers on each of the infra nodes. These utility hosts provide all the command-line tools needed to try out OpenStack, using the command line of course. Carry out the following steps to get access to a utility host and run various commands in order to verify an installation of OpenStack manually:

1. First, view the running containers by issuing the following command:

```
lxc-ls -f
```

2. As you can see, this lists a number of containers because the OpenStack-Ansible installation uses isolated Linux containers for running each service. By the side of each one its IP address and running state will be listed. You can see here that the container network of `172.29.236.0/24` was used in this chapter and why this was named this way. One of the containers on here is the utility container, named with the following format: `nodename_utility_container_randomuuid`. To access this container, you can SSH to it, or you can issue the following command:

   ```
   lxc-attach -n infra01_utility_container_34u477d
   ```

3. You will now be running a Terminal inside this container, with access only to the tools and services belonging to that containers. In this case, we have access to the required OpenStack clients. The first thing you need to do is source in your OpenStack credentials. The OpenStack-Ansible project writes out a generated bash environment file with an `admin` user and project that was set up during the installation. Load this into your bash environment with the following command:

   ```
   source openrc
   ```

Tip: you can also use the following syntax in Bash: `. openrc`

4. Now you can use the OpenStack CLI to view the services and status of the environment, as well as create networks, and launch instances. A few handy commands are listed here:

   ```
   openstack server list
   openstack network list
   openstack endpoint list
   openstack network agent list
   ```

How it works...

The OpenStack-Ansible method of installing OpenStack installs OpenStack services into isolated containers on our Linux servers. On each of the controller (or infra) nodes are about 12 containers, each running a single service such as nova-api or RabbitMQ. You can view the running containers by logging into any of the servers as root and issuing a `lxc-ls -f` command. The `-f` parameter gives you a full listing showing the status of the instance such as whether it is running or stopped.

One of the containers on the infra nodes has `utility` in its name, and this is known as a *utility container* in OpenStack-Ansible terminology. This container has OpenStack client tools installed, which makes it a great place to start manually testing an installation of OpenStack. Each container has at least an IP address on the container network—in the example used in this chapter this is the `172.29.236.0/24` subnet. You can SSH to the IP address of this container, or use another `lxc` command to attach to it: `lxc-attach -n <name_of_container>`. Once you have a session inside the container, you can use it like any other system, provided those tools are available to the restricted four-walls of the container. To use OpenStack commands, however, you first need to source the resource environment file which is named `openrc`. This is a normal bash environment file that has been prepopulated during the installation and provides all the required credentials needed to use OpenStack straight away.

Modifying the OpenStack configuration

It would be ludicrous to think that all of the playbooks would be needed to run again for a small change such as changing the CPU contention ratio from 4:1 to 8:1. So instead, the playbooks have been developed and tagged so that specific playbooks can be run associated with that particular project that would reconfigure and restart the associated services to pick up the changes.

Getting ready

Ensure that you are `root` on the *deployment host*. In most cases, this is the first infrastructure controller node, `infra01`.

How to do it...

The following are the common changes and how they can be changed using Ansible. As we'll adjust the configuration, all of these commands are executed from the same host you used to perform the installation.

To adjust the CPU overcommit/allocation ratio, carry out the following steps:

1. Edit the `/etc/openstack_deploy/user_variables.yml` file and modify (or add) the following line (adjust the figure to suit):

   ```
   nova_cpu_allocation_ratio: 8.0
   ```

2. Now execute the following commands to make changes in the environment:

   ```
   cd /opt/openstack-ansible/playbooks
   openstack-ansible os-nova-install.yml --tags 'nova-config'
   ```

For more complex changes, for example, to add configuration that isn't a simple one-line change in a template, we can use an alternative in the form of overrides. To make changes to the default Nova Quotas, carry out the following as an example:

1. Edit the `/etc/openstack_deploy/user_variables.yml` file and modify (or add) the following line (adjust the figure to suit):

```
nova_nova_conf_overrides:
  DEFAULT:
   quota_fixed_ips = -1
   quota_floating_ips = 20
   quota_instances = 20
```

2. Now execute the following commands to make changes in the environment:

```
cd /opt/openstack-ansible/playbooks
openstack-ansible os-nova-install.yml --tag 'nova-config'
```

Changes for Neutron, Glance, Cinder, and all other services are modified in a similar way. Adjust the name of the service in the syntax used. For example, to change a configuration item in the `neutron.conf` file, you would use the following syntax:

```
neutron_neutron_conf_overrides:
  DEFAULT:
   dhcp_lease_duration = -1
```

Then execute the following commands:

```
cd /opt/openstack-ansible/playbooks
openstack-ansible os-neutron-install.yml --tag 'neutron-config'
```

How it works...

We modified the same OpenStack-Ansible configuration files as in the *Configuring the installation* recipe and executed the `openstack-ansible playbook` command, specifying the playbook that corresponded to the service we wanted to change. As we were making configuration changes, we notified Ansible of this through the `--tag` parameter.

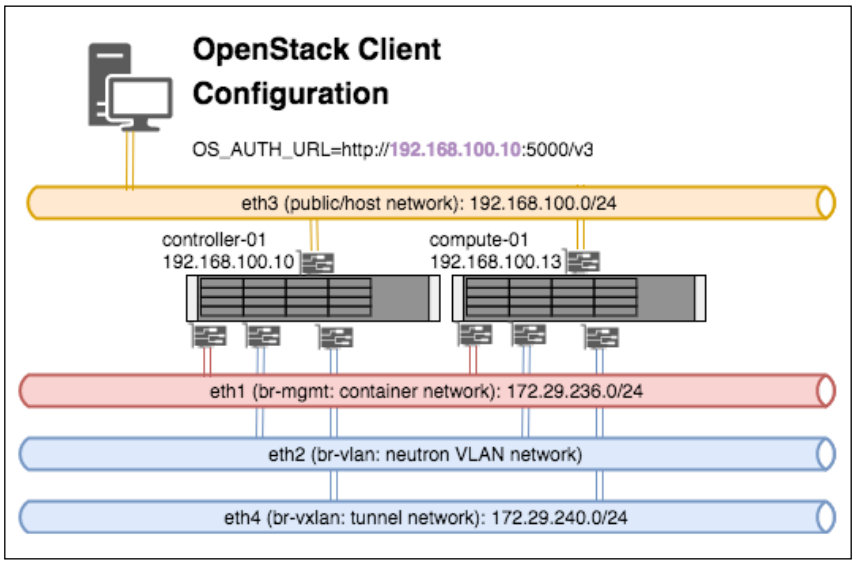
Refer to `https://docs.openstack.org/` for all configuration options for each service.

Virtual lab - vagrant up!

In an ideal world, each of us would have access to physical servers and the network kit in order to learn, test, and experiment with OpenStack. However, most of the time this isn't the case. By using an orchestrated virtual lab, using Vagrant and VirtualBox, allows you to experience this chapter on OpenStack-Ansible using your laptop.

The following Vagrant lab can be found at `http://openstackbook.online/`.

This is the architecture of the Vagrant-based OpenStack environment:



Essentially there are three virtual machines that are created (a controller node, a compute node and a client machine), and each host has four network cards (plus an internal bridged interface used by VirtualBox itself). The four network cards represent the networks described in this chapter:

- **Eth1**: This is included in the `br-mgmt` bridge, and used by the container network
- **Eth2**: This is included in the `br-vlan` bridge, and used when a VLAN-based Neutron network is created once OpenStack is up and running
- **Eth3**: This is the client or host network—the network we would be using to interact with OpenStack services (for example, the public/external side of the load balancer)
- **Eth4**: This is included in the `br-vxlan` bridge, and used when a VXLAN-based Neutron overlay network is created once OpenStack is up and running

Note that the virtual machine called `openstack-client`, which gets created in this lab, provides you with all the command-line tools to conveniently get you started with working with OpenStack.

Getting ready

In order to run a multi-node OpenStack environment, running as a virtual environment on your laptop or designated host, the following set of requirements are needed:

- A Linux, Mac, or Windows desktop, laptop or server. The authors of this book use macOS and Linux, with Windows as the host desktop being the least tested configuration.
- At least 16GB RAM. 24GB is recommended.
- About 50 GB of disk space. The virtual machines that provide the infra and compute nodes in this virtual environment are thin provisioned, so this requirement is just a guide depending on your use.
- An internet connection. The faster the better, as the installation relies on downloading files and packages directly from the internet.

How to do it...

To run the OpenStack environment within the virtual environment, we need a few programs installed, all of which are free to download and use: VirtualBox, Vagrant, and Git. VirtualBox provides the virtual servers representing the servers in a normal OpenStack installation; Vagrant describes the installation in a fully orchestrated way; Git allows us to check out all of the scripts that we provide as part of the book to easily test a virtual OpenStack installation. The following instructions describe an installation of these tools on Ubuntu Linux.

We first need to install VirtualBox if it is not already installed. We recommend downloading the latest available releases of the software. To do so on Ubuntu Linux as `root`, follow these steps:

1. We first add the `virtualbox.org` repository key with the following command:

   ```
   wget -q https://www.virtualbox.org/download/oracle_vbox_2016.asc
   -O- | sudo apt-key add -
   ```

2. Next we add the repository file to our `apt` configuration, by creating a file called `/etc/apt/sources.list.d/virtualbox.conf` with the following contents:

   ```
   deb http://download.virtualbox.org/virtualbox/debian xenial
   contrib
   ```

3. We now run an `apt update` to refresh and update the `apt` cache with the following command:

   ```
   apt update
   ```

4. Now install VirtualBox with the following command:

   ```
   apt install virtualbox-5.1
   ```

Once VirtualBox is installed, we can install Vagrant. Follow these steps to install Vagrant:

1. Vagrant is downloaded from `https://www.vagrantup.com/downloads.html`. The version we want is Debian 64-Bit. At the time of writing, this is version 2.0.1. To download it on our desktop issue the following command:

   ```
   wget https://releases.hashicorp.com/vagrant/2.0.1/vagrant_2.0.1_x86_64.deb
   ```

2. We can now install the file with the following command

   ```
   dpkg -i ./vagrant_2.0.1_x86_64.deb
   ```

The lab utilizes two vagrant plugins: `vagrant-hostmanager` and `vagrant-triggers`. To install these, carry out the following steps:

1. Install `vagrant-hostmanager` using the `vagrant` tool:

   ```
   vagrant plugin install vagrant-hostmanager
   ```

2. Install `vagrant-triggers` using the `vagrant` tool:

   ```
   vagrant plugin install vagrant-triggers
   ```

If Git is not currently installed, issue the following command to install `git` on a Ubuntu machine:

```
apt update
apt install git
```

Now that we have the required tools, we can use the `OpenStackCookbook` Vagrant lab environment to perform a fully orchestrated installation of OpenStack in a VirtualBox environment:

1. We will first checkout the lab environment scripts and supporting files with `git` by issuing the following command:

   ```
   git clone https://github.com/OpenStackCookbook/vagrant-openstack
   ```

2. We will change into the `vagrant-openstack` directory that was just created:

   ```
   cd vagrant-openstack
   ```

3. We can now orchestrate the creation of the virtual machines and installation of OpenStack using one simple command:

   ```
   vagrant up
   ```

Tip: This will take quite a while as it creates the virtual machines and runs through all the same playbook steps described in this chapter.

How it works...

Vagrant is an awesome tool for orchestrating many different virtual and cloud environments. It allows us to describe what virtual servers need to be created, and using Vagrant's provisioner allows us to run scripts once a virtual machine has been created.

Vagrant's environment file is called **Vagrantfile**. You can edit this file to adjust the settings of the virtual machine, for example, to increase the RAM or number of available CPUs.

This allows us to describe a complete OpenStack environment using one command:

```
vagrant up
```

The environment consists of the following:

- A controller node, `infra-01`
- A compute node, `compute-01`
- A client virtual machine, `openstack-client`

Once the environment has finished installing, you can use the environment by navigating to `http://192.168.100.10/` in your web browser. To retrieve the admin password, follow the steps given here and view the file named `openrc`.

There is a single controller node that has a `utility` container configured for use in this environment. Attach to this with the following commands:

```
vagrant ssh controller-01
sudo -i
lxc-attach -n (lxc-ls | grep utility)
openrc
```

Once you have retrieved the `openrc` details, copy these to your `openstack-client` virtual machine. From here you can operate OpenStack, mimicking a desktop machine accessing an installation of OpenStack utilizing the command line.

```
vagrant ssh openstack-client
openrc
```

You should now be able to use OpenStack CLI tools to operate the environment.

2
The OpenStack Client

In this chapter, we will cover the following topics:

- Introduction – using OpenStack
- Installing Python on Windows
- Installing the OpenStack clients
- Configuring your Linux or macOS environment
- Configuring your Windows environment
- Common OpenStack networking tasks
- Common OpenStack server (instances) tasks
- Common OpenStack image tasks
- Common OpenStack identity tasks
- Common OpenStack storage tasks
- Common OpenStack orchestration tasks

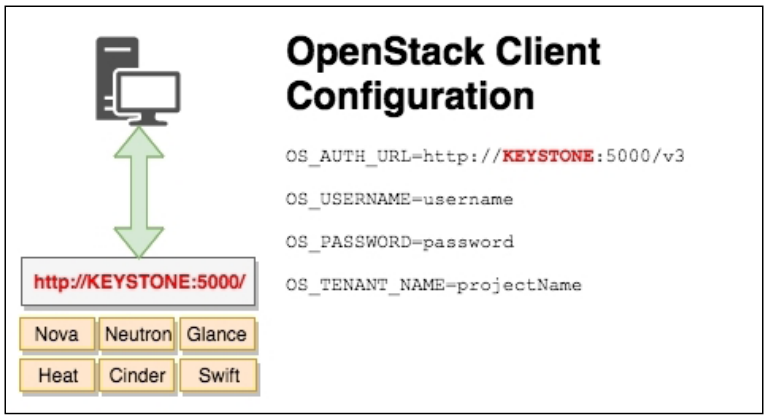
Introduction – using OpenStack

OpenStack can be interacted with in a number of ways – from using the Horizon (the OpenStack control panel) web interface to, the power of the command line to using third -party software, all of which are transparently integrated with OpenStack. In this chapter, we will introduce the OpenStack **Command-Line Interface** (**CLI**) and see how to configure it for use with your installation of OpenStack. The OpenStack command-line tools, like the vast majority of OpenStack, are written in Python. This means that there is a direct dependency on Python being available on the computer that will be running the clients.

When we interact with OpenStack, we are technically making REST API calls to the services that run the Service **APIs** (**Application Programming Interfaces**). A REST API defines a set of functions that developers can perform requests and receive responses via HTTP protocol such as GET and POST. The OpenStack command-line clients translate your intuitive commands into these HTTP calls. In a typical deployment of OpenStack, and one that has been described in *Chapter 1*, *Installing OpenStack with Ansible*, using Ansible, our OpenStack API services have been deployed across three Controller nodes. To allow us to interact with any one of the three Controller nodes so that each of them can respond independently, we place these services behind a load balancer. It is the load balancer **VIP** (**Virtual IP**), the IP address we have associated for use with each of the configured pools, that as a user of OpenStack that we are interested in.

There is one particular service that a user is interested in when configuring their environment for use with OpenStack and that is the `Keystone` service.

The `Keystone` service (OpenStack Identity service) essentially performs two functions in OpenStack. It authorizes users to allow them to perform the actions they have requested, as well as provides a catalog of services back to the user. This catalog is a mapping of the OpenStack service address endpoints. For this reason, we do not need to configure our client to know where to find each and every OpenStack service. When a user configures their environment to use OpenStack from the command line, the only information they need to be aware of is the IP address and port that `Keystone` has been installed on. See the following diagram for how this conceptually looks to a user:



This chapter is intended as a quick reference guide for commands that are explained in more detail throughout the book.

Installing Python on Windows

In order to be able to install the required OpenStack client tools under Windows, we must first prepare our Windows desktop machine with Python. The following applies to Windows 10.

Getting ready

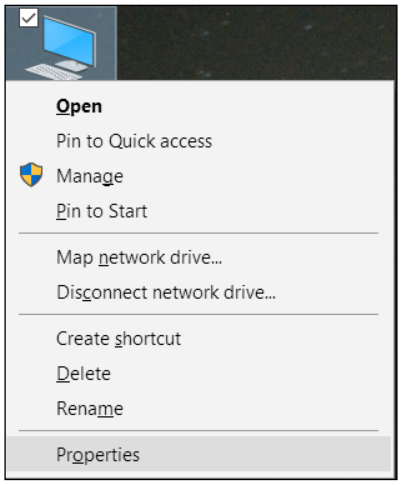
Ensure that you are logged into your desktop and have the following installed:

- PowerShell
- Python 2.7 from here: `https://www.python.org/downloads/windows/`
- Microsoft Visual C++ Build Tools with feature Windows 10 SDK

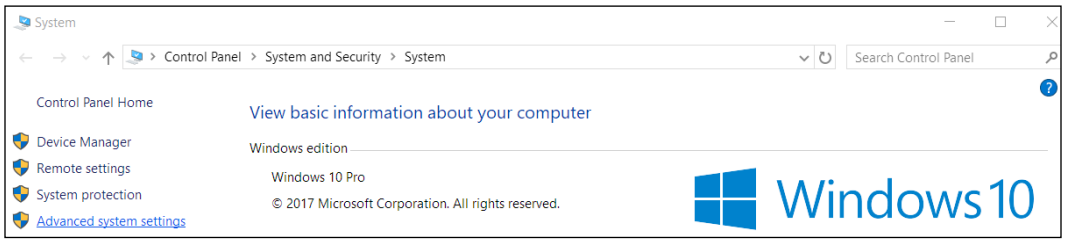
How to do it...

Configuring your Windows environment can be achieved from **Properties** of your PC. Follow these instructions to ensure that Python is available in your system path, as well as set the appropriate environment variables under PowerShell:

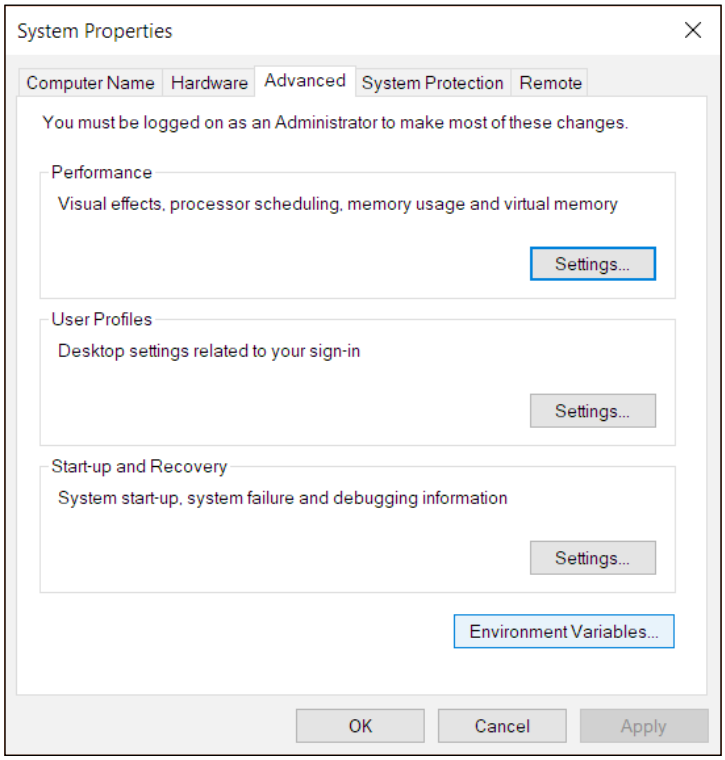
1. Navigate to the **This PC** icon on your desktop and choose **Properties**, as shown here:



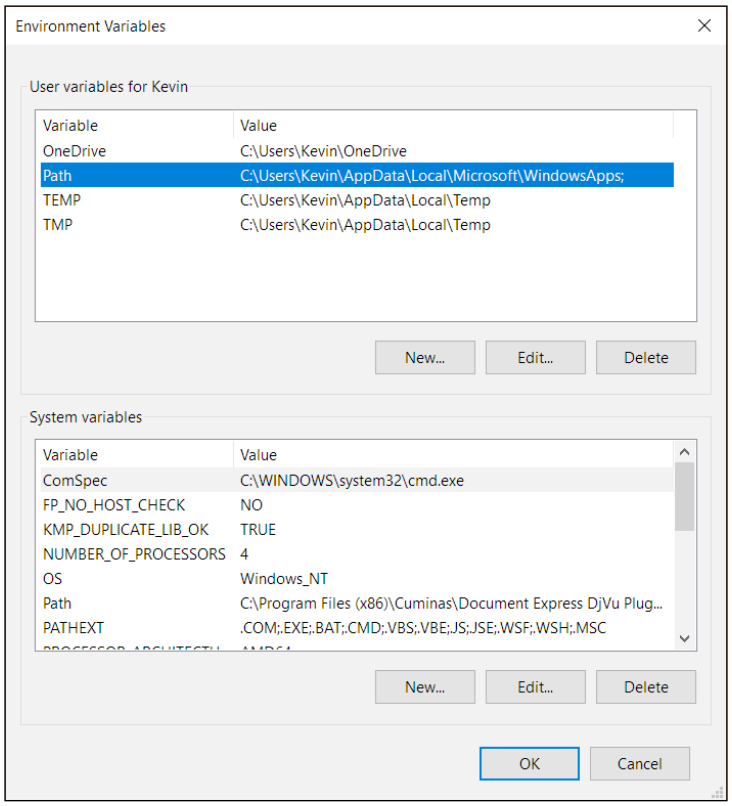
2. Next, choose **Advanced system settings** from the menu on the left:



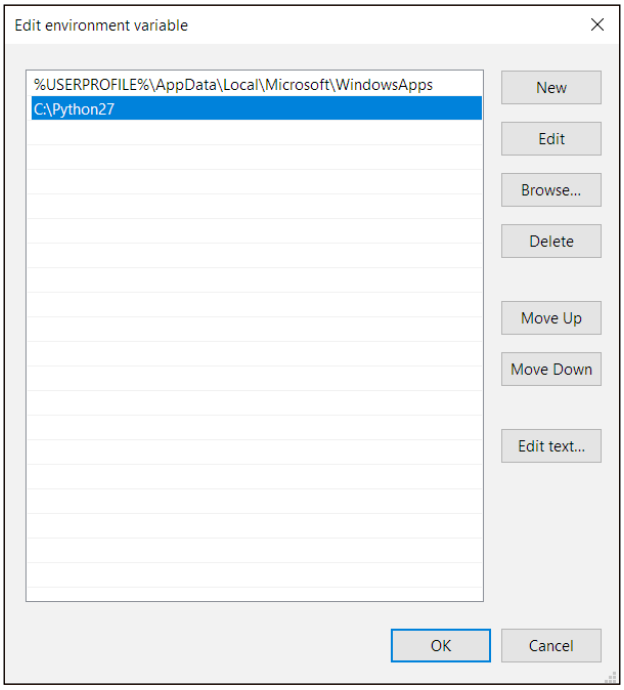
3. Now, select **Environment Variables** from the **Advanced** tab of the **System Properties** window, as shown here:



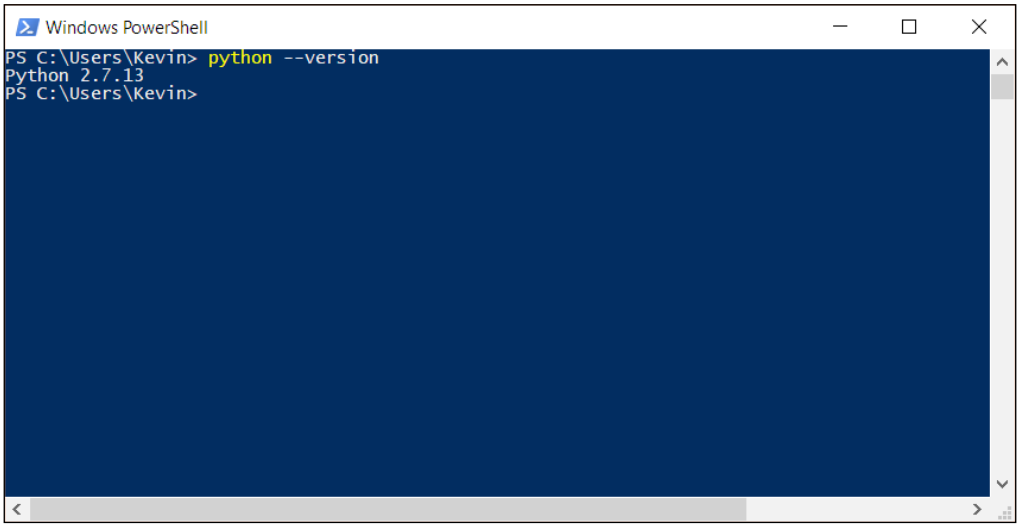
4. To set the path for Python, double-click on the **Path** entry as shown here:



5. Now add a **New** line to the path as shown as follows. We assume that you did a default installation of Python 2.7 and it is installed at `C:\Python27`:



6. Now click on **OK**. When you load up a PowerShell session, you should now be able to test that Python is working as expected, as shown here:



7. You should now be able to install the OpenStack clients as described in the next recipe.

How it works...

Configuring your Windows environment is a little more involved than it is for Unix/Linux environments. We first have to ensure that Python is set up properly, and available for use in a shell. We then to have a mechanism for loading environment variables into our shell, which isn't a native feature of Windows. We do this using a PowerShell script. However, because PowerShell is quite powerful, we have to remove a restriction to allow this to work. Once we have this all set up correctly, we are able to use the OpenStack environment from our Windows desktop.

Installing the OpenStack clients

There are a number of OpenStack clients available that are used to interact with OpenStack from the command line. Historically, each service in OpenStack has its own client. For example, the OpenStack Compute project, Nova, has its own `nova` client. Similarly, the OpenStack networking project, Neutron, also has its own client called `neutron` client. And so on.

Officially, there is a convergence to using one client: the OpenStack client. However, not all commands and features are available under this one tool. Moreover, the OpenStack client still requires each individual project command-line tool installed to function; however, it provides a more consistent interface without the need to remember each individual project name.

Getting ready

As we are preparing your desktop for interacting with OpenStack from the command line, you will appreciate that there are a variety of choices you can make for your desktop OS of choice. This section will describe the installation of OpenStack clients.

As we will be installing the OpenStack clients using `pip`, ensure that this is installed by following these steps:

1. First, load up a Terminal and become the *root* user with the following command:

   ```
   sudo -i
   ```

2. We'll be using `pip` to install the clients. If `pip` is not installed, carry out the following steps:

   On macOS or variants use this:

   ```
   wget  https://bootstrap.pypa.io/get-pip.py
   python get-pip.py
   ```

3. On popular Linux systems, use your package manager to install the packages:

   Ubuntu: Ensure that `python-dev` and `python-pip` are installed

   Red Hat: Ensure that `python-devel` and `python-pip` are installed

Windows: Be sure that you have Python installed and available in **PATH** and Microsoft Visual C++ Build Tools installed.

How to do it...

With `pip` installed on our system, we are able to install the clients using the following simple steps:

1. To install the OpenStack client, carry out the following command:

   ```
   pip install python-openstackclient
   ```

2. To install the individual clients, carry out the following commands:

   ```
   pip install python-novaclient
   pip install python-neutronclient
   pip install python-glanceclient
   pip install python-heatclient
   pip install python-keystoneclient
   pip install python-cinderclient
   pip install python-swiftclient
   ```

Each of the projects have their own client, so the syntax is:

```
pip install python-PROJECTclient
```

Alternative – use a preconfigured OpenStack client virtual machine

Sometimes the clients are in development at a different pace to the projects installed in your, environment which can make for version incompatibilities. To overcome this, either use `virtualenv` (`https://pypi.python.org/pypi/virtualenv`) or use a virtual machine, under VirtualBox, that has been preconfigured for accessing your OpenStack environments. To use a prebuilt virtual environment, carry out the following:

1. Clone the client VirtualBox Vagrant environment:

   ```
   git clone https://github.com/OpenStackCookbook/openstack-client.
   git
   ```

2. Launch the client:

```
cd openstack-client
vagrant up
```

3. Access the virtual machine:

```
vagrant ssh
```

How it works...

Installing the OpenStack clients is made very simple using the `pip` command-line tool, which is used to install Python packages. The main tool for using OpenStack on the command line is called the **OpenStack client**. This tool is used to control all aspects of OpenStack. However, there are some commands and options that have yet to make it into the main OpenStack clients. To overcome this, the older legacy project tools can still be used. Ensure that these are also installed using the following syntax:

```
pip install python-PROJECTclient
```

Replace *PROJECT* with the specific name of the OpenStack project, such as *glance* or *neutron*.

Alternatively, create these tools in small virtual machine so that the tools are always available. A Vagrant OpenStack client environment is available from `https://github.com/OpenStackCookbook/openstack-client`.

Configuring your Linux or macOS environment

The OpenStack tools are configured by setting environment variables in your shell or desktop.

Getting ready

Ensure that you have the OpenStack clients installed as described in the first recipe, *Introduction – using OpenStack*, in this chapter.

How to do it...

Configuration of your command-line environment is achieved by setting environment variables; however, it is easier and more convenient to place these variables in a file that we can later load into our environment. During the installation, OpenStack-Ansible creates a plain text file called `openrc` and places this in the `/root` directory of all the containers created. This file is a great starting point for configuring the environment as it has all the required elements needed to operate your CLI environment.

On your own client, for example, Linux or Mac-based, choose a working directory, such as `$HOME/openstack`, and create a file called `openrc` (or a meaningful name of your choice) with the following contents:

```
export OS_USERNAME=admin
export OS_PASSWORD=secrete
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://192.168.100.117:5000/v3
export OS_NO_CACHE=1
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_DOMAIN_NAME=Default

export OS_IDENTITY_API_VERSION=3
export OS_AUTH_VERSION=3
```

Now you use this file in your shell by sourcing it as follows:

```
source openrc
```

In Bash, you can use this syntax too:

```
. openrc
```

This file includes authentication details into your cloud environment. Keep it safe and ensure the permissions don't allow any other users to read this file. If in doubt, just set the permissions as follows:

```
chmod 0600 openrc
```

This will set the file to be read/write for you (user) only.

You can now use the command-line tools. If anything has gone wrong when executing the commands in the book, check your credentials in the file – ensure that you've set the correct tenant/project, username, and password, as well as ensure you're specifying the correct OpenStack authentication URL endpoint. Once you make any changes to this file, remember to source them back into your shell.

How it works...

Essentially, we're just setting some environment variables in our shell, which our client tools use to authenticate into our OpenStack environment. To make it easier though, we store these environment variables in a file. This makes access to our environment easy as we just run one command to set all the required credentials.

Configuring your Windows environment

Configuring your Windows environment for use with OpenStack requires a little more effort – but the basic premise remains: we're configuring our desktop so that it has access to environment variables, as well as ensuring that our tools, written in Python, are able to execute properly.

Getting ready

The following applies to Windows 10. Ensure that you have followed the steps to install Python.

How to do it...

Carry out the following to load the required environment variables into your Windows session:

1. To be able to source in the required OpenStack environment functions as we do on a Unix/Linux platform, we can achieve a similar outcome using some PowerShell. In the PowerShell Terminal, go and grab the following PowerShell script and download it into your working client directory (for example `C:\Users\Username\OpenStack`):

   ```
   mkdir OpenStack

   cd OpenStack

   wget https://raw.githubusercontent.com/OpenStackCookbook/vagrant-openstack/master/Source-OpenRC.ps1 -UseBasicParsing -OutFile Source-OpenRC.ps1
   ```

2. Using the same OpenStack credentials as described in the *Configuring your Linux or macOS environment* recipe, ensure that it is named `openrc` to match the following example and execute the following in PowerShell:

   ```
   .\Source-OpenRC.ps1 .\openrc
   ```

Warning: You may get an error executing the PowerShell script. Most Windows 10 desktops appear to have a default policy of restricted, which excludes the running of PowerShell scripts that aren't signed – even the ones you have created yourself. To remove this for this session, execute the following and acknowledge the warning:

```
Set-ExecutionPolicy –ExecutionPolicy RemoteSigned Unrestricted -Scope CurrentUser
```

3. You should now be able to use your OpenStack clients under your Windows desktop PowerShell Terminal.

How it works...

Configuring your Windows environment is a little more involved than it is for Unix/Linux environments. We have to have a mechanism for loading environment variables into our shell that isn't a native feature of Windows. We do this using a PowerShell script. However, we have to remove a restriction to allow this to work. Once we have this all set up correctly, we are able to use the OpenStack environment from our Windows desktop.

Common OpenStack networking tasks

This section outlines common OpenStack networking tasks for quick reference only. For more details on using Neutron and how Neutron works – including details of when and where to use features such as Floating IPs and Routers, refer to *Chapter 4*, *Neutron – OpenStack Networking*.

Getting ready

Ensure that you have the OpenStack clients installed, as described in the first recipes in this chapter.

How to do it...

Carry out the following steps to create and modify networks in OpenStack:

Creating a network

There are usually two steps to create a network: creating the equivalent of an L2 network, followed by assigning a subnet (and details to it).

1. First, create the network:

```
openstack network create NETWORK_NAME
```

2. Now create the subnet on this network:

```
openstack subnet create SUBNET_NAME
    --network NETWORK_NAME
    --subnet-range CIDR
```

Creating a provider network (for use with Floating IPs)

To create a floating IP provider network, carry out the following commands. This command assumes that our provider interface, as seen from OpenStack (and configured in Neutron), is using the "flat" interface. Typical deployments in a datacenter would likely use "vlan" as the provider type and device, so adjust to suit your environment.

1. First, create the network (in this example, we're specifying a provider type of `flat`):

```
openstack network create
    --share
    --project admin
    --external
    --default
    --provider-network-type flat
    --provider-physical-network flat
    GATEWAY_NET
```

2. Now we specify some options of the subnet that make sense for this network to be accessed from outside of OpenStack:

```
openstack subnet create
    --project admin
    --subnet-range 192.168.100.0/24
    --dhcp
    --gateway 192.168.100.1
    --allocation-pool start=192.168.100.200,end=192.168.100.250
    --network GATEWAY_NET
    GATEWAY_SUBNET
```

Creating a new security group

Creating a new security group, for example, `webserver` in the project `development`, is achieved as follows:

```
openstack security group create
    --project development
    webserver
```

Adding a rule to a security group

To add a rule to a security group called `webserver` created in the previous step, such as allowing inbound access from anywhere to port `80`, carry out the following:

```
openstack security group rule create
    --remote-ip 0.0.0.0/0
    --dst-port 80:80
    --protocol tcp
    --ingress
    --project development
    webserver
```

Creating a router

To create a router called `myRouter` in our project, execute the following command:

```
openstack router add myRouter
```

Adding a subnet to a router

To add a private tenant subnet, called `private-subnet`, to our router called `myRouter`, issue the following command:

```
openstack router add subnet myRouter private-subnet
```

Setting a gateway on the router

To add a gateway to our router, we first must ensure that the gateway network has been created with the `--external` flag as described in the *Creating a provider network* section in this chapter (for use with Floating IPs). We will then execute the following command to set the external gateway network to be that of `GATEWAY_NET` on our router called `myRouter`:

```
openstack router set myRouter --external-gateway GATEWAY_NET
```

Common OpenStack server (instances) tasks

This section outlines a number of common commands that can be run when operating with instances (for example, virtual machines). For more detailed information and explanation of each task, refer to *Chapter 5*, *Nova – OpenStack Compute*.

Getting ready

Ensure that you have the OpenStack clients installed as described in the first recipes of this chapter.

How to do it...

Carry out the following to launch and manipulate running instances:

Launching an instance

To launch in an instance from the command line, you need the following information:

- An image
- A network
- A flavor
- An optional security group (default is used otherwise)
- An optional key (if you intend to access the instance)

Carry out the following steps to launch an instance from the command line:

1. First, list the images available:

   ```
   openstack image list
   ```

2. Now we list the networks available (it will be the UUID of the Network we will use):

   ```
   openstack network list
   ```

3. We need a flavor, if you need reminding of them, list them with the following command:

   ```
   openstack flavor list
   ```

4. If you require specific security groups, list them with the following command:

   ```
   openstack security group list
   ```

5. If you need to get the name of the key pair to use, use the following command:

   ```
   openstack keypair list
   ```

6. Now you can boot the instance with the following command:

   ```
   openstack server create
       --image IMAGE
       --flavor FLAVOR
       --security-group SECGROUP
       --nic net-id=NETWORK_UUID
       --key-name KEYPAIR_NAME
       INSTANCE_NAME
   ```

Listing OpenStack Instances

To list the OpenStack instances launched, issue the following command:

```
openstack server list
```

Create instance snapshot

To create a snapshot of a running instance, carry out the following command:

```
openstack server image create
    --name snapshotRunningWebserver1
    myRunningWebserver1
```

Resizing an instance

To alter the flavor of a running instance and respawn with the new settings, carry out the following commands:

1. We first specify the new flavor size with the following command:

   ```
   openstack server resize --flavor m1.small myWebserver1
   ```

2. Next, list the running instances to confirm the state. The status should be `VERIFY_RESIZE`:

   ```
   openstack server list
   ```

   This shows output like the following:

```
+--------------------------------------+-------------+---------------+----------------------------+--------------+
| ID                                   | Name        | Status        | Networks                   | Image Name   |
+--------------------------------------+-------------+---------------+----------------------------+--------------+
| 58ea640b-16ba-447c-85db-952174d70f7c | myWebserver1 | VERIFY_RESIZE | GATEWAY_NET=192.168.100.25 | cirros-image |
+--------------------------------------+-------------+---------------+----------------------------+--------------+
```

3. Then we confirm the action with the following command:

   ```
   openstack server resize --confirm myWebserver1
   ```

Creating a flavor

To create a flavor, called `m1.tiny`, which is made up of 512 MB RAM, 1 vCPU, and no fixed size disk, carry out the following:

```
openstack flavor create
    --ram 512
    --disk 0
    --vcpus 1
    --public
    m1.tiny
```

Common OpenStack image tasks

This section outlines a number of steps, intended as a quick overview only, when operating against the OpenStack Image service (known as Glance). For more detailed information and explanation of each task, refer to *Chapter 6*, *Glance – OpenStack Image Service*.

Getting ready

Ensure that you have the OpenStack clients installed, as described in the first recipes in this chapter.

How to do it...

Carry out the following steps to create and modify images in OpenStack:

Uploading an image to Glance

Uploading an image to OpenStack is achieved with the following. To upload, a QCOW2 image such as one provided by CirrOS for testing, carry out the following command:

```
openstack image create
    --container-type bare
    --disk-format qcow2
    --public
    --file /path/to/cirros-0.3.5-x86_64-disk.img
```

Downloading an image or snapshot from Glance as a file

To download an image from Glance, perhaps for copying to another environment or to store as an offsite backup, carry out the following command:

```
openstack image save
    --file myImage.qcow2 myImage
```

Sharing images between projects

In most situations, an image is either public (available to all projects), or private (only available to the project that the image was uploaded onto). However, you are able to share a private image to isolated projects of your choosing. To do this, you will need the following:

- The image UUID of the image you will be sharing
- The UUID of the project that you will be sharing the image with

Carry out the following steps to share an image with another project. In the following example, we will share the `cirros-image`, currently only available in `admin` project with the `anotherProject` project:

1. First, query the project list:

   `openstack project list`

   This will bring back output like the following:

```
+----------------------------------+-------------------------------------------------------------------------+
| ID                               | Name                                                                    |
+----------------------------------+-------------------------------------------------------------------------+
| 31d0c145fad24a53a7838f2629465dab | admin                                                                   |
| 7321238d575342089c957d9e73f77af3 | anotherProject                                                          |
| b604fbcc259047b4813d45a71b1f246e | 31d0c145fad24a53a7838f2629465dab-0f2ba0b6-a047-4e31-9ec9-9ccb0e9        |
| db5594c6479346c690397d5bb5b5fa3c | service                                                                 |
+----------------------------------+-------------------------------------------------------------------------+
```

2. We will set the image to be shared:

   `openstack image set cirros-image --shared`

3. We will then tell OpenStack which project we want this sharing using the following command:

   `openstack image add project anotherProject`

   This will bring back output like the following – we will use the **image_id** in the next step:

```
+------------+--------------------------------------+
| Field      | Value                                |
+------------+--------------------------------------+
| created_at | 2017-12-14T11:08:59Z                 |
| image_id   | f6578a80-5f6f-4f2d-9a8a-9d84cec8a60d |
| member_id  | 7321238d575342089c957d9e73f77af3     |
| schema     | /v2/schemas/member                   |
| status     | pending                              |
| updated_at | 2017-12-14T11:08:59Z                 |
+------------+--------------------------------------+
```

4. Important: as a user in the **receiving** (**anotherProject**) project, execute the following:

   `openstack image set --accept f6578a80-5f6f-4f2d-9a8a-9d84cec8a60d`

5. Now, as that same user, you can confirm that you can see this shared image by executing an image listing:

   `openstack image list`

This will bring back output like the following, showing the available image:

```
+--------------------------------------+--------------+--------+
| ID                                   | Name         | Status |
+--------------------------------------+--------------+--------+
| f6578a80-5f6f-4f2d-9a8a-9d84cec8a60d | cirros-image | active |
| d870f943-7617-4a57-a1d9-c3cdaedce03e | xenial-image | active |
+--------------------------------------+--------------+--------+
```

Common OpenStack identity tasks

This section outlines a number of common steps to take for a number of common actions using the OpenStack Identity service. This is intended as a quick reference guide only. For more detailed information and explanation of each task, refer to *Chapter 3*, *Keystone – OpenStack Identity Service*.

Getting ready

Ensure that you have the OpenStack clients installed, as described in the first recipes of this chapter.

How to do it...

Carry out the following steps to create and modify users and projects in OpenStack:

Creating a new project

Creating a new user in a project is achieved with the following command. For example, to create the project called `development`, execute the following command:

```
openstack project create development
```

Creating a user

To create a user called `developer`, with a password of `password123`, carry out the following command:

```
openstack user create
    --domain default
    --password password123
    --enable
    developer
```

Adding a user to a project

To add a user with the `_member_` role (regular user) to the `development` project, carry out the following command:

```
openstack role add
    --project development
    --user developer
    _member_
```

Changing a user's password as an administrator

As an administrator, you have the ability to alter someone's password. To do this for the `developer` user, carry out the following command:

```
openstack user set --password cookbook4 developer
```

Changing your own password

To change your own password to something else, issue the following command:

```
openstack user password set --password cookbook4
```

Common OpenStack storage tasks

This section outlines a number of common tasks using the OpenStack Block and Object Storage service. For more information on storage, refer to *Chapter 7*, *Cinder – OpenStack Block Storage* and *Chapter 8*, *Swift – OpenStack Object Storage*.

Getting ready

Ensure that you have the OpenStack clients installed, as described in the first recipes of this chapter.

How to do it...

Carry out the following steps to create and modify users and projects in OpenStack:

Create a new Cinder volume

To create a new Cinder block storage volume, carry out the following command. The size is in gigabytes:

```
openstack volume create --size 5 my5GVolume
```

Attaching a volume

To attach a volume to a running instance, carry out the following command. The running **instance UUID** is used and can be found by listing the running instances:

```
openstack server add volume my5GVolume 58ea640b-16ba-447c-85db-
952174d70f7c
```

Detaching a volume

To detach a volume, first unmount it from the running instance as you would normally, then carry out the following command:

```
openstack server remove volume my5GVolume 58ea640b-16ba-447c-85db-
952174d70f7c
```

Creating a volume snapshot

To make a snapshot of a volume, carry out the following steps. First, you must detach the volume from the running instance to ensure data consistency. The action is described in the previous task.

Now once detached, issue the following

```
openstack snapshot create --name myVolumeSnapshot myVolume
```

Listing Object Storage statistics

To display information about Object Storage containers, carry out the following command:

```
openstack object store account show
```

Listing Object Storage containers and contents

To list contents of an object store, issue the following command:

```
openstack object store list
openstack object store list myContainer
```

Creating and uploading files to the Object Storage service

To create an Object Storage container, issue the following command:

```
openstack object create myContainer
```

To upload files to an Object Storage container, issue the following command:

```
openstack object create myContainer myFile
```

Downloading from Object Storage

To download files from an Object Storage container, use this command

```
openstack object save myContainer myFile
```

To download all files from an Object Storage container, issue the following command

```
openstack object save myContainer
```

Common OpenStack orchestration tasks

This section outlines a number of common tasks using the OpenStack Orchestration (Heat) service to launch *stacks* (orchestrated environments using Heat). For more information on Heat and orchestration, refer to *Chapter 9*, *OpenStack Orchestration Using Heat and Ansible*.

Getting ready

Ensure that you have the OpenStack clients installed, as described in the first recipes of this chapter.

How to do it...

Carry out the following steps to create and use Heat templates in OpenStack to create orchestrated environments:

Launch a stack from a template and environment file

To launch a stack from a heat orchestration template (hot), issue the following command:

```
openstack stack create
    --template myStack.yml
    --environment myStack-Env.yml
    myStack
```

Listing stacks

To list the running stacks, issue the following command:

```
openstack stack list
```

Deleting a running stack

To destroy a running stack named `myStack`, issue the following command:

```
openstack stack delete myStack
```

Listing resources in a stack

To list the resources in a running stack, issue the following command:

```
openstack stack resource list
```

To list the details of a specific resource, for example, named `myResource`, issue the following command:

```
openstack stack resource show myResource
```

Viewing the outputs of a running Stack

To view the outputs produced by the Stack, issue the following command:

```
openstack stack output list
```

To view the details of the specific output, carry out the following command:

```
openstack stack output show myOutput
```

3
Keystone – OpenStack Identity Service

In this chapter, we will cover the following topics:

- Introduction – OpenStack Identity
- Creating OpenStack domains in Keystone
- Enabling domains in the OpenStack dashboard
- Creating OpenStack projects in Keystone
- Configuring roles in Keystone
- Adding users in Keystone
- Configuring groups in Keystone
- Deleting projects
- Deleting users
- Deleting roles
- Deleting groups
- Deleting domains
- OpenStack endpoint information

Introduction – OpenStack Identity

The OpenStack Identity service, known as **Keystone**, provides services for authenticating and managing user accounts and role information for our OpenStack cloud environment.

It is a crucial service that underpins the authentication and verification between all of our OpenStack cloud services and is the first service that needs to be installed within an OpenStack environment. The OpenStack Identity service authenticates users and projects by sending a validated authorization token between all OpenStack services. This token is passed to the other services, such as Storage and Compute, to grant user access to specific functionalities. Therefore, configuration of the OpenStack Identity service must be completed first before using any of the other services. Setting up of the Identity service involves the creation of appropriate roles for users and services, projects, the user accounts, and the service API endpoints that make up our cloud infrastructure. Since we are using Ansible for deploying our environment (refer to *Chapter 1*, *Installing OpenStack with Ansible* for more details), all the basic configuration is done for us in the Ansible playbooks.

In Keystone, we have the concepts of domains, projects, roles, users, and user groups. A Keystone domain (not to be confused with a DNS domain) is a high level OpenStack Identity resource that contains projects, users, and groups. A project has resources such as users, images, and instances, as well as networks in it that can be restricted only to that particular project, unless explicitly shared with others. A user can belong to one or more projects and is able to switch between them to gain access to those resources. Users within a project can have various roles assigned. Users can be organized into user groups and the groups can have roles assigned to them. In the most basic scenario, a user can be assigned either the role of admin or just be a member. When a user has admin privileges within a project, the admin is able to utilize features that can affect the project (such as modifying external networks), whereas a normal user is assigned the member role. This member role is generally assigned to perform user-related roles, such as spinning up instances, creating volumes, and creating isolated, project-specific networks.

Projects used to be called **tenants** in early versions of OpenStack.

Creating OpenStack domains in Keystone

If you wish to use more than one domain for your OpenStack deployment, consider using separate domains. Think of domains as separate accounts or departments in large organizations. For this section, we will create a domain for our project, called `bookstore`.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with admin privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

To create a domain in our OpenStack environment, perform the following step:

1. We start by creating a domain called `bookstore` as follows:

   ```
   openstack domain create --description "Book domain" bookstore
   ```

   The output will look similar to this:

   ```
   +-------------+----------------------------------+
   | Field       | Value                            |
   +-------------+----------------------------------+
   | description | Book domain                      |
   | enabled     | True                             |
   | id          | 9ac17ca9db364339ac4c2ec0a63944dc |
   | name        | bookstore                        |
   +-------------+----------------------------------+
   ```

How it works...

In OpenStack, high level identity resources can be grouped under different domains. If you have to manage distinct organizations within your OpenStack environment, having separate domains for managing them might be very beneficial. By default, your OpenStack environment most likely has a default domain called "Default." By running the preceding command, we just created an additional domain that can used to manage resources related to this book. The syntax is as follows:

```
openstack domain create --description <description> <name>
```

The `description` parameter is also optional, but highly recommended. The domain name must be unique to other domains in the environment.

In our recipes, we will use the `--domain` parameter and specify a domain name. If the domain is not specified, the OpenStack command-line client will use the domain set for the current user that was specified in the `openrc` file. Most likely, that will be the `default` domain.

Enabling domains in the OpenStack dashboard

If you are using multiple domains in your OpenStack environment, you will need to enable them in the OpenStack dashboard (Horizon) as well. To do so, the `OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT` variable in Horizon settings needs to be set to `True`. In this example, we will show you how to do so using OpenStack Ansible playbook.

Getting ready

We are going to use Ansible to update Horizon settings. Make sure that you have access to your `openstack-ansible` deployment host.

How to do it...

To enable multidomain support in the OpenStack dashboard, we will update one horizon variable, `OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT` in `local_settings.py` using the `openstack-ansible` deployment tool. First, you will need to connect to your `openstack-ansible` deployment host. Once connected, execute the following steps:

1. Edit the `/etc/openstack_deploy/user_variables.yml` file to add the following line:

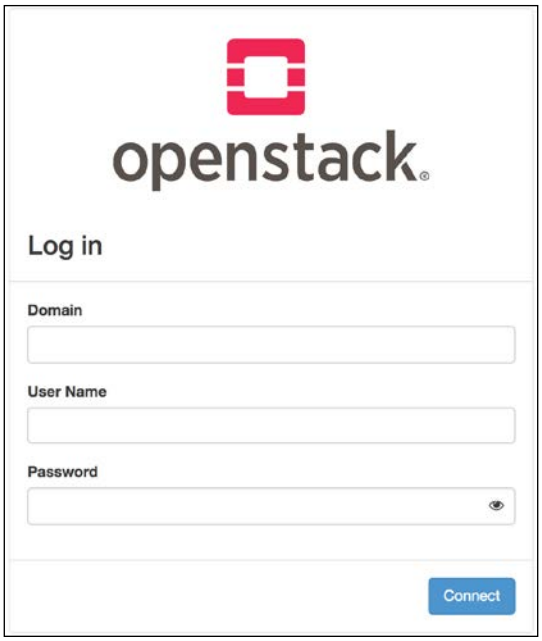
   ```
   horizon_keystone_multidomain_support: True
   ```

2. Deploy Horizon with the `openstack-ansible` command:

   ```
   openstack-ansible
    /opt/openstack-ansible/playbooks/os-horizon-install.yml
   ```

   The `openstack-ansible` command produces a lot of output. For brevity, its output has been omitted.

3. Launch the OpenStack dashboard to verify that the login screen now shows domain field:



How it works...

In OpenStack, if you are taking advantage of the multiple domain functionality, you have full control via command-line tools. However, if you want to be able to use the OpenStack dashboard with multiple domains, you will need to enable Horizon's multidomain support. To do so, you need to update the Horizon settings file. Since we are using the `openstack-ansible` tool, we updated the `user_variables.yml` file and ran the `openstack-ansible` command. This command updated the required variable and restarted the `apache2` (HTTP server) services on the Horizon container.

Creating OpenStack projects in Keystone

Users can't be created without having a project assigned to them, so these must be created first. For this section, we will create a project for our users, called `cookbook`.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with admin privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

To create a project in our OpenStack environment, perform the following steps:

1. We start by creating a project called `cookbook` as follows:

```
openstack project create --domain bookstore
    --description "Cookbook Project" cookbook
```

   The command should produce the output similar to this:

```
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Cookbook Project                 |
| domain_id   | 9ac17ca9db364339ac4c2ec0a63944dc |
| enabled     | True                             |
| id          | c076aa3d961d4b8e85c0a7fe2c563708 |
| is_domain   | False                            |
| name        | cookbook                         |
| parent_id   | 9ac17ca9db364339ac4c2ec0a63944dc |
+-------------+----------------------------------+
```

How it works...

By running the preceding command, we just created a project in our `bookstore` domain. The syntax is as follows:

```
openstack project create --domain <domain>
    --description <description> <name>
```

The `domain` parameter is optional, used to determine which domain the project will belong to. If omitted, the project will be created in the `default` domain. The `description` parameter is also optional, but highly recommended. The `name` parameter has to be unique to the installation.

Configuring roles in Keystone

Roles are the permissions given to users within a project. Roles can also be scoped to a particular domain, making it possible to restrict permissions for particular users to a domain and project. If you used Ansible to install your OpenStack environment, it should already contain some default roles, such as `admin` and `_member_`. Here we will configure one role, a `cloud_admin` role that allows for administration of our example `bookstore` domain environment and a `user` role for the `default` domain that is given to ordinary users who will be using the cloud environment.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with admin privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

To create the required roles in our OpenStack environment, perform the following steps:

1. Creation of the `cloud_admin` role is done as follows:

   `openstack role create --domain bookstore cloud_admin:`

```
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | 9ac17ca9db364339ac4c2ec0a63944dc |
| id        | f33dcb973c6649ea9c7cb6d8ce7a1188 |
| name      | cloud_admin                      |
+-----------+----------------------------------+
```

2. To configure the `user` role for the `default` domain, execute the following command:

   `openstack role create user`

```
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | default                          |
| id        | 29fad90250ef4e1fa05e1fd872c34e76 |
| name      | user                             |
+-----------+----------------------------------+
```

   This command created a new role called `user`. Since we didn't specify a domain, it was created under the `default` domain.

3. View roles associated with the `bookstore` domain:

   `openstack role list --domain bookstore`

```
+----------------------------------+-------------+-----------+
| ID                               | Name        | Domain    |
+----------------------------------+-------------+-----------+
| f33dcb973c6649ea9c7cb6d8ce7a1188 | cloud_admin | bookstore |
+----------------------------------+-------------+-----------+
```

4. List roles associated with the current admin user:

```
openstack role list
```

```
+----------------------------------+------------------+
| ID                               | Name             |
+----------------------------------+------------------+
| 26be34552f10466591f61b5ddf687407 | swiftoperator    |
| 29fad90250ef4e1fa05e1fd872c34e76 | user             |
| 2ff2b5fd48a54566801efb5297d874fd | heat_stack_owner |
| 3275ad86248045efa90eafed7a8afdd7 | admin            |
| 727c56418b2a40efb8bb43ab506e5b1d | heat_stack_user  |
| 9fe2ff9ee4384b1894a90878d3e92bab | _member_         |
| c2e2e4c4c9b54ac8bb7d5c2b7748fecf | reseller_admin   |
| d43790edbbfd4439817894d02037b3aa | ResellerAdmin    |
| d5aa33d9cbe4451cb126b024742b1b80 | remote_image     |
+----------------------------------+------------------+
```

How it works...

Creation of roles is simply achieved using the OpenStack client, specifying the `role create` option with the following syntax:

```
openstack role create --domain <domain_name> <role_name>
```

The `domain_name` attribute is optional, if you omit it, a role for the `default` domain will be created.

For the `role_name` attribute, the `admin` and `_member_` roles names cannot be used again. The `admin` role is set by default in OpenStack code starting with the Pike release, and in releases before Pike in the `/etc/keystone/policy.json` file, as having administrative rights:

```
{
    "admin_required": "role:admin or is_admin:1",
}
```

The `_member_` role is also configured by default in the dashboard when a nonadmin user is created through the web interface.

On creation of the role, the ID associated with the role is returned, and we can use it when assigning roles to users. To see a list of roles and the associated IDs in our environment, we can issue the following command:

```
openstack role list --domain <domain_name>
```

If the `domain` parameter is not specified, you will see roles associated only with your current user's domain.

Adding users in Keystone

Adding users to OpenStack Identity service requires the user to belong to a domain or a project in the domain and to be assigned a role defined in the domain or the project. For this section, we will create two users. The first user will be named `cloud_admin` and will have the `cloud_admin` role assigned to them in the `cookbook` project. The second user will be named `reader` and will have the default `_member_` role assigned to them in the same `cookbook` project.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with admin privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

In order to create a user in OpenStack and assign roles to it, we will need to execute the commands listed below.

1. First, get the `bookstore` domain ID or domain name:

```
openstack domain list
+----------------------------------+-----------+---------+--------------------+
| ID                               | Name      | Enabled | Description        |
+----------------------------------+-----------+---------+--------------------+
| 7fa46026e99d412f8e8e1bafbe223b28 | heat      | True    |                    |
| 9ac17ca9db364339ac4c2ec0a63944dc | bookstore | True    | Book domain        |
| default                          | Default   | True    | The default domain |
+----------------------------------+-----------+---------+--------------------+
```

2. Using the `bookstore` domain name, create a new `cloud_admin` user:

```
openstack user create --domain bookstore
    --password verysecret
    cloud_admin
```

```
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | 9ac17ca9db364339ac4c2ec0a63944dc |
| enabled             | True                             |
| id                  | 3280e5d54e0048a381fa728e51f1ef7a |
| name                | cloud_admin                      |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

3. Next, get a cookbook project ID or project name:

```
openstack project list
```

```
+----------------------------------+----------+
| ID                               | Name     |
+----------------------------------+----------+
| 2381eba808264d1889bd526f7ab613d6 | alt_demo |
| 34637400afb9427d968242677f00df38 | admin    |
| 6e076f2e3b5544bfacd47dafc28a5e46 | service  |
| c076aa3d961d4b8e85c0a7fe2c563708 | cookbook |
| f4c838a069884bf0a6f0641ef3160be1 | demo     |
+----------------------------------+----------+
```

4. Create the `reader` user in the `bookstore` domain `cookbook` project:

```
openstack user create --domain bookstore
    --project cookbook
    --password verysecret
    reader
```

```
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| default_project_id  | c076aa3d961d4b8e85c0a7fe2c563708 |
| domain_id           | 9ac17ca9db364339ac4c2ec0a63944dc |
| enabled             | True                             |
| id                  | 81607c74ea7340198ccda9c5e568df91 |
| name                | reader                           |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

5. Assign `cloud_admin` user to the `admin` role:

```
openstack role add --domain bookstore
    --user cloud_admin
    --role-domain bookstore
    admin
```

There is no output from this command.

6. Assign the `reader` user to the `_member_` role:

```
openstack role add --project cookbook
    --user reader _member_
```

There is no output from this command.

7. List user and role assignment:

```
openstack role assignment list
```

   The output will be a matrix of role, user, group, project, and domain IDs. We omit the example output due to sheer size of the table that contains a matrix of IDs for each role, user, group, project, domain, and whether the role was inherited.

How it works...

Adding users in the OpenStack Identity service involves a number of steps and dependencies. First, a domain and a project are required for the user to be part of. A user must always belong to a domain. If there are no custom domains created, a `default` domain will be used. Once the project exists, the user can be added. At this point, the user has no role associated, so the final step is to assign the role to this user, such as `_member_`, `admin`, or a custom role.

To create a user with the `user create` option, the syntax is as follows:

```
openstack user create --domain <domain>
    --password <password> <user_name>
```

The `user_name` attribute is an arbitrary name, but cannot contain any spaces. A `password` attribute must be present. In the previous examples, these were set to `verysecret`. If the `domain` attribute is not specified, the `default` domain will be set.

To assign a role to a user with the `role add` option, the syntax is as follows for the `default` domain:

```
openstack role add --project <project>
    --user <user>
    <role>
```

For a user in a custom domain, use the following syntax to assign a role:

```
openstack role add --domain <domain>
    --user <user>
    --role-domain <role_domain>
    <role>
```

The `role_domain` parameter is the name (or ID) of the domain to which the role belongs.

We will also need to have the names or IDs of the user, role, and project in order to assign roles to users. These names or IDs can be found using the following commands:

```
openstack project list
    openstack user list
    openstack role list
```

The `--domain <domain>` option is only required if custom domains are used. To obtain a list of domains, issue the following command:

```
openstack domain list
```

To get a matrix of domain, project, user, and role assignments, use the following command:

```
openstack role assignment list
```

Configuring groups in Keystone

If you wish to organize users by their roles, you can create a user group using Keystone groups. Groups are owned by a domain. In this example, we will create one group, `reader_group`, and set a `cloud_admin` role to it. We will also add two users to it, `reader` and `reader1`. We will also verify that the users belong to the group, as well as remove one of the users afterwards.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with admin privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

In order to create groups and assign users to them, we will need to execute the following steps:

1. Start by creating a group:

```
openstack group create --domain bookstore
    --description "Bookstore reader group"
    reader_group
```

```
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Bookstore reader group           |
| domain_id   | 9ac17ca9db364339ac4c2ec0a63944dc |
| id          | 6ed63575c8774ac19b55c4e68bdf3346 |
| name        | reader_group                     |
+-------------+----------------------------------+
```

2. List existing groups:

```
openstack group list
```

```
+----------------------------------+--------------+
| ID                               | Name         |
+----------------------------------+--------------+
| 6ed63575c8774ac19b55c4e68bdf3346 | reader group |
+----------------------------------+--------------+
```

3. Add group to the role:

```
openstack role add --group reader_group
    --domain bookstore
    --role-domain bookstore
    cloud_admin
```

   Both the `--domain` and `--role-domain` parameters are needed. There is no output to this command.

4. Add the `reader` user to the group:

```
openstack group add user --group-domain bookstore
    --user-domain bookstore
    reader_group
    reader
```

   This produces a message like the following:

```
reader added to group reader_group
```

5. Add the `reader1` user to the group:

```
openstack group add user --group-domain bookstore
    --user-domain bookstore
    reader_group
    reader1
```

   This should output the following message:

```
reader1 added to group reader_group
```

6. Check that `reader` is in the `reader_group` group:

   ```
   openstack group contains user reader_group reader
   ```

   This should output the following message:

   ```
   reader in group reader_group
   ```

7. Check that `reader1` is in the `reader_group` group:

   ```
   openstack group contains user readergroup reader1
   ```

   This produces a message like the following:

   ```
   reader1 in group readergroup
   ```

8. Remove the `reader1` user from the `reader_group` group:

   ```
   openstack group remove user reader_group reader1
   ```

   This produces a message like the following:

   ```
   reader1 removed from group reader_group
   ```

9. Check again whether `reader1` is in the `reader_group` group:

   ```
   openstack group contains user reader_group reader1
   ```

   This produces a message like the following:

   ```
   reader1 not in group reader_group
   ```

How it works...

Adding users to identity groups is a good way to grant them specific set of roles without individually assigning each role to a user. If you have to manage users that always get the same set of roles, you can create a user group and add or remove users as needed, rather than setting individual roles to each user. Roles are assigned to the group in a similar fashion that they are assigned to the user. In our example, we first created a custom group, `reader_group`. The command for creating groups is as follows:

```
openstack group create --domain <domain-name>
    --description <group-description>
    <group-name>
```

Since we are using a custom domain in our example, `<domain-name>` is needed. However, if `<domain-name>` is omitted, domain for the current user will be used.

Add group to a role:

```
openstack role add --group <group>
    --domain <domain>
    --role-domain <role_domain>
    <role>
```

The `role_domain` parameter is the name (or ID) of the domain to which the role belongs.

We will also need to have the name of the group, the name of the role, and the name of the project in order to assign roles to users. These names can be found using the following commands:

```
openstack project list
    openstack group list
    openstack role list
```

Pass `--domain <domain>` as an option to the preceding commands if you have custom domains. To obtain a list of domains:

```
openstack domain list
```

To add a user to the group, use the following:

```
openstack group add user --group-domain <group_domain>
    --user-domain <user_domain>
    <group>
    <user>
```

To remove a user from the group, execute the following command:

```
openstack group remove user <group> <user>
```

To verify if the user belongs to the group, use this command:

```
openstack group contains user <group> <user>
```

To get a matrix of domain, project, user, group, and role assignments, use the following command:

```
openstack role assignment list
```

You can also use IDs instead of names for projects for all of the preceding commands instead of names.

Groups were introduced in Keystone v3.

Deleting projects

Projects can be deleted even if they have users associated with them, so be sure to delete users and other project assets before deleting a project, otherwise you risk having orphaned resources. In this example, we will show how to delete a project named `oldbook`.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with admin privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

In order to delete a project, execute the following commands:

1. Get the project's name from a current project list:

   ```
   openstack project list
   ```

   ```
   +----------------------------------+----------+
   | ID                               | Name     |
   +----------------------------------+----------+
   | 2381eba808264d1889bd526f7ab613d6 | alt_demo |
   | 34637400afb9427d968242677f00df38 | admin    |
   | 6e076f2e3b5544bfacd47dafc28a5e46 | service  |
   | c076aa3d961d4b8e85c0a7fe2c563708 | cookbook |
   | e70912f53a8c4b1bb75e99de18db1732 | oldbook  |
   | f4c838a069884bf0a6f0641ef3160be1 | demo     |
   +----------------------------------+----------+
   ```

2. Delete the project:

   ```
   openstack project delete oldbook
   ```

   This command will have no output.

How it works...

Deleting unnecessary projects is simple when using OpenStack command-line tool. Be sure to delete only empty projects. Start with a command to get existing projects:

```
openstack project list
```

Delete a project by executing the following:

```
openstack project delete <project>
```

Here, the `<project>` parameter can be either project ID or project name.

If project has users assigned, they will not be deleted, but the project will be.

Deleting users

Removing users from OpenStack Identity is a simple one step process. In this example, we will show how to delete a user named `oldreader`.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with admin privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

In order to delete a user, execute the following commands:

1. Get the user's name from the users in a current domain:

   ```
   openstack user list
   ```

   ```
   +----------------------------------+---------------------+
   | ID                               | Name                |
   +----------------------------------+---------------------+
   | 04fdb90ce9414c17837e8798638d7866 | designate           |
   | 12f4bb1040144fe79f7ea64e937efce3 | keystone            |
   | 140d7a1d25ee452aa0334f384f2e09f2 | cinder              |
   | 2691dc86d02e4c22a65ef85e3f47a20b | stack_domain_admin  |
   | 28b2ba508e0241528b781854b602469b | nova                |
   | 644299adb2a043af8c435b84efe013c0 | swift               |
   | 67d124ea1a0242768d55534bf02b2d14 | developer           |
   | 6f65086f244b4d78a53b7c8965f74c2e | oldreader           |
   | 72dfeb591bb549e882009168c8fe3251 | glance              |
   | 792b25e2bd034200869b23e1e07a4033 | heat                |
   | 8b3632f9c6244432abd5b236733dc814 | dispersion          |
   | b3cd17987bc64f6783c1c52ae6e5a6ff | admin               |
   | c5a31faa7f444f65bfa93a59f58c50f7 | neutron             |
   | dfe887e4ff2c4ce2899d842d878d593f | alt_demo            |
   | ea6b80e54cc649c4ba4a569ab2ae3274 | placement           |
   | edd0d6d781a149d9a07e96189aa5c654 | demo                |
   +----------------------------------+---------------------+
   ```

2. Delete the `oldreader` user:

   ```
   openstack user delete oldreader
   ```

   This command will have no output.

How it works...

Deleting unnecessary users is simple when using OpenStack command-line tool. Start with a command to get existing users:

```
openstack user list
```

Delete a user by executing the following:

```
openstack user delete <user>
```

Here the `<user>` parameter can be either user ID or username.

Deleting roles

Removing roles from the OpenStack Identity is a simple one step. In this example, we will show how to delete a role named `oldrole`.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with admin privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

In order to delete a role, execute the following commands:

1. Get the role's name from a current role list:

   ```
   openstack role list
   ```

```
+----------------------------------+------------------+
| ID                               | Name             |
+----------------------------------+------------------+
| 26be34552f10466591f61b5ddf888408 | oldrole          |
| 29fad90250ef4e1fa05e1fd872c34e76 | user             |
| 2ff2b5fd48a54566801efb5297d874fd | heat_stack_owner |
| 3275ad86248045efa90eafed7a8afdd7 | admin            |
| 727c56418b2a40efb8bb43ab506e5b1d | heat_stack_user  |
| 9fe2ff9ee4384b1894a90878d3e92bab | _member_         |
| c2e2e4c4c9b54ac8bb7d5c2b7748fecf | reseller_admin   |
| d43790edbbfd4439817894d02037b3aa | ResellerAdmin    |
| d5aa33d9cbe4451cb126b024742b1b80 | remote_image     |
+----------------------------------+------------------+
```

2. Delete the `oldrole` role:

   ```
   openstack role delete oldrole
   ```

   This command will have no output.

How it works...

Deleting unnecessary roles is simple when using OpenStack command-line tool. Start with a command to get an existing role:

```
openstack role list
```

Delete a role by executing the following:

```
openstack role delete <role>
```

Here the `<role>` parameter can be either role ID or role name.

Deleting groups

Removing user groups from the OpenStack Identity service is a simple one step. In this example, we will show how to delete a group named `oldgroup`. Note that when you delete a group, it doesn't delete the users assigned to the group. The role and group permission mappings will be gone, and as a result, users may lose permissions.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with admin privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

In order to delete a user group, execute the following commands:

1. Get the group's name from a current group list:

   ```
   openstack group list
   ```

```
+----------------------------------+-------------+
| ID                               | Name        |
+----------------------------------+-------------+
| 6ed63575c8774ac19b55c4e68bdf3346 | readergroup |
| 6ed63585c9345633546754d98acf4374 | oldgroup    |
+----------------------------------+-------------+
```

2. Delete the `oldgroup` group:

   ```
   openstack group delete oldgroup
   ```

   This command will have no output.

How it works...

Deleting unnecessary groups is simple when using OpenStack command-line tool. Start with a command to get existing group:

```
openstack group list
```

Delete a group by executing the following:

```
openstack group delete <group>
```

Here the `<group>` parameter can be either group ID or group name.

Deleting domains

Keystone domains can be deleted if there are no users associated with them. If there are any users associated with the domain when trying to delete it, an error will be shown. In this example, we will show how to delete a domain that is no longer being used, called `olddomain`.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with admin privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

In order to delete a domain, execute the following commands:

1. Get the domain's name from a current domain list:

```
openstack domain list
+----------------------------------+-----------+---------+--------------------+
| ID                               | Name      | Enabled | Description        |
+----------------------------------+-----------+---------+--------------------+
| 7fa46026e99d412f8e8e1bafbe223b28 | heat      | True    |                    |
| 9ac17ca9db364339ac4c2ec0a63944dc | bookstore | True    | Book domain        |
| default                          | Default   | True    | The default domain |
| 8743598cd239809898ac3434c2789891 | olddomain | True    |                    |
+----------------------------------+-----------+---------+--------------------+
```

2. Verify that there are no users associated with the `olddomain` domain that we will be deleting:

   ```
   openstack user list --domain olddomain
   ```

   This list should be empty before proceeding. If it is not, delete all the users before proceeding to the next step.

3. Disable the domain:

   ```
   openstack domain set --disable  olddomain
   ```

   This command will have no output.

4. Delete domain:

   ```
   openstack domain delete olddomain
   ```

   If successful, this command will have no output.

How it works...

Deleting unnecessary domains requires that domains have no users associated with them.

Verify that there are no users attached to this domain:

```
openstack user list --domain <domain>
```

Deleting a domain requires that it first be disabled. A domain can be disabled even if there are users attached to it:

```
openstack domain set --disable  <domain>
```

Only after a domain is disabled and no longer has any users associated with it, will you be able to delete a domain:

```
openstack domain delete <domain>
```

If you need to delete users, refer to the *Deleting users* recipe, earlier in this chapter.

OpenStack endpoint information

If you need to find out the endpoint information for different OpenStack services, or the services running on the OpenStack installation, use the OpenStack service catalog. If you need to automate any OpenStack tasks or are integrating any other tools with OpenStack, catalog will also come in handy. The catalog lists services and endpoint information for each of them. The catalog may be different for each deployment.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with admin privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

To list OpenStack catalog information, execute the following command:

```
openstack catalog list
```

How it works...

In OpenStack, different service API endpoints can be found out using the OpenStack catalog. The catalog will list all the OpenStack services that it knows about and their respective internal, admin, and public URLs. The catalog is also handy if you are trying to automate any of the OpenStack tasks. On the **Command-Line Interface** (**CLI**), list the available endpoints by running the following command:

```
openstack catalog list
```

```
+-----------+-----------+-----------------------------------------------------------------------------------------+
| Name      | Type      | Endpoints                                                                               |
+-----------+-----------+-----------------------------------------------------------------------------------------+
| neutron   | network   | RegionOne                                                                               |
|           |           |   public: https://192.168.100.117:9696                                                 |
|           |           | RegionOne                                                                               |
|           |           |   internal: http://172.29.236.117:9696                                                  |
|           |           | RegionOne                                                                               |
|           |           |   admin: http://172.29.236.117:9696                                                     |
|           |           |                                                                                         |
| cinderv2  | volumev2  | RegionOne                                                                               |
|           |           |   public: https://192.168.100.117:8776/v2/34637400afb9427d968242677f00df38              |
|           |           | RegionOne                                                                               |
|           |           |   internal: http://172.29.236.117:8776/v2/34637400afb9427d968242677f00df38              |
|           |           | RegionOne                                                                               |
|           |           |   admin: http://172.29.236.117:8776/v2/34637400afb9427d968242677f00df38                 |
|           |           |                                                                                         |
| placement | placement | RegionOne                                                                               |
|           |           |   public: https://192.168.100.117:8780/placement                                        |
|           |           | RegionOne                                                                               |
|           |           |   internal: http://172.29.236.117:8780/placement                                        |
|           |           | RegionOne                                                                               |
|           |           |   admin: http://172.29.236.117:8780/placement                                           |
|           |           |                                                                                         |
| keystone  | identity  | RegionOne                                                                               |
|           |           |   admin: http://172.29.236.117:35357/v3                                                 |
|           |           | RegionOne                                                                               |
|           |           |   internal: http://172.29.236.117:5000/v3                                               |
|           |           | RegionOne                                                                               |
|           |           |   public: https://192.168.100.117:5000/v3                                              |
|           |           |                                                                                         |
```

The output from our `catalog` command has been trunkated for this example. The full catalog command will list all the installed services.

The service catalog will be different for each OpenStack installation.

4
Neutron – OpenStack Networking

In this chapter, we will cover the following topics:

- Introduction to OpenStack networking
- Managing networks, subnets, and ports
- Creating provider networks
- Creating tenant networks
- Creating ports
- Updating network attributes
- Deleting ports
- Deleting networks
- Managing routers and floating IPs
- Attaching networks to routers
- Creating and assigning floating IPs
- Deleting routers
- Managing security groups
- Managing load balancers

Introduction to OpenStack networking

Networking in OpenStack is provided by a project that goes by the name of Neutron. Neutron is an API-driven system that manages physical and virtual networking resources in an OpenStack cloud. Operators and users can leverage the Neutron API to build rich network architectures that best suit the requirements of their applications.

Neutron utilizes a pluggable and extensible architecture that allows developers to write robust drivers that implement and configure virtual and physical network resources, including switches, routers, firewalls, and load balancers. Neutron is composed of an API server and various agents that are responsible for implementing the virtual network architected by users. The following diagram demonstrates how the Neutron API server interacts with various plugins and agents to construct networking in the cloud:

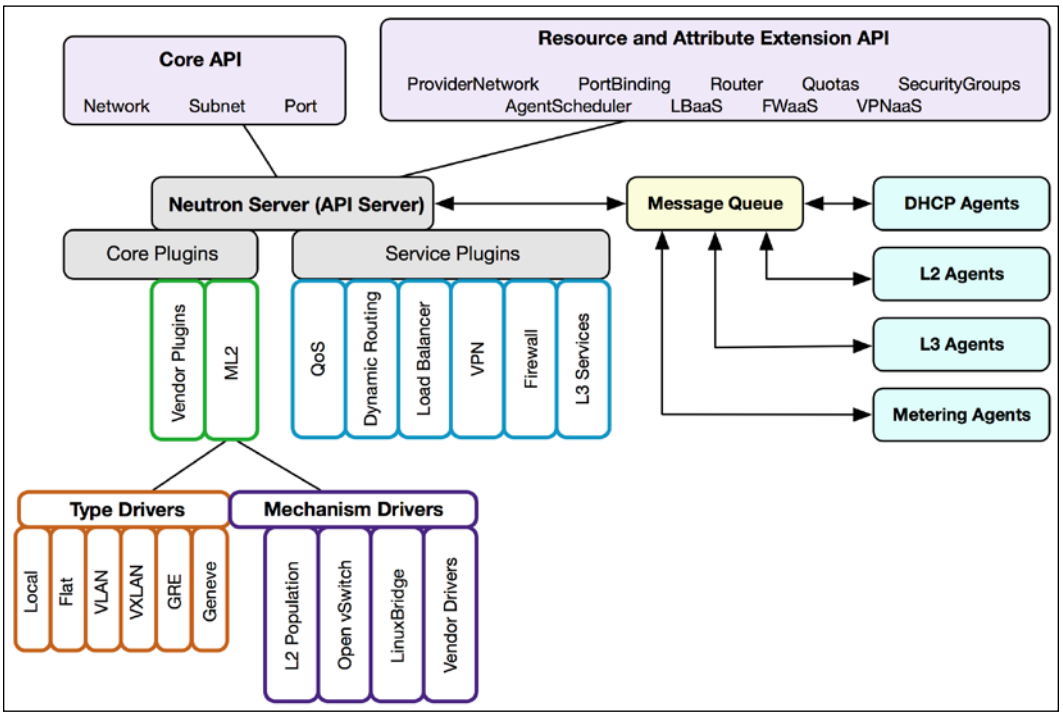


Figure 4.1

The figure demonstrates the interaction between the Neutron API service, Neutron plugins and drivers, and services such as L2 and L3 agents in a stock OpenStack-based cloud. As network actions are performed via the API or agents, Neutron publishes messages to a queue that are consumed and implemented by the agents.

This chapter will cover many common networking-related tasks, but will not go far into the details of any given topic. For an introduction to the fundamentals of Neutron networking, refer to *OpenStack Networking Essentials*, *Packt Publishing*, 2016. For a more in-depth look at Neutron features, including the core functionality of switching and routing, load balancing, VPN, and more, refer to *Learning OpenStack Networking, Second Edition*, *Packt Publishing*, 2015.

Managing networks, subnets, and ports

Networks, subnets, and ports make up the foundation of Neutron's logical network architecture. A network describes a layer 2 segment, and is typically used to define a boundary such as a VLAN. A subnet is an IPv4 or IPv6 address block that is associated with the network. Networks can be associated with one or more subnets. Lastly, a port represents a switch port on a logical switch that spans the entire cloud. A port object contains information about the device it is associated with, including its MAC addresses, IP addresses, and device ID. A device could be a virtual machine instance interface, a virtual router interface, or some other device that will be connected to the virtual network.

Network objects in OpenStack have many attributes that describe how that network connects the physical and virtual infrastructures. The following table describes a few of these details:

Attribute	Description
`provider:physical_network`	This describes the physical interface used for this network. The label here is an alias for interfaces such as `eth0` and `bond1`. The alias is referred to as a **provider label** and is configured in the respective plugin and agent configuration files. This is used primarily by the `flat` and `vlan` type networks.
`provider:segmentation_id`	This describes the segment ID, such as VLAN ID or VXLAN VNI. It may not be used for all network types.
`provider:network_type`	This describes the network type, such as Flat, VLAN, VXLAN, and GRE.
`router:external`	Boolean (true or false) is used to determine if the network is eligible for use as a floating IP pool.

The role of the user creating a network determines which attributes can be specified by that user at network creation. An administrative user can specify details such as the physical network or segmentation ID when creating a network. Regular users must rely on Neutron to automatically provision the network based on details set in Neutron configuration files, including a pool of segmentation IDs per physical network from which to choose from. Networks that are created specifically to connect virtual devices to the physical network infrastructure are often referred to as **provider networks**, since their attributes are deliberately set based on the environment or data center in which they reside. Provider networks are typically shared across projects or tenants and often provide connectivity to upstream devices that can facilitate routing in and out of the environment. Networks that are created by regular users are referred to as **tenant networks**, and are typically only used by the project or tenant that created them. In most cases, tenant networks must be connected to virtual routers, which are in turn connected to provider networks, and can provide connectivity in and out of connected tenant networks. In the following sections, we will look at common tasks involving those resources.

Creating provider networks

When creating a provider network in OpenStack, one must provide attributes that describe how the network is connected to the physical infrastructure. These attributes include the network type, network interface used by the server, and the segmentation ID of the network. Typically, provider networks are only created and managed by users with administrator-level permissions. Provider networks can be shared or private, and they can also be used as floating IP networks when connected to Neutron routers when the network's `router:external` attribute has been set to `True`.

Getting ready

When creating a provider network, you must be authenticated as an administrator. You will need the following details, at a minimum, for the network:

- Network name
- Provider label
- Network type
- Segmentation ID

For our example, the following will be used:

- Network name: `COOKBOOK_PROVIDER_NET`
- Provider label: `vlan`
- Network type: `vlan`
- Segmentation ID: `200`

You will need the following details, at a minimum, for the corresponding subnet:

- Subnet name
- Network name or ID
- Subnet range (CIDR)

For our example, the following will be used:

- Subnet name: `COOKBOOK_PROVIDER_SUBNET`
- Network name or ID: `COOKBOOK_PROVIDER_NET`
- Subnet range (CIDR): `192.168.200.0/24`

How to do it...

With the OpenStack client installed on our system, we are now able to create a provider network with the following steps:

1. Create the network:

```
openstack network create COOKBOOK_PROVIDER_NET \
--provider-network-type vlan \
--provider-physical-network vlan \
--provider-segment 200
```

   The output will resemble the following:

```
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2017-10-11T21:26:24Z                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | c881ce20-1649-4f03-bea7-40da536e21b2 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | False                                |
| is_vlan_transparent       | None                                 |
| mtu                       | 1500                                 |
| name                      | COOKBOOK_PROVIDER_NET                |
| port_security_enabled     | True                                 |
| project_id                | 4819f952f3d0411183956113f0727b30     |
| provider:network_type     | vlan                                 |
| provider:physical_network | vlan                                 |
| provider:segmentation_id  | 200                                  |
| qos_policy_id             | None                                 |
| revision_number           | 2                                    |
| router:external           | Internal                             |
| segments                  | None                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      |                                      |
| updated_at                | 2017-10-11T21:26:24Z                 |
+---------------------------+--------------------------------------+
```

2. Create the subnet:

```
openstack subnet create COOKBOOK_PROVIDER_SUBNET \
--network COOKBOOK_PROVIDER_NET \
--subnet-range 192.168.200.0/24
```

The output will resemble the following:

Field	Value
allocation_pools	192.168.200.2-192.168.200.254
cidr	192.168.200.0/24
created_at	2017-10-12T11:25:07Z
description	
dns_nameservers	
enable_dhcp	True
gateway_ip	192.168.200.1
host_routes	
id	83b8a728-4914-4667-af19-bd8b798a2e25
ip_version	4
ipv6_address_mode	None
ipv6_ra_mode	None
name	COOKBOOK_PROVIDER_SUBNET
network_id	c881ce20-1649-4f03-bea7-40da536e21b2
project_id	4819f952f3d0411183956113f0727b30
revision_number	0
segment_id	None
service_types	
subnetpool_id	None
tags	
updated_at	2017-10-12T11:25:07Z
use_default_subnet_pool	None

How it works...

Provider networks are created with the following syntax:

```
openstack network create NETWORK_NAME \
--provider-network-type NETWORK_TYPE \
--provider-physical-network PROVIDER_LABEL \
--provider-segment SEGMENTATION_ID \
[--external | --internal]
```

Creating a network creates a logical layer 2 segment, whose details are used to construct virtual network connections within the Cloud that connect virtual machines and other virtual network objects to the physical infrastructure.

The `provider-network-type` parameter defines the type of network. Options include `vlan`, `vxlan`, `gre`, `flat`, `geneve`, and `local`, and these must be supported by the configured network driver.

The `provider-physical-network` parameter defines the interface used for the network. In Neutron, interfaces are not referenced directly, but are mapped to a **provider label**. In an OpenStack-Ansible deployment, the default provider label is `vlan` and maps to a physical interface such as `bond1` or `eth1`.

The `provider-segment` parameter defines the layer 2 segmentation ID used by the network. For the `vlan` network types, the segmentation ID is VLAN ID. For the `vxlan` network types, the segmentation ID is VXLAN VNI. Segmentation IDs may not be used by all network types, and if not specified, may be automatically assigned by Neutron if required.

When specified, the `--external` option qualifies a network as a gateway network for a router. The network will serve as a floating IP network for attached instances. Networks are considered *internal* by default.

There are other optional network parameters that can be discovered using the `--help` flag shown here:

```
openstack network create --help
```

The `--help` flag can be appended to most commands within the OpenStack command-line utility, and will be helpful when constructing commands throughout this chapter.

Subnets are created with the following syntax:

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

Creating a subnet creates a logical layer 3 routing domain, whose details are used to provide IP services to virtual machines and other virtual network objects. The `network` parameter maps the subnet to a layer 2 network defined in OpenStack. The `subnet-range` parameter defines the L3 address range used by the subnet and is written in CIDR notation. More than one subnet can be associated with a network, which is often the case when all addresses in a particular subnet have been consumed. While logically separated, multiple subnets in a network are all part of the same layer 2 broadcast domain.

When a network and subnet are created in OpenStack, and DHCP is enabled, a corresponding network namespace is created on one or more nodes running the DHCP agent. The namespace can be identified using the `ip netns` command shown as follows:

```
# ip netns list
...
qdhcp-c881ce20-1649-4f03-bea7-40da536e21b2
...
```

A DHCP namespace has a prefix of `qdhcp-` and a suffix that corresponds to the network ID.

The `ip netns` command must be run by the `root` user or a user with `sudo` permissions.

Creating tenant networks

When a tenant network is created in OpenStack, provider attributes that describe how the network is connected to the physical infrastructure are automatically determined by Neutron based on settings hard-coded in configuration files. Typically, tenant networks are created and managed by users within a particular project and are not shared with other projects.

Getting ready

You will need the following details, at a minimum, for the network:

- Network name

For our example, the following network name will be used:

- Network name: `COOKBOOK_TENANT_NET_1`

You will need the following details, at a minimum, for the corresponding subnet:

- Subnet name
- Network name or ID
- Subnet range (CIDR)

For our example, the following will be used:

- Subnet name: `COOKBOOK_TENANT_SUBNET_1`
- Network name or ID: `COOKBOOK_TENANT_NET_1`
- Subnet range (CIDR): `172.16.200.0/24`

How to do it...

With the `openstack` client installed on our system, we are now able to create a tenant network with the following steps:

1. Create the network:

   ```
   openstack network create COOKBOOK_TENANT_NET_1
   ```

   The output will resemble the following:

```
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2017-10-12T11:26:37Z                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | 1a1a4e20-5d02-40aa-b534-7f76cbeb77a8 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | False                                |
| is_vlan_transparent       | None                                 |
| mtu                       | 1450                                 |
| name                      | COOKBOOK_TENANT_NET_1                |
| port_security_enabled     | True                                 |
| project_id                | 4819f952f3d0411183956113f0727b30     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 40                                   |
| qos_policy_id             | None                                 |
| revision_number           | 2                                    |
| router:external           | Internal                             |
| segments                  | None                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| tags                      |                                      |
| updated_at                | 2017-10-12T11:26:38Z                 |
+---------------------------+--------------------------------------+
```

As a non-admin user, certain network attributes may not be visible.

2. Create the subnet:

```
openstack subnet create COOKBOOK_TENANT_SUBNET_1 \
--network COOKBOOK_TENANT_NET_1 \
--subnet-range 172.16.200.0/24
```

The output will resemble the following:

```
+-------------------------+--------------------------------------+
| Field                   | Value                                |
+-------------------------+--------------------------------------+
| allocation_pools        | 172.16.200.2-172.16.200.254          |
| cidr                    | 172.16.200.0/24                      |
| created_at              | 2017-10-12T11:28:43Z                 |
| description             |                                      |
| dns_nameservers         |                                      |
| enable_dhcp             | True                                 |
| gateway_ip              | 172.16.200.1                         |
| host_routes             |                                      |
| id                      | eb7ccbbe-aac5-4c65-8f49-6d82bf806e15 |
| ip_version              | 4                                    |
| ipv6_address_mode       | None                                 |
| ipv6_ra_mode            | None                                 |
| name                    | COOKBOOK_TENANT_SUBNET_1             |
| network_id              | 1a1a4e20-5d02-40aa-b534-7f76cbeb77a8 |
| project_id              | 4819f952f3d0411183956113f0727b30     |
| revision_number         | 0                                    |
| segment_id              | None                                 |
| service_types           |                                      |
| subnetpool_id           | None                                 |
| tags                    |                                      |
| updated_at              | 2017-10-12T11:28:43Z                 |
| use_default_subnet_pool | None                                 |
+-------------------------+--------------------------------------+
```

How it works...

Tenant networks are created with the following syntax:

```
openstack network create NETWORK_NAME
```

When created as a non-admin user, a network's provider attributes are automatically determined by Neutron based on settings defined in the respective network plugin configuration files. Tenant networks are associated with the project that created them, and by default, are not visible or usable by other projects.

Neutron **role-based access control** (**RBAC**) can be used to share networks with other projects if desired. More information on using RBAC in the Pike release of OpenStack can be found at the following website:

`https://docs.openstack.org/neutron/pike/admin/config-rbac.html`

Creating ports

Ports in OpenStack can be created using the `openstack port create` command. Ports are automatically created by OpenStack upon server creation or can be created and attached to instances at a later time. Users may also create ports as a method of reserving IP addresses for later use or to avoid certain addresses from being allocated by OpenStack at all.

Getting ready

You will need the following details, at a minimum, for the port:

- Network name or ID
- Port name

For our example, the following will be used:

- Network name: `COOKBOOK_TENANT_NET_1`
- Port name: `COOKBOOK_TEST_PORT_1`

How to do it...

With the OpenStack client installed on our system, we are now able to create a port with the following command:

```
openstack port create COOKBOOK_TEST_PORT_1 \
--network COOKBOOK_TENANT_NET_1
```

The output will resemble the following:

Field	Value
admin_state_up	UP
allowed_address_pairs	
binding_host_id	
binding_profile	
binding_vif_details	
binding_vif_type	unbound
binding_vnic_type	normal
created_at	2017-10-12T11:30:06Z
data_plane_status	None
description	
device_id	
device_owner	
dns_assignment	None
dns_name	None
extra_dhcp_opts	
fixed_ips	ip_address='172.16.200.7', subnet_id='eb7ccbbe-aac5-4c65-8f49-6d82bf806e15'
id	5e9dde89-e498-4c63-83da-331e96b7fbd5
ip_address	None
mac_address	fa:16:3e:6c:91:83
name	COOKBOOK_TEST_PORT_1
network_id	1a1a4e20-5d02-40aa-b534-7f76cbeb77a8
option_name	None
option_value	None
port_security_enabled	True
project_id	4819f952f3d0411183956113f0727b30
qos_policy_id	None
revision_number	3
security_group_ids	42231729-317e-4512-8996-635437b9adca
status	DOWN
subnet_id	None
tags	
trunk_details	None
updated_at	2017-10-12T11:30:06Z

How it works...

When a port is created in OpenStack and associated with an instance or other virtual network device, it is bound to a Neutron agent on the respective node hosting the instance or device. Using details provided by the port, OpenStack services may construct a **virtual machine interface** (**vif**) or **virtual ethernet interface** (**veth**) on the host for use with a virtual machine, network namespace, or more depending on the application.

Updating network attributes

Network attributes can be updated using the `openstack network set` and `openstack network unset` commands.

Getting ready

When updating a network, ensure that you are authenticated as an administrator or are the owner of the network. You will need the following details:

- Network name or ID
- Attribute to update

For our example, the following will be used:

- Network name: `COOKBOOK_PROVIDER_NET`
- Attribute to update: `router:external`

How to do it...

With the OpenStack client installed on our system, we are now able to update the network with the following command:

```
openstack network set COOKBOOK_PROVIDER_NET --external
```

No output is returned.

How it works...

Networks are updated with the following syntax:

```
openstack network set NETWORK \
[--share | --no-share] \
[--description <description>] \
[--external | --internal]

openstack network unset [--tag <tag> | --all-tag] NETWORK
```

The `share` and `no-share` parameters dictate whether the network can be shared among projects or is limited to the owner of the network.

The `description` parameter allows users to provide a useful description of the network.

When specified, the `--external` option qualifies a network as a gateway network for a router. The network will serve as a floating IP network for attached instances. Networks are considered internal by default.

Not all network plugins support updating certain network attributes for existing networks. If a change is required, the network may need to be deleted and recreated.

Deleting ports

Ports in OpenStack can be deleted with the `openstack port delete` command. OpenStack automatically deletes ports it creates, such as when a server or floating IP is created, but may not delete ports created by users and associated with instances at a later time.

Deleting ports associated with active instances may cause the instance to crash or can cause unexpected behavior in the instance.

Getting ready

When deleting a port, ensure that you are authenticated as an administrator or are the owner of the port. You will need the following details:

- Port name or ID

For our example, the following will be used:

- Port name: COOKBOOK_TEST_PORT_1

How to do it...

With the OpenStack client installed on our system, we are now able to delete a port with the following command:

```
openstack port delete COOKBOOK_TEST_PORT_1
```

No output is returned.

Deleting networks

Deleting a network in OpenStack is as easy as invoking the `openstack network delete` command. Neutron will delete any automatically-created ports associated with the network, like the ones created by/for the DHCP or router namespaces, and will return any automatically-assigned segmentation IDs to the respective pool for allocation to another network.

Getting ready

When deleting a network in OpenStack, ensure that all associated user-created ports have been deleted. This may require deleting instances, detaching and deleting ports from instances, or detaching and deleting ports from routers. When deleting a network, the following information will be necessary:

- Network name or ID

How to do it...

With the OpenStack client installed on our system, we are able now to delete a network with the following command:

```
openstack network delete COOKBOOK_TENANT_NETWORK_3
```

No output is returned.

How it works...

When invoked, the `openstack network delete` command will instruct Neutron to delete the specified network and any associated subnet(s), as long as all user-managed ports have been deleted. In a reference architecture, the layer 2 agents are responsible for deleting the respective virtual bridges and interfaces configured on the hosts, and all records of the network are purged from the OpenStack database.

Neutron does not maintain network information in the database once those objects have been deleted. Requests against the OpenStack API may be logged, but using third-party tools or proxies is highly recommended if an audit trail is required.

Managing routers and floating IPs

A **router** in OpenStack represents a virtual routing device that provides routing capabilities to directly connected networks. To provide end-to-end connectivity to a virtual machine, a router must be connected to an external provider network and the tenant network where the instance resides. Typically, routers are created and managed by individual projects. By default, external provider networks are shared and available for use by all projects. The following diagram represents an external provider network owned by the `ADMIN` project and utilized by three other projects:

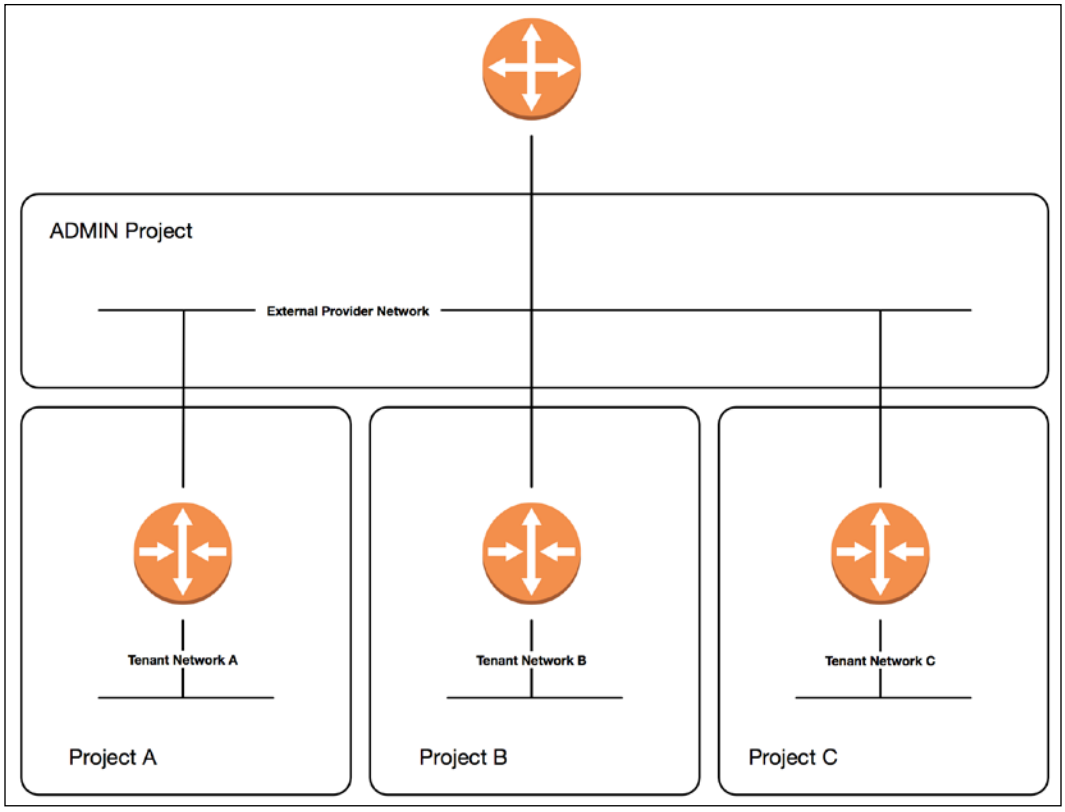


Figure 4.2

In *Figure 4.2*, three projects have routers connected to an external provider network. The external provider network not only provides connectivity to the routers and the networks behind them, but also provides a network from which floating IPs can be derived. Floating IPs provide 1-to-1 address translations that allow external clients to connect directly to instances.

Creating routers

Routers in OpenStack can be created using the `openstack router create` command. By default, routers are considered *internal* and can route only between directly-connected tenant networks. An *external* router, on the other hand, is capable of routing to an external gateway and can provide **network address translation** (**NAT**) services to connected networks.

There are three types of routers that can be created in an OpenStack reference architecture:

- Standalone
- **Highly-Available** (**H**A)
- Distributed (DVR)

Standalone routers do not provide any level of resiliency, while highly available routers implement VRRP to provide redundancy should one or more Neutron nodes fail. Distributed virtual routers reside on compute nodes rather than on centralized network nodes, and provide higher performance than their counterparts for east/west traffic, or traffic between instances in different networks, since that traffic is forwarded between compute nodes without having to traverse a network node.

For non-admin users, the type of router that is created with the `openstack router create` command is determined automatically by settings defined in Neutron configuration files. Only users with administrator-level permissions can specify router types when creating routers.

Getting ready

You will need the following details, at a minimum, for the router:

- Router name

For our examples, the following will be used:

- Router name: `COOKBOOK_ROUTER_STANDALONE`
- Router name: `COOKBOOK_ROUTER_HA`
- Router name: `COOKBOOK_ROUTER_DVR`

You will need the following details, at a minimum, for networks attached to the router:

- External provider network name or ID
- Tenant subnet name or ID

For our example, the following will be used:

- External provider network name: `COOKBOOK_PROVIDER_NET`
- Tenant subnet name: `COOKBOOK_TENANT_SUBNET`

How to do it...

With the `openstack` client installed on our system, we are now able to create a router with the following steps:

1. Create the standalone router:

   ```
   openstack router create COOKBOOK_ROUTER_STANDALONE
   ```

   The output will resemble the following:

   ```
   +-------------------------+--------------------------------------+
   | Field                   | Value                                |
   +-------------------------+--------------------------------------+
   | admin_state_up          | UP                                   |
   | availability_zone_hints |                                      |
   | availability_zones      |                                      |
   | created_at              | 2017-10-12T11:33:17Z                 |
   | description             |                                      |
   | distributed             | False                                |
   | external_gateway_info   | None                                 |
   | flavor_id               | None                                 |
   | ha                      | False                                |
   | id                      | 058ea6a3-ca86-46f8-80c8-d63bbc195212 |
   | name                    | COOKBOOK_ROUTER_STANDALONE           |
   | project_id              | 4819f952f3d0411183956113f0727b30     |
   | revision_number         | None                                 |
   | routes                  |                                      |
   | status                  | ACTIVE                               |
   | tags                    |                                      |
   | updated_at              | 2017-10-12T11:33:17Z                 |
   +-------------------------+--------------------------------------+
   ```

2. Create an HA router:

   ```
   openstack router create COOKBOOK_ROUTER_HA --ha
   ```

   The output will resemble the following:

   ```
   +-------------------------+--------------------------------------+
   | Field                   | Value                                |
   +-------------------------+--------------------------------------+
   | admin_state_up          | UP                                   |
   | availability_zone_hints |                                      |
   | availability_zones      |                                      |
   | created_at              | 2017-10-12T12:04:58Z                 |
   | description             |                                      |
   | distributed             | False                                |
   | external_gateway_info   | None                                 |
   | flavor_id               | None                                 |
   | ha                      | True                                 |
   | id                      | 90fa4310-8ead-4f8c-b867-8395ff089928 |
   | name                    | COOKBOOK_ROUTER_HA                   |
   | project_id              | 4819f952f3d0411183956113f0727b30     |
   | revision_number         | None                                 |
   | routes                  |                                      |
   | status                  | ACTIVE                               |
   | tags                    |                                      |
   | updated_at              | 2017-10-12T12:04:58Z                 |
   +-------------------------+--------------------------------------+
   ```

3. Create a distributed router:

```
openstack router create COOKBOOK_ROUTER_DVR --distributed
```

The output will resemble the following:

Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2017-10-12T12:06:33Z
description	
distributed	True
external_gateway_info	None
flavor_id	None
ha	False
id	c4dd4dbc-dfec-4a73-bc6d-e5df473699e9
name	COOKBOOK_ROUTER_DVR
project_id	4819f952f3d0411183956113f0727b30
revision_number	None
routes	
status	ACTIVE
tags	
updated_at	2017-10-12T12:06:33Z

How it works...

When *any* router is created in OpenStack using the native routing services, a corresponding network namespace is created on one or more nodes running the `neutron-l3-agent` service. The namespace can be identified using the `ip netns` command shown here:

```
# ip netns list
...
qrouter-058ea6a3-ca86-46f8-80c8-d63bbc195212
...
```

A router namespace has a prefix of `qrouter-` and a suffix that corresponds to the router ID. In environments where distributed virtual routers are configured, other namespaces needed to facilitate proper networking may exist, such as the `fip-` and `snat-` namespaces.

By default, a non-admin user cannot specify the type of router being created. The router type is automatically determined by Neutron based on the layer 3 agent configuration file. The author recommends HA or distributed virtual routers, where possible, to limit the impact of failure for physical nodes hosting virtual routers.

Cloud operators can modify Neutron configuration files to change default behaviors. The `policy.json` file used by the Neutron API service can be modified to allow users and roles to perform actions usually reserved for administrators. Refer to community documentation for guidelines and caveats related to modifications to the `policy.json` files.

Attaching networks to routers

Routers in OpenStack can connect to a single external provider network and one or more tenant networks, as shown in the following diagram:

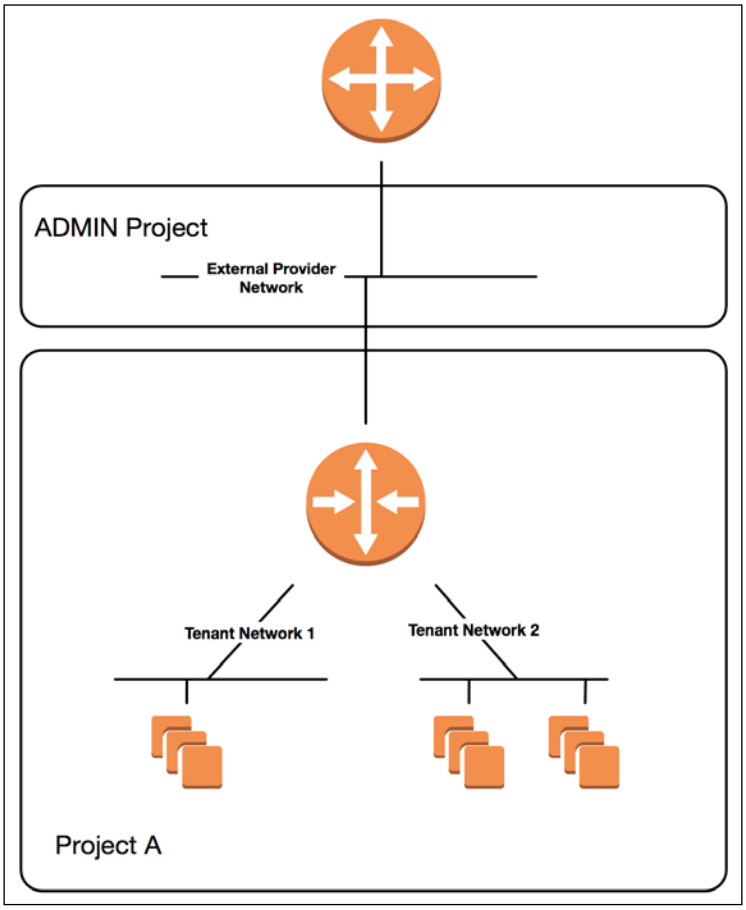


Figure 4.3

In *Figure 4.3*, an external provider network provides external connectivity, while tenant networks provide connectivity to virtual machines and other virtual network devices within a project. The router's job is to facilitate end-to-end connectivity using routes and sometimes the NAT via floating IPs.

The commands necessary to attach a network to a router may vary based on network and need:

- Attaching a router to an external network:

  ```
  openstack router set --external-gateway <network> <router>
  ```

- Attaching a router to a tenant network using subnet ID or name:

  ```
  openstack router add subnet <router> <subnet>
  ```

- Attaching a router to a tenant network using a specific port ID or name:

  ```
  openstack router add port <router> <port>
  ```

Getting ready

When attaching a router to a network, the following information will be necessary:

- Router name or ID
- Network name or ID, or Subnet name or ID, or Port name or ID

Remember, when attaching a router to an external provider network, the network's `router:external` attribute must be set to `External` or `True`.

How to do it...

To attach a network to a router in OpenStack, follow these steps:

1. Attach the router to the external provider network `COOKBOOK_PROVIDER_NET`:

   ```
   openstack router set --external-gateway COOKBOOK_PROVIDER_NET
   COOKBOOK_ROUTER_STANDALONE
   ```

   No output is provided.

2. Attach the router to the `COOKBOOK_TENANT_SUBNET` subnet using the subnet name:

   ```
   openstack router add subnet COOKBOOK_ROUTER_STANDALONE COOKBOOK_
   TENANT_SUBNET_1
   ```

   No output is provided.

How it works...

When routers are attached to external provider networks, the router is assigned an IP address from the pool of addresses available for allocation from the network. The router is also configured with a default gateway that corresponds to the specified gateway for the respective provider subnet.

When a router is attached to a tenant network, the router becomes the gateway for the attached network and all instances within it. It is assigned the IP address specified as the gateway for the respective tenant subnet.

Ports attached to routers can be listed using the following command:

```
openstack port list --router ROUTER_NAME_OR_ID
```

In this example, the following ports and corresponding subnets have been attached to the router:

```
root@controller-01-utility-container-e2bad038:~# openstack port list --router COOKBOOK_ROUTER_STANDALONE
+--------------------------------------+------+-------------------+-------------------------------------------------------------------------------+--------+
| ID                                   | Name | MAC Address       | Fixed IP Addresses                                                            | Status |
+--------------------------------------+------+-------------------+-------------------------------------------------------------------------------+--------+
| c69005cb-aa86-4fe7-81aa-64ea265863b4 |      | fa:16:3e:38:03:1c | ip_address='172.16.200.1', subnet_id='eb7ccbbe-aac5-4c65-8f49-6d82bf806e15'   | ACTIVE |
| ed006ed1-b832-41ab-b08c-7fc1ee4da10c |      | fa:16:3e:ed:6c:e7 | ip_address='192.168.200.3', subnet_id='83b8a728-4914-4667-af19-bd8b798a2e25'  | ACTIVE |
+--------------------------------------+------+-------------------+-------------------------------------------------------------------------------+--------+
```

Using the `ip netns exec` command, coupled with the name of the respective router namespace, we can see the router has two attached interfaces, `qg-ed006ed1-b8` and `qr-c69005cb-aa`:

```
root@controller-01-neutron-agents-container-21200603:~# ip netns exec qrouter-058ea6a3-ca86-46f8-80c8-d63bbc195212 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: qg-ed006ed1-b8@if12: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:ed:6c:e7 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.168.200.3/24 brd 192.168.200.255 scope global qg-ed006ed1-b8
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:feed:6ce7/64 scope link
       valid_lft forever preferred_lft forever
3: qr-c69005cb-aa@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default qlen 1000
    link/ether fa:16:3e:38:03:1c brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.16.200.1/24 brd 172.16.200.255 scope global qr-c69005cb-aa
       valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe38:31c/64 scope link
       valid_lft forever preferred_lft forever
```

The names of the interfaces in the router namespace correspond to the first 10 characters of the respective port ID. External, or gateway-side, interfaces are prefixed with `qg-`, while internal, or router-side, interfaces are prefixed with `qr-`. The naming scheme dates back to when the Neutron project was known as Quantum.

Creating and assigning floating IPs

Floating IPs in OpenStack are static IPv4 addresses that are mapped to instances behind Neutron routers and provide direct inbound connectivity to those instances. Floating IPs can be used in a similar fashion to Elastic IPs in Amazon Web Services, where users can quickly remap an IP address from one instance to another in the event of failure. The heart of this functionality is network address translation. A floating IP is usually considered an *external* address that is mapped to the *internal* address configured on an instance. The NAT is implemented on the Neutron router connected to the instance's network. Floating IPs offer connectivity to instances that would otherwise be isolated behind a Neutron router on a non-routable network.

Getting ready

Recall that instances are connected to ports that reflect the connected network and associated IP address. When creating a floating IP, the following information is required:

- External network name or ID

When assigning a floating IP to a port, the following is necessary:

- Floating IP ID
- Internal port name or ID

For this example, create a new port in the existing tenant network named `COOKBOOK_TEST_PORT_2`:

```
openstack port create COOKBOOK_TEST_PORT_2 \
--network COOKBOOK_TENANT_NET_1
```

How to do it...

To create a floating IP in OpenStack, issue the following command:

```
openstack floating ip create --port COOKBOOK_TEST_PORT_2 COOKBOOK_
PROVIDER_NET
```

The output will resemble the following:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| created_at          | 2017-10-12T13:03:43Z                 |
| description         |                                      |
| fixed_ip_address    | 172.16.200.11                        |
| floating_ip_address | 192.168.200.13                       |
| floating_network_id | c881ce20-1649-4f03-bea7-40da536e21b2 |
| id                  | e9659231-1820-464a-a154-a949bb332821 |
| name                | 192.168.200.13                       |
| port_id             | 122dae7b-0340-42b5-856a-3e8e19f731e1 |
| project_id          | 4819f952f3d0411183956113f0727b30     |
| revision_number     | 0                                    |
| router_id           | 058ea6a3-ca86-46f8-80c8-d63bbc195212 |
| status              | DOWN                                 |
| updated_at          | 2017-10-12T13:03:43Z                 |
+---------------------+--------------------------------------+
```

How it works...

Floating IPs are created with the following syntax:

```
openstack floating ip create EXTERNAL_NETWORK_NAME_OR_ID \
[--port PORT_NAME_OR_ID]
```

Floating IPs can then be associated with a port, using the following syntax:

```
openstack floating ip associate FLOATING_IP_NAME_OR_ID \
PORT_NAME_OR_ID
```

When a floating IP is associated with a port, Neutron uses the port information to determine which router to configure the NAT on. Once the NAT is in place, connections to the floating IP will be translated to the internal IP and forwarded to the respective instance. Responses from the instance will be forwarded to the router, translated from the internal to the floating IP, and routed back out to the origin.

In our example, the instance's port is associated with the `COOKBOOK_TENANT_NET` network, which in turn is connected to the `COOKBOOK_ROUTER_STANDALONE` router. Within the respective `qrouter` network namespace, we can see the source and destination NATs applied using `iptables`:

```
root@controller-01-neutron-agents-container-21200603:~# ip netns exec qrouter-058ea6a3-ca86-46f8-80c8-d63bbc195212 iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
neutron-l3-agent-PREROUTING  all  --  anywhere             anywhere

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
neutron-l3-agent-OUTPUT  all  --  anywhere             anywhere

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
neutron-l3-agent-POSTROUTING  all  --  anywhere             anywhere
neutron-postrouting-bottom  all  --  anywhere             anywhere

Chain neutron-l3-agent-OUTPUT (1 references)
target     prot opt source               destination
DNAT       all  --  anywhere             192.168.200.13       to:172.16.200.11

Chain neutron-l3-agent-POSTROUTING (1 references)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere             ! ctstate DNAT

Chain neutron-l3-agent-PREROUTING (1 references)
target     prot opt source               destination
REDIRECT   tcp  --  anywhere             169.254.169.254      tcp dpt:http redir ports 9697
DNAT       all  --  anywhere             192.168.200.13       to:172.16.200.11

Chain neutron-l3-agent-float-snat (1 references)
target     prot opt source               destination
SNAT       all  --  172.16.200.11        anywhere             to:192.168.200.13

Chain neutron-l3-agent-snat (1 references)
target     prot opt source               destination
neutron-l3-agent-float-snat  all  --  anywhere             anywhere
SNAT       all  --  anywhere             anywhere             to:192.168.200.3
SNAT       all  --  anywhere             anywhere             mark match ! 0x2/0xffff ctstate DNAT to:192.168.200.3

Chain neutron-postrouting-bottom (1 references)
target     prot opt source               destination
neutron-l3-agent-snat  all  --  anywhere             anywhere             /* Perform source NAT on outgoing traffic. */
```

Used to translate floating IP to fixed IP for traffic heading inbound from external provider network to instance/port

Used to translate fixed IP to floating IP for traffic heading outbound from instance/port in tenant network

Used to translate all other outbound traffic from instances/ports in tenant network

Deleting routers

Routers in OpenStack can be deleted using the `openstack router delete` command. Before a router can be deleted, all subnets/ports attached to the router must be detached. This will cause a disruption of traffic for any instance connected to a network behind the router. Subnets can be detached using the `openstack router remove subnet` or `openstack router remove port` commands.

Getting ready

When deleting a router, the following information will be necessary:

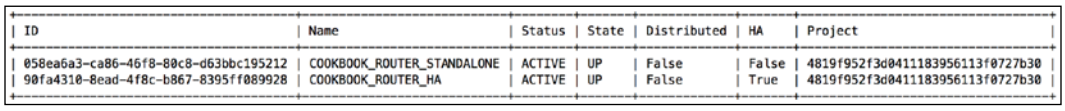
- Router name or ID

How to do it...

To delete a router in OpenStack, issue the following command:

```
openstack router delete COOKBOOK_ROUTER_DVR
```

No output is given. However, the operation can be verified using the `openstack router list` command:



ID	Name	Status	State	Distributed	HA	Project
058ea6a3-ca86-46f8-80c8-d63bbc195212	COOKBOOK_ROUTER_STANDALONE	ACTIVE	UP	False	False	4819f952f3d0411183956113f0727b30
90fa4310-8ead-4f8c-b867-8395ff089928	COOKBOOK_ROUTER_HA	ACTIVE	UP	False	True	4819f952f3d0411183956113f0727b30

Deleted routers will no longer appear in the list.

How it works...

When invoked, the `openstack router delete` command will instruct Neutron to delete the specified router and associated resources. In a reference architecture, the layer 3 agents are responsible for deleting the respective network namespace(s) and virtual interfaces configured on the hosts, and all records of the router are purged from the OpenStack database.

Managing security groups

In OpenStack, a security group describes a grouping of ports of similar security requirements. Security group rules are associated with security groups, and provide ingress and egress filtering capabilities to the group. Security group rules can reference other groups or remote networks using CIDR notation. The actual filtering takes place on the compute node at the "port" level, and may be implemented using iptables or as openflow rules depending on the firewall driver that is configured on a given node. Newly created projects each contain a security group named `default` that allows egress, or outbound, communication only. Ingress, or inbound, communication is denied.

Creating security groups

Security groups in OpenStack can be created using the `openstack security group create` command. Security groups are project-owned objects and cannot be shared or referenced by other projects.

Getting ready

When creating a security group, each port associated with the group will inherit the rules applied to the group. You will need the following details, at a minimum, for the group:

- Security group name

For our example, the following will be used:

- Security group name: `COOKBOOK_SG_WEB`

How to do it...

With the OpenStack client installed on our system, we are now able to create a security group with the following command:

```
openstack security group create COOKBOOK_SG_WEB
```

The output will resemble the following:

```
+-----------------+--------------------------------------------------------------------------------------------------------------------------------------------------+
| Field           | Value                                                                                                                                            |
+-----------------+--------------------------------------------------------------------------------------------------------------------------------------------------+
| created_at      | 2017-10-12T13:34:49Z                                                                                                                             |
| description     | COOKBOOK_SG_WEB                                                                                                                                  |
| id              | aafe2ee4-9958-4859-a2b3-2fac65650fee                                                                                                             |
| name            | COOKBOOK_SG_WEB                                                                                                                                  |
| project_id      | 4819f952f3d0411183956113f0727b30                                                                                                                 |
| revision_number | 2                                                                                                                                                |
| rules           | created_at='2017-10-12T13:34:49Z', direction='egress', ethertype='IPv4', id='6057b795-92c6-41f6-9c7f-6cd4d676223c', updated_at='2017-10-12T13:34:49Z' |
|                 | created_at='2017-10-12T13:34:49Z', direction='egress', ethertype='IPv6', id='86fbdb0a-ff65-42f6-8983-3785a95f08bf', updated_at='2017-10-12T13:34:49Z' |
| updated_at      | 2017-10-12T13:34:49Z                                                                                                                             |
+-----------------+--------------------------------------------------------------------------------------------------------------------------------------------------+
```

How it works...

Security groups are created with the following syntax:

```
openstack security group create SECURITY_GROUP_NAME
```

When a security group is created, OpenStack applies a default set of rules to the group that allows a port to communicate outbound (egress) over IPv4 and IPv6. By default, all inbound traffic is denied.

Creating a security group is only the first step in providing filtering to instances. The next steps, namely creating security group rules and applying security groups to ports, will be discussed in the following sections.

Creating security group rules

Security group rules in OpenStack can be created using the `openstack security group rule create` command. A security group rule provides information on filtering at a layer 3 and layer 4 level, which includes IP addresses and destination ports.

Getting ready

When creating a security group rule, remember that each port associated with the group will inherit the rules applied to the group. Therefore, it is important to limit the rules to only those needed to provide the desired access to the associated group. You will need the following details, at a minimum, for the rule:

- Security group name

For our example, the following will be used:

- Security group name: `COOKBOOK_SG_WEB`
- Destination port: `80`
- Protocol: TCP
- Direction: Ingress
- Source: All Addresses

How to do it...

With the OpenStack client installed on our system, we are now able to create a security group rule with the following command:

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

The output will resemble the following:

```
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| created_at        | 2017-10-12T13:37:48Z                 |
| description       |                                      |
| direction         | ingress                              |
| ether_type        | IPv4                                 |
| id                | 08fd181f-d19f-49ca-b2d9-35b3f5acc574 |
| name              | None                                 |
| port_range_max    | 80                                   |
| port_range_min    | 80                                   |
| project_id        | 4819f952f3d0411183956113f0727b30     |
| protocol          | tcp                                  |
| remote_group_id   | None                                 |
| remote_ip_prefix  | 0.0.0.0/0                            |
| revision_number   | 0                                    |
| security_group_id | aafe2ee4-9958-4859-a2b3-2fac65650fee |
| updated_at        | 2017-10-12T13:37:48Z                 |
+-------------------+--------------------------------------+
```

How it works...

Security groups are created with the following syntax:

```
openstack security group rule create SECURITY_GROUP_NAME \
[--remote-ip <ip-address> | --remote-group <group>] \
[--dst-port <port-range>] \
[--icmp-type <icmp-type>] \
[--icmp-code <icmp-code>] \
[--protocol <protocol>] \
[--ingress | --egress] \
[--ethertype <ethertype>]
```

Security group rules provide details to OpenStack that are used by Neutron agents to implement traffic filters on individual ports on compute nodes. In our environment, these rules are implemented as iptables rules.

Creating a security group rule without providing any additional filtering information may result in a rule that allows ingress access from *all* source addresses. OpenStack assumes certain defaults when details are not specified, so it is important to verify the provided output to ensure the proper filtering is in place.

Applying security groups to instances

Security groups are typically applied to instances upon boot, using the `openstack server create` command, but they can also be applied to individual ports, using the `openstack network port create` or `set` commands. When applied at boot, each listed security group is applied to each port associated with the instance. As a result, unnecessary rules may be applied to each interface that could result in a security risk or regression in performance of the underlying Neutron agent responsible for applying the rules. When applied to individual ports, users can ensure that if an instance is multihomed, the respective port has only the necessary rules needed to provide access to that interface.

The term multihomed refers to an instance that is connected to multiple networks via different interfaces.

Getting ready

When applying security groups to instances at boot, you will need the following information:

- Security group name or ID

When applying security groups to individual ports, you will need the following information:

- Port name or ID
- Security group name or ID

For our examples, the following will be used:

- Port name: `COOKBOOK_TEST_PORT_1`
- Security group name: `COOKBOOK_SG_WEB`
- Instance name: `COOKBOOK_INSTANCE_WEB`
- Instance flavor: `COOKBOOK_FLAVOR_TINY`
- Instance image: `COOKBOOK_IMAGE_CIRROS`
- Network name: `COOKBOOK_TENANT_NET_1`

How to do it...

With the `openstack` client installed on our system we are able to apply security groups with the following steps:

1. Create an instance and supply the security group at boot:

```
openstack server create COOKBOOK_INSTANCE_WEB \
--flavor COOKBOOK_FLAVOR_TINY \
```

```
--image COOKBOOK_IMAGE_CIRROS \
--nic net-id=COOKBOOK_TENANT_NET_1 \
--security-group COOKBOOK_SG_WEB
```

Use flavors and images that are appropriate for your environment, or those that may have been created in *Chapter 5*, *Nova – OpenStack Compute* and *Chapter 6*, *Glance – OpenStack Image Service*.

Alternatively, security groups can be applied during port creation or to existing ports. With the OpenStack client installed on our system we are able to apply security groups with the following stes.

2. Apply a security group to a new port:

```
openstack port create COOKBOOK_TEST_PORT_3 \
--network COOKBOOK_TENANT_NET_1 \
--security-group COOKBOOK_SG_WEB
```

The output will resemble the following:

```
+-------------------------+-------------------------------------------------------------------------------+
| Field                   | Value                                                                         |
+-------------------------+-------------------------------------------------------------------------------+
| admin_state_up          | UP                                                                            |
| allowed_address_pairs   |                                                                               |
| binding_host_id         |                                                                               |
| binding_profile         |                                                                               |
| binding_vif_details     |                                                                               |
| binding_vif_type        | unbound                                                                       |
| binding_vnic_type       | normal                                                                        |
| created_at              | 2017-10-12T14:18:42Z                                                          |
| data_plane_status       | None                                                                          |
| description             |                                                                               |
| device_id               |                                                                               |
| device_owner            |                                                                               |
| dns_assignment          | None                                                                          |
| dns_name                | None                                                                          |
| extra_dhcp_opts         |                                                                               |
| fixed_ips               | ip_address='172.16.200.3', subnet_id='eb7ccbbe-aac5-4c65-8f49-6d82bf806e15'   |
| id                      | e1dfc164-3015-4b78-83ec-6a31ccd1b7cd                                          |
| ip_address              | None                                                                          |
| mac_address             | fa:16:3e:b0:8c:d3                                                             |
| name                    | COOKBOOK_TEST_PORT_3                                                          |
| network_id              | 1a1a4e20-5d02-40aa-b534-7f76cbeb77a8                                          |
| option_name             | None                                                                          |
| option_value            | None                                                                          |
| port_security_enabled   | True                                                                          |
| project_id              | 4819f952f3d0411183956113f0727b30                                              |
| qos_policy_id           | None                                                                          |
| revision_number         | 3                                                                             |
| security_group_ids      | aafe2ee4-9958-4859-a2b3-2fac65650fee                                          |
| status                  | DOWN                                                                          |
| subnet_id               | None                                                                          |
| tags                    |                                                                               |
| trunk_details           | None                                                                          |
| updated_at              | 2017-10-12T14:18:42Z                                                          |
+-------------------------+-------------------------------------------------------------------------------+
```

3. Apply the security group to an existing port:

```
openstack port set COOKBOOK_TEST_PORT_2 \
--security-group COOKBOOK_SG_WEB
```

No output is returned. However, the change can be confirmed by querying the security groups applied to the port using the `openstack port show` command:

```
openstack port show COOKBOOK_TEST_PORT_2 -c security_group_ids
```

The output will resemble the following:

```
+-------------------+--------------------------------------------------------------------------+
| Field             | Value                                                                    |
+-------------------+--------------------------------------------------------------------------+
| security_group_ids | 42231729-317e-4512-8996-635437b9adca, aafe2ee4-9958-4859-a2b3-2fac65650fee |
+-------------------+--------------------------------------------------------------------------+
```

How it works...

Security groups can be applied to instances at boot, ports at creation, or to existing ports. Neutron plugin agents on compute nodes are responsible for applying the respective filtering rules to the port on the host. Multiple security groups can be applied to a port in all scenarios.

Security groups can be applied at boot with the following syntax:

```
openstack server create INSTANCE_NAME \
...
--security-group SECURITY_GROUP_NAME
```

Security groups can also be applied to new ports with the following syntax:

```
openstack port create PORT_NAME \
--network NETWORK_NAME \
--security-group SECURITY_GROUP_NAME
```

Lastly, security groups can be applied to existing ports with the following syntax:

```
openstack port set PORT_NAME \
--security-group SECURITY_GROUP_NAME
```

Ports are bound to Neutron plugin agents that reside on hosts within the environment. For example, ports belonging to instances are bound to the agent on the respective compute node where the instance lives. This agent is responsible for setting up the virtual networking for the port. When a security group is applied to a port, the Neutron agent where the port is bound implements the filtering rules on the host. Other agents throughout the environment are notified that the group has changed and are updated accordingly.

Managing load balancers

Neutron includes a service known as **Load Balancing as-a-Service** (**LBaaS**), which provides users with the ability to create load balancers that balance traffic to applications deployed across instances in the cloud. In a reference architecture, Neutron relies on an open source load balancing package known as HAProxy to provide the load balancing functionality. Much like the Neutron L3 agent handles virtual routers and the DHCP agent handles virtual DHCP servers, the Neutron LBaaS agent handles the construction and configuration of virtual load balancers upon request.

LBaaS should not be confused with another load balancing project known as Octavia. Both provide similar load balancing functions, but only LBaaS is covered here.

There are three major components to a load balancer in OpenStack:

- Pool members
- Pools
- Listeners

A **pool member** describes a layer 4 object that is composed of the IP address and port of a service residing on an instance. For example, a pool member might be a web server with a configured address of `10.30.0.2` listening on TCP port `80`.

A **pool** is a group of pool members serving identical content.

A **listener** is an object that represents a **virtual IP** (**VIP**) and port that is listening on the load balancer itself. Traffic to the virtual IP will be balanced among the members of the associated pool.

Additional components, such as health monitors and L7 policies, help extend the usefulness and functionality of a load balancer, but are not required.

The workflow for creating a functioning load balancer is as follows:

- Create a load balancer object
- Create and associate listener
- Create and associate a pool
- Create and associate pool member(s)
- Create and associate health monitor(s) (optional)

Creating load balancers

As of the Pike release of OpenStack, load balancer-related commands are not available in the OpenStack client. Instead, the `neutron` client should be used. Load balancers in OpenStack can be created using the `neutron lbaas-loadbalancer-create` command.

Getting ready

You will need the following details, at a minimum, for the load balancer:

- VIP subnet

A name is also recommended. For our example, the following will be used:

- Name: `COOKBOOK_LOADBALANCER_1`
- VIP subnet: `COOKBOOK_TENANT_SUBNET_1`

How to do it...

With the Neutron client installed on our system, we are now able to create a load balancer object with the following command:

```
neutron lbaas-loadbalancer-create COOKBOOK_TENANT_SUBNET_1 \
--name COOKBOOK_LOADBALANCER_1
```

The output will resemble the following:

```
Created a new loadbalancer:
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| admin_state_up      | True                                 |
| description         |                                      |
| id                  | aa599cee-b49f-44f1-a0fd-51fa69ebf6db |
| listeners           |                                      |
| name                | COOKBOOK_LOADBALANCER_1              |
| operating_status    | OFFLINE                              |
| pools               |                                      |
| provider            | haproxy                              |
| provisioning_status | PENDING_CREATE                       |
| tenant_id           | 4819f952f3d0411183956113f0727b30     |
| vip_address         | 172.16.200.14                        |
| vip_port_id         | ea1b4c37-814b-4ac7-a31e-df1e694a1987 |
| vip_subnet_id       | eb7ccbbe-aac5-4c65-8f49-6d82bf806e15 |
+---------------------+--------------------------------------+
```

How it works...

Load balancers are created with the following syntax:

```
neutron lbaas-loadbalancer-create VIP_SUBNET \
[--name NAME]
```

When a load balancer is created, OpenStack assigns an IP address known as a virtual IP. The VIP will be used by clients to access the load-balanced application. Creating a load balancer object is only the first step in load balancing traffic to instances. The next steps, creating a listener, pool, and health monitor will be discussed in the following sections.

Creating pools

Pools that are associated with load balancers are objects that represent a collection of instances that receive traffic sent to the VIP. Load balancing pools in OpenStack can be created using the `neutron lbaas-pool-create` command.

Getting ready

You will need the following details, at a minimum, for the pool:

- Balancing algorithm
- Protocol
- Load balancer or listener

A name is also recommended. For our example, the following will be used:

- Name: `COOKBOOK_POOL_1`
- Balancing algorithm: `ROUND_ROBIN`
- Protocol: `HTTP`
- Load balancer: `COOKBOOK_LOADBALANCER_1`

A load balancer can be associated with multiple listeners, which in turn can be associated with their own respective pool. A common scenario for this type of set up would be a load balancer with a listener on port `80` and another on port `443`, each with their respective backend pool.

How to do it...

With the Neutron client installed on our system, we are now able to create a load balancer pool with the following command:

```
neutron lbaas-pool-create --lb-algorithm ROUND_ROBIN \
--protocol HTTP \
--loadbalancer COOKBOOK_LOADBALANCER_1 \
--name COOKBOOK_POOL_1
```

The output will resemble the following:

```
Created a new pool:
+---------------------+--------------------------------------------------+
| Field               | Value                                            |
+---------------------+--------------------------------------------------+
| admin_state_up      | True                                             |
| description         |                                                  |
| healthmonitor_id    |                                                  |
| id                  | 0eddb485-af4b-4cf5-8d2f-4eaa13466059             |
| lb_algorithm        | ROUND_ROBIN                                      |
| listeners           |                                                  |
| loadbalancers       | {"id": "aa599cee-b49f-44f1-a0fd-51fa69ebf6db"}   |
| members             |                                                  |
| name                | COOKBOOK_POOL_1                                  |
| protocol            | HTTP                                             |
| session_persistence |                                                  |
| tenant_id           | 4819f952f3d0411183956113f0727b30                 |
+---------------------+--------------------------------------------------+
```

How it works...

Load balancer pools are created with the following syntax:

```
neutron lbaas-pool-create [--name NAME] \
--lb-algorithm {ROUND_ROBIN,LEAST_CONNECTIONS,SOURCE_IP} \
[--listener LISTENER | --loadbalancer LOADBALANCER] \
--protocol {HTTP,HTTPS,TCP}
```

Other load-balancing objects, such as members and monitors, reference pools and cannot be created without being applied to one at that time.

Creating members

Members are associated with pools, and are objects that represent a backend application listening on a particular IP and port. Pool members in OpenStack can be created using the `neutron lbaas-member-create` command.

Getting ready

You will need the following details, at a minimum, for the member:

- Subnet name
- IP address
- Port
- Pool name

A name is also recommended. For our example, the following will be used:

- Name: `COOKBOOK_MEMBER_1`
- Subnet name: `COOKBOOK_TENANT_SUBNET_1`
- IP Address: `172.16.200.11` (Corresponds to `COOKBOOK_TEST_PORT_2`)
- Port: `80`
- Pool name: `COOKBOOK_POOL_1`

A member can only be associated with a single pool. However, the same IP address and application port combination can be used for multiple members.

How to do it...

With the Neutron client installed on our system we are able to create a pool member with the following command:

```
neutron lbaas-member-create --name COOKBOOK_MEMBER_1 \
--subnet COOKBOOK_TENANT_SUBNET_1 \
--address 172.16.200.11 \
--protocol-port 80 \
COOKBOOK_POOL_1
```

The output will resemble the following:

```
Created a new member:
+----------------+--------------------------------------+
| Field          | Value                                |
+----------------+--------------------------------------+
| address        | 172.16.200.11                        |
| admin_state_up | True                                 |
| id             | dbf61f40-e9cd-435a-9a86-f3c8b537f1bb |
| name           | COOKBOOK_MEMBER_1                    |
| protocol_port  | 80                                   |
| subnet_id      | eb7ccbbe-aac5-4c65-8f49-6d82bf806e15 |
| tenant_id      | 4819f952f3d0411183956113f0727b30     |
| weight         | 1                                    |
+----------------+--------------------------------------+
```

How it works...

Pool members are created with the following syntax:

```
neutron lbaas-member-create [--name NAME] \
--subnet SUBNET --address ADDRESS \
--protocol-port PROTOCOL_PORT \
POOL
```

Creating listeners

Listeners are associated with load balancer objects, and they describe the relationship between the load balancer VIP and the port a service is listening on. Clients send traffic to the listener address and port, which is then proxied and sent to one member within pool of servers. Each listener can be configured to send traffic to a different pool. Listeners in OpenStack can be created using the `neutron lbaas-listener-create` command.

Getting ready

You will need the following details, at a minimum, for the listener:

- Load balancer name
- Protocol
- Port

A name and default pool are also recommended. For our example, the following will be used:

- Name: `COOKBOOK_LISTENER_1`
- Load balancer name: `COOKBOOK_LOADBALANCER_1`

- Protocol: `HTTP`
- Port: `80`
- Default pool: `COOKBOOK_POOL_1`

How to do it...

With the Neutron client installed on our system, we are now able to create a listener with the following command:

```
neutron lbaas-listener-create --name COOKBOOK_LISTENER_1 \
--loadbalancer COOKBOOK_LOADBALANCER_1 \
--protocol HTTP \
--protocol-port 80 \
--default-pool COOKBOOK_POOL_1
```

The output will resemble the following:

```
Created a new listener:
+---------------------------+------------------------------------------------+
| Field                     | Value                                          |
+---------------------------+------------------------------------------------+
| admin_state_up            | True                                           |
| connection_limit          | -1                                             |
| default_pool_id           | 0eddb485-af4b-4cf5-8d2f-4eaa13466059           |
| default_tls_container_ref |                                                |
| description               |                                                |
| id                        | 6f31848a-e55e-4ada-8a14-38cbb2e89cb1           |
| loadbalancers             | {"id": "aa599cee-b49f-44f1-a0fd-51fa69ebf6db"} |
| name                      | COOKBOOK_LISTENER_1                            |
| protocol                  | HTTP                                           |
| protocol_port             | 80                                             |
| sni_container_refs        |                                                |
| tenant_id                 | 4819f952f3d0411183956113f0727b30               |
+---------------------------+------------------------------------------------+
```

How it works...

Listeners are created with the following syntax:

```
neutron lbaas-listener-create [--name NAME] \
--loadbalancer LOADBALANCER \
--protocol {TCP,HTTP,HTTPS,TERMINATED_HTTPS} \
--protocol-port PORT \
[--default-pool DEFAULT_POOL]
```

Access to the listener may be restricted by applying a security group to the respective port of the load balancer address. Use the `openstack port set` command described earlier in this chapter to apply a security group to the listener's port.

Verifying connectivity

Once the workflow has been completed and the application on the pool members has been started, connectivity to the load balancer VIP can be verified using a web browser or the `curl` command from a client that can reach the VIP. In this example, a web server is running on the pool member configured in a previous section. We will be connecting from the `qlbaas` namespace associated with the load balancer, but you can also connect from the `qdhcp` namespace associated with the network from which the VIP address was sourced.

Getting ready

You will need the following details, at a minimum, to test connectivity

- Network ID or Load balancer ID
- VIP address and port

For our example, the following will be used:

- Load Balancer ID: `aa599cee-b49f-44f1-a0fd-51fa69ebf6db` (`COOKBOOK_LOADBALANCER_1`)
- VIP address and port: `172.16.200.14:80`

How to do it...

From within the Neutron agent container, connectivity to the VIP can be confirmed using `curl` within the `qdhcp` or `qlbaas` namespace:

```
# ip netns exec qlbaas-aa599cee-b49f-44f1-a0fd-51fa69ebf6db \
curl http://172.16.200.14:80
```

The output will resemble the following:

```
Hello Cookbook Readers!
```

You may need to install the `curl` utility for the command to work. In the Neutron agent container, this can be accomplished by running `apt install curl`. The configuration of the web server on the pool member is beyond the scope of this book.

To provide access to the load balancer virtual IP from outside networks, it may be necessary to map a floating IP to the virtual IP. Instructions provided earlier in this chapter can assist with that task.

5
Nova – OpenStack Compute

In this chapter, we will cover the following topics:

- Introduction to OpenStack Compute
- Adding a compute host using OpenStack-Ansible
- Suspending a host for maintenance
- Configuring Nova Scheduler to use host aggregates
- Creating a host aggregate
- Adding a compute host to a host aggregate
- Removing a compute host from a host aggregate
- Adding metadata to a host aggregate
- Deleting a host aggregate
- Creating an Availability Zone
- Booting an instance into an Availability Zone
- Removing an Availability Zone
- Creating a flavor
- Deleting a flavor
- Setting CPU limits for a flavor
- Setting IOPS limits for a flavor
- Booting an instance
- Stopping an instance

- Deleting an instance
- Live migration
- Snapshotting an instance
- Booting an instance from a snapshot
- Rescuing an instance
- Shelving an instance
- Reviewing the console logs

Introduction to OpenStack Compute

Compute services in OpenStack are provided by a project that goes by the name Nova. Nova is an API-driven system that manages physical and virtual compute resources in an OpenStack cloud, providing **Infrastructure as a Service** (**IaaS**). OpenStack operators, administrators, and users can leverage the Nova API to manage the life cycle of compute resources.

Nova is primarily responsible for managing two resource types: **instances**, which are the running virtual machines, application containers, or even full bare-metal machines a user has requested, and **hosts**, that provide the hardware resources required by instances. In most circumstances, instances are synonymous to virtual machines.

It is important to make a distinction between two main features of Nova: the Nova API service (and associated services such as the `nova-scheduler`, `nova-conductor`, and `nova-placement`), which runs on our cluster of three controller nodes, and the `nova-compute` service, which runs on each compute host in our environment.

In this chapter, we will first cover the administration of the physical host and Nova services, then the remainder of the chapter will take a task-centric approach to working with Nova. This means that the depth and detail on Nova features and functionality are beyond the scope of this book.

Adding a compute host using OpenStack-Ansible

In order to run instances, OpenStack Compute needs to be aware of the physical resources on which to run the instances. As OpenStack has grown and matured over time, the process for adding a host has also grown with it. OpenStack-Ansible provides a very convenient and consistent method for adding new compute hosts which allows you to scale your environment as your needs grow. A compute host runs the `nova-compute` service and the hypervisor that spawns the appropriate instance type requested.

Getting ready

In order to add a compute host to an OpenStack cluster using `openstack-ansible`, you will need the following information:

- The IP address of the host
- SSH access to the host
- The SSH key from your deployment host
- Access to the deployment host

The values used in our example are as follows:

- Compute host: `172.29.236.15`
- Deployment host: `controller-01.cook.book`

We assume that your environment was already installed using OpenStack-Ansible as described in *Chapter 1*, *Installing OpenStack with Ansible*.

How to do it...

To install and configure an additional compute host using `openstack-ansible`, carry out the following steps:

1. From the *deployment host*, add the additional compute host to the `/etc/openstack_deploy/openstack_user_config.yml` file, as shown in the following example:

```
compute_hosts:
  compute-01:
    ip: 172.29.236.13
  compute-02:
    ip: 172.29.236.14
  compute-03:
    ip: 172.29 236.15
```

2. Run `setup-hosts.yml`, limiting the tasks to just `compute-03`:

```
cd /opt/openstack-ansible/playbooks
openstack-ansible setup-hosts.yml --limit compute-03
```

3. Have Ansible gather facts about the new host:

```
ansible nova_all -m setup -a 'filter=ansible_local gather_
subset="!all"'
```

4. Simply run `os-nova-install.yml` to install and configure `nova-compute` on our new compute host:

```
openstack-ansible os-nova-install.yml --limit compute-03
```

Should the `openstack-ansible` command fail, the `-v` flag can be used to increase verbosity to assist in troubleshooting. Adding `-v` multiple times will provide additional levels of detail.

5. Confirm the new hypervisor by running the following commands on any of the controller servers (as they run the utility container):

```
lxc-attach --name $(lxc-ls -1 | grep utility)
source openrc
openstack hypervisor list
```

This should bring back output like the following:

```
+----+---------------------+-----------------+---------------+-------+
| ID | Hypervisor Hostname | Hypervisor Type | Host IP       | State |
+----+---------------------+-----------------+---------------+-------+
|  1 | compute-01.cook.book | QEMU            | 172.29.236.13 | up    |
|  2 | compute-02.cook.book | QEMU            | 172.29.236.14 | up    |
|  3 | compute-03.cook.book | QEMU            | 172.29.236.15 | up    |
+----+---------------------+-----------------+---------------+-------+
```

How it works...

The `/etc/openstack_deploy/openstack_user_config` file provides mappings of roles to hosts in our environment. Adding the IP address for `compute-03.cook.book` to the `compute_hosts` stanza, informs the `openstack-ansible` command to apply the packages, settings, and so forth to the new host. It also adds any required configuration to the remainder of your environment.

A full discussion of the Openstack-Ansible playbooks is beyond the scope of this book. For additional information on the playbooks and how they operate, you can review their documentation here: `https://docs.openstack.org/project-deploy-guide/openstack-ansible/pike/`

Suspending a host for maintenance

Often times a host needs to be taken offline to have memory replaced, an operating system upgraded, or other routine maintenance done. In order to ensure that running instances are not adversely affected, we use the `openstack compute service set` command.

Getting ready

In order to place a host into maintenance mode, you will need the following information:

- The `openstack` command-line utility
- The `nova` command-line utility
- An `openrc` file with admin credentials
- The *name* of the host

The host we will put into maintenance mode is as follows:

- Compute host: `compute-03`

How to do it...

To remove a hypervisor for maintenance, carry out the following steps:

1. First, we will list the available hosts:

   ```
   source ~/openrc

   openstack compute service list -c Binary -c Host -c Status -f
   table
   ```

   This will bring back an output like the following:

```
+------------------+-------------------------------------------------+---------+
| Binary           | Host                                            | Status  |
+------------------+-------------------------------------------------+---------+
| nova-conductor   | controller-01-nova-conductor-container-660b30cd | enabled |
| nova-scheduler   | controller-01-nova-scheduler-container-95b0fedf | enabled |
| nova-consoleauth | controller-01-nova-console-container-d428ff27   | enabled |
| nova-compute     | compute-01                                      | enabled |
| nova-compute     | compute-02                                      | enabled |
| nova-compute     | compute-03                                      | enabled |
+------------------+-------------------------------------------------+---------+
```

2. Next, we will disable the service called `nova-compute` for the compute host, so we will specify:

   ```
   openstack compute service set --disable compute-03 nova-compute
   ```

This command produces no output when successful.

3. Verify that the host is disabled with the following command:

```
openstack compute service list -c Binary -c Host -c Status -f
table
```

This produces an output like the following:

```
+------------------+----------------------------------------------------+----------+
| Binary           | Host                                               | Status   |
+------------------+----------------------------------------------------+----------+
| nova-conductor   | controller-01-nova-conductor-container-660b30cd | enabled  |
| nova-scheduler   | controller-01-nova-scheduler-container-95b0fedf | enabled  |
| nova-consoleauth | controller-01-nova-console-container-d428ff27   | enabled  |
| nova-compute     | compute-01                                         | enabled  |
| nova-compute     | compute-02                                         | enabled  |
| nova-compute     | compute-03                                         | disabled |
+------------------+----------------------------------------------------+----------+
```

4. Once the compute host has been disabled (meaning that it will no longer accept requests to run new instances), we can then migrate the running instances off this disabled compute host, onto other running compute hosts in our environment:

```
nova host-evacuate-live --block-migrate compute-03
```

This produces an output like the following:

```
+--------------------------------------+------------------------+---------------+
| Server UUID                          | Live Migration Accepted | Error Message |
+--------------------------------------+------------------------+---------------+
| ee2b46c0-015f-4928-9a49-367ccea599e2 | True                   |               |
+--------------------------------------+------------------------+---------------+
```

How it works...

In order to take a host offline for maintenance, Nova must first be told to no longer place new instances onto the host. This is done by disabling the `nova-compute` service on the compute host with the `openstack service set --disable [host] [service]` command. Once the host is marked disabled, and before powering it down for maintenance, the running instances need to be migrated off the host. The `nova host-evacuate-live [host]` command attempts to live-migrate all running instances on the specified host. It does this by asking Nova to reschedule the instances onto hosts with availability.

In cases where instances are not using shared storage, the `--block-migrate` flag can be used to attempt to migrate the storage as well.

Configuring Nova Scheduler to use host aggregates

OpenStack Compute provides the ability to create logical groupings of hypervisors called **host aggregates**, which allow users and administrators to control what physical compute hosts can run the requested workload. Often times, host aggregates are used to organize hosts with similar attributes, such as SSD, performance, or hosts, that have passed a security audit such as HITRUST or PCI. A host can be assigned to multiple aggregates. That is, a host can be part of the PCI and SSD grouping. To enable host aggregate scheduling in Nova Scheduler, you must first enable host aggregate filtering in `nova.conf`. In `openstack-ansible`, Ansible manages this file, and we will use it to push the appropriate changes.

Getting ready

Ensure that you are `root` on the deployment host. In most cases, this is the first infrastructure controller node, `infra01`.

How to do it...

To enable scheduling for host aggregates, the following steps can be used:

1. On the deployment host, add the following line to the `/etc/openstack_deploy/user_variables.yml` file:

   ```
   nova_scheduler_default_filters: "AggregateInstanceExtraSpecsFilter
   ,RetryFilter,AvailabilityZoneFilter,RamFilter,ComputeFilter,Comput
   eCapabilitiesFilter,ImagePropertiesFilter,ServerGroupAntiAffinityF
   ilter,ServerGroupAffinityFilter,AggregateCoreFilter,AggregateDiskF
   ilter"
   ```

2. We can then use Ansible to deploy the changes with the following commands:

   ```
   cd /opt/openstack-ansible/playbooks
   openstack-ansible os-nova-install.yml
   ```

3. Log in to the `nova-scheduler` container and verify the change:

   ```
   lxc-attach --name controller-01_nova_scheduler_container-ed0f657d
   grep Aggregate /etc/nova/nova.conf
   ```

The containers have unique UUIDs in their names. Look at the name of your containers with the following command:

```
lxc-ls -f
```

This will bring back an output like the following if the word `Aggregate` was found, which shows that the change was successful:

```
enabled_filters = AggregateInstanceExtraSpecsFilter,RetryFilter,Av
ailabilityZoneFilter,RamFilter,ComputeFilter,ComputeCapabilitiesFi
lter,ImagePropertiesFilter,ServerGroupAntiAffinityFilter,ServerGro
upAffinityFilter,AggregateCoreFilter,AggregateDiskFilter
```

How it works...

Adding `AggregateInstanceExtraSpecsFilter` to `enabled_filters`, tells `nova-scheduler` to match the instance *flavor metadata* to that of the *host-aggregate metadata*. The `openstack-ansible` command is then used to propagate this change to the `/etc/nova/nova.conf` files within the OpenStack cloud environment.

Once enabled, and when a new instance is requested, the Nova Scheduler examines the metadata of the flavor associated with the request and attempts to match it with the metadata of host aggregates.

Creating a host aggregate

After we have enabled OpenStack to allow host aggregate filtering, we can then proceed to create our host aggregates. This recipe will show you how to create a host aggregate.

Getting ready

To create a host aggregate, you will need the following information:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- A *name* for the aggregate

For our example, the host aggregate will be named `cookbook-ssd-hosts`.

How to do it...

To create a host aggregate, carry out the following steps:

1. First, we will list the current host aggregates:

```
openstack aggregate list
```

This gives an output like the following:

```
+----+-----------------------------+-------------------+
| ID | Name                        | Availability Zone |
+----+-----------------------------+-------------------+
| 19 | cookbook-threadripper-hosts | None              |
+----+-----------------------------+-------------------+
```

2. Next, we will create the aggregate:

```
openstack aggregate create cookbook-ssd-hosts
```

This will bring back an output like the following:

```
+-------------------+----------------------------+
| Field             | Value                      |
+-------------------+----------------------------+
| availability_zone | None                       |
| created_at        | 2017-09-08T19:43:41.141075 |
| deleted           | False                      |
| deleted_at        | None                       |
| id                | 20                         |
| name              | cookbook-ssd-hosts         |
| updated_at        | None                       |
+-------------------+----------------------------+
```

3. List the host aggregates:

```
openstack aggregate list
```

This gives an output like the following:

```
+----+-----------------------------+-------------------+
| ID | Name                        | Availability Zone |
+----+-----------------------------+-------------------+
| 19 | cookbook-threadripper-hosts | None              |
| 20 | cookbook-ssd-hosts          | None              |
+----+-----------------------------+-------------------+
```

How it works...

In Nova, a **host aggregate** provides OpenStack administrators and operators a mechanism to group hosts based on arbitrary attributes that are later used for controlling what compute hosts service a user's request. Creating a host aggregate is done with the `openstack aggregate create name` command.

Adding a compute host to a host aggregate

Before a host aggregate can be used by OpenStack Compute, you must first add hosts to the aggregate groups created.

Getting ready

To add a host to an aggregate, you will need the following information:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* or *ID* of the aggregate
- The *name* or *ID* of the host

For the following example, the required values are as follows:

- Host aggregate: `cookbook-ssd-hosts`
- Compute host: `compute-01`

How to do it...

The following process is used to add a host to a host aggregate:

1. List hosts that are already in the aggregate:

   `openstack aggregate show cookbook-ssd-hosts`

   This will bring back an output like the following:

```
+-------------------+----------------------------+
| Field             | Value                      |
+-------------------+----------------------------+
| availability_zone | None                       |
| created_at        | 2017-09-08T19:43:41.000000 |
| deleted           | False                      |
| deleted_at        | None                       |
| hosts             | [u'compute-02']            |
| id                | 20                         |
| name              | cookbook-ssd-hosts         |
| properties        |                            |
| updated_at        | None                       |
+-------------------+----------------------------+
```

2. Now add the specified compute host to this aggregate as follows:

   `openstack aggregate add host cookbook-ssd-hosts compute-01`

This will bring back an output like the following:

```
+-------------------+---------------------------------+
| Field             | Value                           |
+-------------------+---------------------------------+
| availability_zone | None                            |
| created_at        | 2017-09-08T19:43:41.000000      |
| deleted           | False                           |
| deleted_at        | None                            |
| hosts             | [u'compute-02', u'compute-01']  |
| id                | 20                              |
| metadata          | {}                              |
| name              | cookbook-ssd-hosts              |
| updated_at        | None                            |
+-------------------+---------------------------------+
```

How it works...

The `openstack aggregate host add` command associates a host with a host aggregate in the Nova database. This information is then used by the Nova Scheduler to make decisions about where to place an instance. If the metadata of the host aggregate matches that of the flavor or a particular project a user is in, an instance can be scheduled to a host within the aggregate.

Removing a compute host from a host aggregate

If the properties of a host change, or the needs of an aggregate change, hosts can be removed from a host aggregate.

Getting ready

To remove a host from a host aggregate, you will need the following information:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* or *ID* of the aggregate
- The *name* or *ID* of the host

How to do it...

1. List hosts already in the aggregate with the following command:

   ```
   openstack aggregate show cookbook-ssd-hosts
   ```

   This will bring back an output like the following. Note that the `hosts` field shows the compute hosts that are currently associated with the host aggregate:

   ```
   +-------------------+--------------------------------+
   | Field             | Value                          |
   +-------------------+--------------------------------+
   | availability_zone | None                           |
   | created_at        | 2017-09-08T19:43:41.000000     |
   | deleted           | False                          |
   | deleted_at        | None                           |
   | hosts             | [u'compute-02', u'compute-01'] |
   | id                | 20                             |
   | metadata          | {}                             |
   | name              | cookbook-ssd-hosts             |
   | updated_at        | None                           |
   +-------------------+--------------------------------+
   ```

2. To remove the compute host from this host aggregate, issue the following command:

   ```
   openstack aggregate remove host cookbook-ssd-hosts compute-01
   ```

   This will bring back the following. Note that in the `hosts` field, `compute-01` is now missing:

   ```
   +-------------------+----------------------------+
   | Field             | Value                      |
   +-------------------+----------------------------+
   | availability_zone | None                       |
   | created_at        | 2017-09-08T19:43:41.000000 |
   | deleted           | False                      |
   | deleted_at        | None                       |
   | hosts             | [u'compute-02']            |
   | id                | 20                         |
   | metadata          | {}                         |
   | name              | cookbook-ssd-hosts         |
   | updated_at        | None                       |
   +-------------------+----------------------------+
   ```

How it works...

Removing a host from a host aggregate tells OpenStack Compute that the additional metadata filters no longer apply and the host can be scheduled as normal.

Currently, running instances are unaffected by this operation. Operations on host aggregates affect new instance requests only.

Adding metadata to a host aggregate

The power of host aggregates comes from the ability to use them to logically group hosts based on their properties. This can be used, for example, to enable load balancing, enforce physical separation, or keep instances that are insecure from being scheduled into a secured environment. We will show this by matching the **metadata** of a host aggregate with that of an instance flavor.

Getting ready

To add metadata to a host aggregate, you will need the following information:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* or *ID* of the aggregate
- The *metadata* property to add

In our example, these values will be the following:

- Host aggregate: `cookbook-ssd-hosts`
- Metadata: `ssd=true`

How to do it...

To add metadata to host aggregate, use the following process:

1. Show the existing metadata associated with an aggregate:

   ```
   openstack aggregate show cookbook-ssd-hosts
   ```

   This will bring back an output like the following. Note that the `properties` field is blank if no metadata is associated yet:

```
+-------------------+----------------------------+
| Field             | Value                      |
+-------------------+----------------------------+
| availability_zone | None                       |
| created_at        | 2017-09-08T19:43:41.000000 |
| deleted           | False                      |
| deleted_at        | None                       |
| hosts             | [u'compute-02']            |
| id                | 20                         |
| name              | cookbook-ssd-hosts         |
| properties        |                            |
| updated_at        | None                       |
+-------------------+----------------------------+
```

2. Add the metadata with the following command:

```
openstack aggregate set --property ssd=true cookbook-ssd-hosts
```

This command displays no output when successful.

3. Now confirm the addition of the metadata with the following command again:

```
openstack aggregate show cookbook-ssd-hosts
```

You will see that the `properties` field has been updated with this information:

```
+-------------------+----------------------------+
| Field             | Value                      |
+-------------------+----------------------------+
| availability_zone | None                       |
| created_at        | 2017-09-08T19:43:41.000000 |
| deleted           | False                      |
| deleted_at        | None                       |
| hosts             | [u'compute-02']            |
| id                | 20                         |
| name              | cookbook-ssd-hosts         |
| properties        | ssd='true'                 |
| updated_at        | None                       |
+-------------------+----------------------------+
```

4. Now that we have set the metadata of our host aggregate, we can now associate this with a flavor so that when that flavor is specified by a user during the boot process, Nova will match the property to only the hosts within that host aggregate group. For example, we may have a flavor that is called `cookbook.ssd` that sets the expectation, which when a user select, the host will have SSDs available. This is the power of host aggregates. In the example here, `compute-02` has SSDs available as we have specified this in the host aggregate named `cookbook-ssd-hosts`. To take advantage of this, let's create a new flavor called `cookbook.ssd` with the `ssd=true` property:

```
openstack flavor create
    --vcpus 1
    --ram 512
    --disk 5
    --public
    cookbook.ssd
```

Creating flavors is described in more detail in the *Creating a flavor* recipe.

5. We now need to set the flavor **extra specs**, so that on choosing the flavor it is associated with the relevant host aggregate. For this, we use the `nova` command:

```
nova flavor-key cookbook.ssd set ssd=true
```

You may need to use the UUID of the flavor instead of the name. If so, list the flavors with the following command and not the UUID of the `cookbook.ssd` flavor:

```
openstack flavor list
```

6. Now when a user selects the `cookbook.ssd` flavor, unknown to them, the instance will be restricted to the hosts within the host aggregate group that has this key/value pair set.

How it works...

Metadata is specified as a key/value pairing, which is then associated with a host aggregate in the Nova database. These key/value pairs are arbitrary and can be defined to match a given environment. A host aggregate can have any number of key/value pairs stored; however, this may adversely affect instance scheduling because of potentially conflicting or confusing key/value pairs.

Deleting a host aggregate

When a host aggregate no longer applies in a given environment, it can be deleted. A host aggregate must have all hosts removed before it can be removed.

Getting ready

To delete a host aggregate, you will need the following information:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* or *ID* of the aggregate

For the following example, we will delete the `cookbook-threadripper-hosts` aggregate.

How to do it...

The following commands are used to delete a host aggregate:

1. First, list the existing aggregates:

   ```
   openstack aggregate list
   ```

   This will bring back the following output:

   ```
   +----+-----------------------------+-------------------+
   | ID | Name                        | Availability Zone |
   +----+-----------------------------+-------------------+
   | 20 | cookbook-ssd-hosts          | None              |
   | 21 | cookbook-threadripper-hosts | None              |
   +----+-----------------------------+-------------------+
   ```

2. Confirm that the aggregate has no hosts (Refer to the *Removing a compute host from a host aggregate* recipe if necessary):

   ```
   openstack aggregate show cookbook-threadripper-hosts
   ```

   This will bring back the following output:

   ```
   +-------------------+-----------------------------+
   | Field             | Value                       |
   +-------------------+-----------------------------+
   | availability_zone | None                        |
   | created_at        | 2017-09-08T20:37:33.000000  |
   | deleted           | False                       |
   | deleted_at        | None                        |
   | hosts             | []                          |
   | id                | 21                          |
   | name              | cookbook-threadripper-hosts |
   | properties        |                             |
   | updated_at        | None                        |
   +-------------------+-----------------------------+
   ```

3. Now we can delete the aggregate with the following command:

   ```
   openstack aggregate delete cookbook-threadripper-hosts
   ```

This command produces no output when successful.

4. Confirm the change with the following command:

```
openstack aggregate list
```

This will bring back the following output, where the host aggregate has now been removed:

```
+----+--------------------+-------------------+
| ID | Name               | Availability Zone |
+----+--------------------+-------------------+
| 20 | cookbook-ssd-hosts | None              |
+----+--------------------+-------------------+
```

How it works...

Deleting a host aggregate removes it and all of its metadata from the Nova database, and it can no longer be used for instance scheduling.

Creating an Availability Zone

Availability Zones (**AZs**) are a special case of host aggregates. Here, host aggregates are used by Nova to make scheduling decisions; they are generally only visible to operators and administrators of the OpenStack Cloud. AZs are the end-user visible component. An AZ can be configured, like host aggregates, to represent features of the hardware.

AZs are typically used to define failure domains, such as a cabinet or data center, or even geography. When configuring AZs, it is important to note that a host can only be a member of one AZ at a time.

As AZs are a special use-case of host aggregates, a lot of operations, such as adding and removing hosts are the same. Thus, they are not repeated here.

Creating an AZ is a two-step process. First, we create an aggregate with the `--zone` parameter, or add it to an existing aggregate. Second, a host needs to be added to the aggregate before instances can be scheduled into the zone.

Getting ready

To create an AZ, you will need the following information:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* or *ID* of the aggregate

- The desired *name* of the AZ
- The *host* or *hosts* to add to the AZ

For our example, these values are as follows:

- Aggregate name: `cookbook-az`
- Availability zone name: `cookbook-az`
- Compute host: `compute-01`

How to do it...

The following steps are to be used when creating an AZ:

1. First, we will list the current AZs with the following command:

   `openstack availability zone list`

   This will bring back an output like the following:

```
+-------------+-------------+
| Zone Name   | Zone Status |
+-------------+-------------+
| internal    | available   |
| nova        | available   |
| nova        | available   |
| nova        | available   |
+-------------+-------------+
```

2. Create a new aggregate with the `--zone` parameter:

   `openstack aggregate create --zone cookbook-az cookbook-az`

   This will bring back an output like the following:

```
+-------------------+----------------------------+
| Field             | Value                      |
+-------------------+----------------------------+
| availability_zone | cookbook-az                |
| created_at        | 2017-09-19T00:54:48.560890 |
| deleted           | False                      |
| deleted_at        | None                       |
| id                | 24                         |
| name              | cookbook-az                |
| updated_at        | None                       |
+-------------------+----------------------------+
```

3. Now add a host to the aggregate with the following command:

   `openstack aggregate add host cookbook-az compute-01`

This will bring back an output like the following when successful:

```
+-------------------+----------------------------------------+
| Field             | Value                                  |
+-------------------+----------------------------------------+
| availability_zone | cookbook-az                            |
| created_at        | 2017-09-19T00:54:48.000000             |
| deleted           | False                                  |
| deleted_at        | None                                   |
| hosts             | [u'compute-01']                        |
| id                | 24                                     |
| metadata          | {u'availability_zone': u'cookbook-az'} |
| name              | cookbook-az                            |
| updated_at        | None                                   |
+-------------------+----------------------------------------+
```

4. Now list the AZs we have available:

   `openstack availability zone list`

   This will show the following:

```
+-------------+-------------+
| Zone Name   | Zone Status |
+-------------+-------------+
| internal    | available   |
| cookbook-az | available   |
| nova        | available   |
| nova        | available   |
| nova        | available   |
+-------------+-------------+
```

Listing AZs will show all AZs available, irrespective of whether hosts exist in them or not. To verify that the hosts that are within an AZ, issue the following command:

`openstack availability zone show nameofAZ`

This will bring back the output as shown in step 3.

How it works...

The first command, `openstack availability zone list`, lists all AZs. Next, when creating the new aggregate, the special `--zone cookbook-az` parameter is passed, telling OpenStack Nova that this aggregate is also an AZ, specifying its name. Finally, a host is added to the AZ much the same as a host is added to an aggregate.

Booting an instance into an Availability Zone

Instances can be created into a given AZ by passing the `--availability-zone` parameter as part of the creation process. Additionally, AZ can be selected in OpenStack Horizon as part of the instance creation wizard.

Getting ready

The following information is required to boot an instance into an AZ:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* or *ID* of the AZ
- Additional information required to start an instance

Creating instances is covered in detail in the *Booting an instance* recipe. As such, this example skips some of the detail, instead it focuses on how to create an instance in a specific AZ.

How to do it...

To boot an instance into a specific AZ, the `--availability-zone` parameter is specified when running the `openstack server create` command:

```
openstack server create --flavor openstack.cookbook
   --image 08b822ba-be44-4639-b577-45e8dd37c06d
   --nic net-id=6cb5a4ce-1ea5-4817-9c34-a2212a66f394
   --security-group bd7d5e0f-538a-4d93-b123-98cc3207b3d2
   --key-name cookbook_key
   --availability-zone cookbook-az
   cookbook.test-03
```

This will bring back an output like the following, showing the AZ (the `OS-EXT-AZ:availability-zone` field) where the instance was scheduled to:

```
+-------------------------------------+-----------------------------------------------------------------+
| Field                               | Value                                                           |
+-------------------------------------+-----------------------------------------------------------------+
| OS-DCF:diskConfig                   | MANUAL                                                          |
| OS-EXT-AZ:availability_zone         | cookbook-az                                                     |
| OS-EXT-SRV-ATTR:host                | None                                                            |
| OS-EXT-SRV-ATTR:hypervisor_hostname | None                                                            |
| OS-EXT-SRV-ATTR:instance_name       |                                                                 |
| 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                           | doUrpyJUJ5GL                                                    |
| config_drive                        |                                                                 |
| created                             | 2017-09-19T01:15:48Z                                            |
| flavor                              | openstack.cookbook (0e039b72-50b2-4827-9403-e1f855bc2faf)       |
| hostId                              |                                                                 |
| id                                  | 0897c2f6-86e8-4860-86be-54ce6f750658                            |
| image                               | Ubuntu 16.04 amd64 (08b822ba-be44-4639-b577-45e8dd37c06d)       |
| key_name                            | cookbook_key                                                    |
| name                                | cookbook.test-03                                                |
| progress                            | 0                                                               |
| project_id                          | b54c75536541478f9c9ce48b0d8992c5                                |
| properties                          |                                                                 |
| security_groups                     | name='bd7d5e0f-538a-4d93-b123-98cc3207b3d2'                     |
| status                              | BUILD                                                           |
| updated                             | 2017-09-19T01:15:48Z                                            |
| user_id                             | 832892c82c2a48fca3571946e4cdcce9                                |
| volumes_attached                    |                                                                 |
+-------------------------------------+-----------------------------------------------------------------+
```

How it works...

When specified, the `--availability-zone` parameter causes the Nova Scheduler to examine the specified AZ to see if the request can be satisfied.

Removing an Availability Zone

Removing an AZ is a multistep process, and has some caveats. To remove an AZ, you must first remove the hosts from the host aggregate.

Getting ready

The following information is required to remove an AZ:

- The `openstack` command line client
- The `openrc` file containing appropriate credentials
- The *name* or *ID* of the AZ

How to do it...

The following steps can be used to remove an AZ:

1. First, list the AZs available with the following command:

   ```
   openstack availability zone list
   ```

   This will bring back an output like the following:

   ```
   +-------------+-------------+
   | Zone Name   | Zone Status |
   +-------------+-------------+
   | internal    | available   |
   | cookbook-az | available   |
   | nova        | available   |
   | nova        | available   |
   | nova        | available   |
   +-------------+-------------+
   ```

2. Now list the hosts within the AZ that we want to delete:

   ```
   openstack aggregate show cookbook-az
   ```

   This will bring back an output like the following:

   ```
   +-------------------+----------------------------------------+
   | Field             | Value                                  |
   +-------------------+----------------------------------------+
   | availability_zone | cookbook-az                            |
   | created_at        | 2017-09-19T00:54:48.000000             |
   | deleted           | False                                  |
   | deleted_at        | None                                   |
   | hosts             | [u'compute-01']                        |
   | id                | 24                                     |
   | metadata          | {u'availability_zone': u'cookbook-az'} |
   | name              | cookbook-az                            |
   | updated_at        | None                                   |
   +-------------------+----------------------------------------+
   ```

3. Now remove the hosts with following command (repeat as necessary for each host in the aggregate):

   ```
   openstack aggregate remove host cookbook-az compute-01
   ```

This will bring back the following output:

```
+-------------------+----------------------------------------+
| Field             | Value                                  |
+-------------------+----------------------------------------+
| availability_zone | cookbook-az                            |
| created_at        | 2017-09-19T00:54:48.000000             |
| deleted           | False                                  |
| deleted_at        | None                                   |
| hosts             | []                                     |
| id                | 24                                     |
| metadata          | {u'availability_zone': u'cookbook-az'} |
| name              | cookbook-az                            |
| updated_at        | None                                   |
+-------------------+----------------------------------------+
```

4. Remove the AZ using the `aggregate delete` command:

   `openstack aggregate delete cookbook-az`

This command produces no output if successful.

5. Confirm the removal of the AZ with the following command:

   `openstack availability zone list`

   This will bring back an output like the following, showing that our AZ has been removed:

```
+-----------+-------------+
| Zone Name | Zone Status |
+-----------+-------------+
| internal  | available   |
| nova      | available   |
| nova      | available   |
| nova      | available   |
+-----------+-------------+
```

How it works...

In order to remove an AZ, it must first have no active hosts. Because AZs are a special case of host aggregates, the commands for identifying and removing hosts from an AZ are the same. Once you have removed the hosts, the `openstack aggregate delete [name]` command is used to complete the removal of the AZ.

Creating a flavor

Before Nova can start instances, it first needs to know what resources should be assigned to those instances. The way Nova handles resource assignments is to define **flavors**. A flavor specifies the number of vCPUs, RAM, and the disk to assign to an instance.

Getting ready

To create a new flavor, you will need the following:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name*, *vCPU*, *RAM*, and *disk* values for the new flavor

The flavor we will create in this example will have the following attributes:

- Flavor name: `openstack.cookbook`
- vCPU: 1
- Ram: 512 MB
- Disk: 5 GB
- Visibility: Public

How to do it...

The following commands are used to create a new flavor:

1. First, list the available flavors already configured in our environment with the following command:

   ```
   openstack flavor list
   ```

   This will bring back an output like the following:

```
+-----+----------+-----+------+-----------+-------+-----------+
| ID  | Name     | RAM | Disk | Ephemeral | VCPUs | Is Public |
+-----+----------+-----+------+-----------+-------+-----------+
| 201 | tempest1 | 256 |    1 |         0 |     1 | True      |
| 202 | tempest2 | 512 |    1 |         0 |     1 | True      |
+-----+----------+-----+------+-----------+-------+-----------+
```

2. Create the flavor with our given attributes:

```
openstack flavor create
    --vcpus 1
    --ram 512
    --disk 5
    --public
    openstack.cookbook
```

This will bring back an output like the following:

```
+----------------------------+--------------------------------------+
| Field                      | Value                                |
+----------------------------+--------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                |
| OS-FLV-EXT-DATA:ephemeral  | 0                                    |
| disk                       | 5                                    |
| id                         | e15fd116-cf35-4a20-b035-84b1d297ec82 |
| name                       | openstack.cookbook                   |
| os-flavor-access:is_public | True                                 |
| properties                 |                                      |
| ram                        | 512                                  |
| rxtx_factor                | 1.0                                  |
| swap                       |                                      |
| vcpus                      | 1                                    |
+----------------------------+--------------------------------------+
```

3. List the flavors again to see the new flavor:

```
openstack flavor list
```

This gives an output like the following:

```
+--------------------+--------------------+------+------+-----------+-------+-----------+
| ID                 | Name               |  RAM | Disk | Ephemeral | VCPUs | Is Public |
+--------------------+--------------------+------+------+-----------+-------+-----------+
| 201                | tempest1           |  256 |    1 |         0 |     1 | True      |
| 202                | tempest2           |  512 |    1 |         0 |     1 | True      |
| e15fd116-cf35-4a20 | openstack.cookbook |  512 |    5 |         0 |     1 | True      |
| -b035-84b1d297ec82 |                    |      |      |           |       |           |
+--------------------+--------------------+------+------+-----------+-------+-----------+
```

How it works...

The `openstack flavor create --vcpus [vcpu_count] --ram [ram_MB] --disk [disk_GB] --public [name]` command is used to define flavors. The `--public` option is used to specify if the flavor should be available to all users of the OpenStack environment, or if it should remain limited in scope and visibility to the project in which the user belongs to.

Deleting a flavor

Often times, requirements change; there is a demand for more performance, or business needs change. Whatever be the reason, your existing flavors may not meet the needs of those consuming cloud resources. When a given flavor is no longer suitable, it will need to be deleted.

A flavor that is associated with a running instance cannot be deleted.

Getting ready

To change attributes of a flavor for Nova, you will need the following:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* of the flavor to delete

How to do it...

The following commands are used to delete a flavor:

1. First, list the flavors available:

   ```
   openstack flavor list
   ```

   This will bring back a list of flavors like the following:

```
+--------------------+--------------------+------+------+-----------+-------+-----------+
| ID                 | Name               |  RAM | Disk | Ephemeral | VCPUs | Is Public |
+--------------------+--------------------+------+------+-----------+-------+-----------+
| 201                | tempest1           |  256 |    1 |         0 |     1 | True      |
| 202                | tempest2           |  512 |    1 |         0 |     1 | True      |
| e15fd116-cf35-4a20 | openstack.cookbook |  512 |    5 |         0 |     1 | True      |
| -b035-84b1d297ec82 |                    |      |      |           |       |           |
+--------------------+--------------------+------+------+-----------+-------+-----------+
```

2. To delete the flavor, execute the following command:

   ```
   openstack flavor delete openstack.cookbook
   ```

This command shows no output when successful.

3. List the flavors again to view updated attributes:

```
openstack flavor list
```

Once again, this lists the flavors that are now available:

```
+-----+----------+-----+------+-----------+-------+-----------+
| ID  | Name     | RAM | Disk | Ephemeral | VCPUs | Is Public |
+-----+----------+-----+------+-----------+-------+-----------+
| 201 | tempest1 | 256 |    1 |         0 |     1 | True      |
| 202 | tempest2 | 512 |    1 |         0 |     1 | True      |
+-----+----------+-----+------+-----------+-------+-----------+
```

How it works...

To delete a flavor the `openstack flavor delete [name]` command is used. Should you need to delete multiple flavors, you can specify multiple names or IDs, separating each with a space.

Setting CPU limits for a flavor

In addition to defining the number of vCPUs assigned to an instance, limits on the use of these CPU cycles can be further imposed. Nova relies on the underlying hypervisor for the specific implementation of the CPU limits, and thus the values available may vary. Our example is based on QEMU/KVM.

CPU limits are a special case of flavor attributes that you may encounter.

Getting ready

To add a CPU limit to a flavor, you will need the following:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* of the flavor to change
- The values you would like to set the CPU limit to (share of time the allotted CPU is allowed to consume in milliseconds per cycle)

These values in our example are as follows:

- `cpu_quota` = 5000 ms
- `cpu_period` = 2500 ms

How to do it...

The following commands are used to add CPU limits to a flavor.

CPU limits are not applied live. Rather, they are applied on instance launch.

At the time of this writing, the `openstack` command-line client is the only way to view and change this setting. It is also not shown by default when viewing instance attributes.

1. First, view the current attributes of the flavor that we are changing, noting that we have no `properties` set:

   ```
   openstack flavor show openstack.cookbook
   ```

   This gives the following output:

   ```
   +----------------------------+--------------------------------------+
   | Field                      | Value                                |
   +----------------------------+--------------------------------------+
   | OS-FLV-DISABLED:disabled   | False                                |
   | OS-FLV-EXT-DATA:ephemeral  | 0                                    |
   | access_project_ids         | None                                 |
   | disk                       | 20                                   |
   | id                         | 5c3c4400-412d-431e-900d-75f50c6bf73f |
   | name                       | openstack.cookbook                   |
   | os-flavor-access:is_public | True                                 |
   | properties                 |                                      |
   | ram                        | 1024                                 |
   | rxtx_factor                | 1.0                                  |
   | swap                       |                                      |
   | vcpus                      | 2                                    |
   +----------------------------+--------------------------------------+
   ```

2. Add the CPU limit:

   ```
   openstack flavor set
       --property quota:cpu_quota=5000
       --property quota:cpu_period=2500
       openstack.cookbook
   ```

This command shows no output when successful

3. View the CPU limit by interrogating the flavor again, noting the `properties` field:

   `openstack flavor show openstack.cookbook`

   This gives an output like the following:

```
+----------------------------+-----------------------------------------------+
| Field                      | Value                                         |
+----------------------------+-----------------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                         |
| OS-FLV-EXT-DATA:ephemeral  | 0                                             |
| access_project_ids         | None                                          |
| disk                       | 5                                             |
| id                         | 5c3c4400-412d-431e-900d-75f50c6bf73f          |
| name                       | openstack.cookbook                            |
| os-flavor-access:is_public | True                                          |
| properties                 | quota:cpu_period='2500', quota:cpu_quota='5000' |
| ram                        | 512                                           |
| rxtx_factor                | 1.0                                           |
| swap                       |                                               |
| vcpus                      | 1                                             |
+----------------------------+-----------------------------------------------+
```

How it works...

In order to help mitigate the *noisy neighbor* issue, or to provide further definition of service levels, OpenStack supports CPU limits. In OpenStack, this is called the **Instance Resource Quota**. CPU limits are part of the flavor definition. Meaning, all instances of a given flavor will have the same CPU limits.

Imposing CPU limits are hypervisor specific. For KVM/libvirt environments, CPU limits are enforced using cgroups, using a combination of the following three values:

- `cpu_shares`
- `cpu_quota`
- `cpu_period`

A full discussion of what these values are and how they interact is beyond the scope of this book. However, you can find a detailed explanation of them on the OpenStack wiki at the following:

`https://wiki.openstack.org/wiki/InstanceResourceQuota`

Setting IOPS limits for a flavor

As with CPU limits, IOPS limits can also be imposed to prevent an instance type from hogging all of the available IO on a system. IOPS here refers to the storage and network IO. Also, as with CPU limits, IOPS limits are a special case of flavor attributes that you may encounter frequently.

At the time of this writing, the `openstack` command-line client is the only way to view and change this setting. It is also not shown by default when viewing instance attributes.

Getting ready

To add an IOPS limit to a flavor, you will need the following:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* of the flavor to change
- The value you would like to set the IOPS limit to.
- In our example, these values are as follows:
- `disk_read_iops` = 100 IOPS
- `disk_write_iops` = 100 IOPS

How to do it...

The following commands are used to add IOPS limits to a flavor:

As with changing flavor attributes, IOPS limits are not applied to running instances. Rather, they are applied on instance launch.

1. First, view the attributes of the flavor that is changing, noting that the `properties` field is blank:

```
openstack flavor show openstack.cookbook
```

This gives an output like the following:

```
+----------------------------+--------------------------------------+
| Field                      | Value                                |
+----------------------------+--------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                |
| OS-FLV-EXT-DATA:ephemeral  | 0                                    |
| access_project_ids         | None                                 |
| disk                       | 5                                    |
| id                         | 5c3c4400-412d-431e-900d-75f50c6bf73f |
| name                       | openstack.cookbook                   |
| os-flavor-access:is_public | True                                 |
| properties                 |                                      |
| ram                        | 512                                  |
| rxtx_factor                | 1.0                                  |
| swap                       |                                      |
| vcpus                      | 1                                    |
+----------------------------+--------------------------------------+
```

2. Now we can add the IOPS limit with the following command:

```
openstack flavor set
    --property quota:disk_read_iops=100
    --property quota:disk_write_iops=100
    openstack.cookbook
```

This command shows no output when successful.

3. We can verify the IOPS limits by viewing the flavor properties:

```
openstack flavor show openstack.cookbook
```

This will bring back an output like the following, noting that we have now set the properties to reflect the IOPS imposed:

```
+----------------------------+-----------------------------------------------------------+
| Field                      | Value                                                     |
+----------------------------+-----------------------------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                                     |
| OS-FLV-EXT-DATA:ephemeral  | 0                                                         |
| access_project_ids         | None                                                      |
| disk                       | 5                                                         |
| id                         | 5c3c4400-412d-431e-900d-75f50c6bf73f                      |
| name                       | openstack.cookbook                                        |
| os-flavor-access:is_public | True                                                      |
| properties                 | quota:disk_read_iops='100', quota:disk_write_iops='100'   |
| ram                        | 512                                                       |
| rxtx_factor                | 1.0                                                       |
| swap                       |                                                           |
| vcpus                      | 1                                                         |
+----------------------------+-----------------------------------------------------------+
```

How it works...

As with CPU limits, IO limits are defined at a flavor level. However, unlike CPU limits, IO limits can be applied by the hypervisor, the storage layer, or a combination of the two.

To set the IOPS limit values for a given flavor, the following `openstack` command is used:

```
openstack flavor set
    --property quota:disk_read_iops=[read_iops]
    --property quota:disk_write_iops=[write_iops]
    [name]
```

While both read and write IOPS are shown for completeness, you need not specify both.

Booting an instance

The most fundamental tasks that can be performed with instances are life cycle tasks. In this recipe, we will show you how to start or boot an instance.

Getting ready

To start an instance, you will need the following:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* of the instance
- The *image* to use for the instance
- The name of the *flavor* to create the instance with
- The name of the *network* or networks to attach the instance to
- The *keypair* name to allow access to an instance
- The name of any *security groups* to associate with the instance

How to do it...

The instance we will boot will have the following attributes:

- Instance name: `cookbook.test`
- Flavor type: `openstack.cookbook`
- Image name: `Ubuntu 16.04 amd64` (UUID)

- Network: `public` (UUID)
- Keypair name: `cookbook_key`
- Security groups: `ssh`

To start an instance, use the following steps:

1. To list the instances that are currently running, we issue the following command:

   ```
   openstack server list
   ```

   This command will produce no output if there are no instances.

2. List the available images:

   ```
   openstack image list
   ```

   This will bring back a list of images available:

```
+--------------------------------------+--------------------+--------+
| ID                                   | Name               | Status |
+--------------------------------------+--------------------+--------+
| 08b822ba-be44-4639-b577-45e8dd37c06d | Ubuntu 16.04 amd64 | active |
| d47b0acc-4755-44c6-8128-1a70b169437d | cirros             | active |
| 6aa08a3d-c4b4-4f6f-97af-9b37ae02b87b | cirros_alt         | active |
+--------------------------------------+--------------------+--------+
```

3. List the available networks:

   ```
   openstack network list
   ```

   This will list the networks that we can attach an instance to:

```
+--------------------------------------+---------+--------------------------------------+
| ID                                   | Name    | Subnets                              |
+--------------------------------------+---------+--------------------------------------+
| 3f263ef3-6b81-4e4e-a125-4804dff53aeb | private | 6e0cbdcf-2e75-4bf2-9e99-8e3ba4b42ad6 |
| 6cb5a4ce-1ea5-4817-9c34-a2212a66f394 | public  | db3b0e8d-bc2b-4569-8d42-7c833ff3a330 |
+--------------------------------------+---------+--------------------------------------+
```

4. Create a keypair (if you do not have one already created and available for use):

   ```
   ssh-keygen -q -N ""
   ```

   Also, upload this with the following command:

   ```
   openstack keypair create --public-key ~/.ssh/id_rsa.pub cookbook_
   key
   ```

This will bring back an output showing the fingerprint of your key:

```
+-------------+-------------------------------------------------+
| Field       | Value                                           |
+-------------+-------------------------------------------------+
| fingerprint | 79:7e:e3:a1:d3:6b:9c:a8:73:75:94:d7:2a:b5:61:2d |
| name        | cookbook_key                                    |
| user_id     | 832892c82c2a48fca3571946e4cdcce9                |
+-------------+-------------------------------------------------+
```

5. Now we can boot the instance with the following command:

```
openstack server create
    --flavor openstack.cookbook
    --image 08b822ba-be44-4639-b577-45e8dd37c06d
    --nic net-id=6cb5a4ce-1ea5-4817-9c34-a2212a66f394
    --security-group ssh
    --key-name cookbook_key
    cookbook.test
```

   This produces a table of output showing that the instance is booting (not shown here for brevity).

6. To view more information about the booted instance, issue the following command:

```
openstack server show cookbook.test
```

   This gives an output like the following:

```
+--------------------------------------+------------------------------------------------------------------+
| Field                                | Value                                                            |
+--------------------------------------+------------------------------------------------------------------+
| OS-DCF:diskConfig                    | MANUAL                                                           |
| OS-EXT-AZ:availability_zone          | nova                                                             |
| OS-EXT-SRV-ATTR:host                 | compute-01                                                       |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | compute-01.cook.book                                             |
| OS-EXT-SRV-ATTR:instance_name        | instance-00000073                                                |
| OS-EXT-STS:power_state               | Running                                                          |
| OS-EXT-STS:task_state                | None                                                             |
| OS-EXT-STS:vm_state                  | active                                                           |
| OS-SRV-USG:launched_at               | 2017-09-06T18:49:54.000000                                       |
| OS-SRV-USG:terminated_at             | None                                                             |
| accessIPv4                           |                                                                  |
| accessIPv6                           |                                                                  |
| addresses                            | public=10.1.13.3                                                 |
| config_drive                         |                                                                  |
| created                              | 2017-09-06T18:49:40Z                                             |
| flavor                               | openstack.cookbook (0e039b72-50b2-4827-9403-e1f855bc2faf)        |
| hostId                               | 8c16dc36a8f61c00194243889c92a3e0079a987cf15b2029b025201e         |
| id                                   | 62df5e12-e13d-4132-ac73-a17cf7d190d4                             |
| image                                | Ubuntu 16.04 amd64 (08b822ba-be44-4639-b577-45e8dd37c06d)        |
| key_name                             | cookbook_key                                                     |
| name                                 | cookbook.test                                                    |
| progress                             | 0                                                                |
| project_id                           | b54c75536541478f9c9ce48b0d8992c5                                 |
| properties                           |                                                                  |
| security_groups                      | name='default'                                                   |
| status                               | ACTIVE                                                           |
| updated                              | 2017-09-06T18:49:55Z                                             |
| user_id                              | 832892c82c2a48fca3571946e4cdcce9                                 |
| volumes_attached                     |                                                                  |
+--------------------------------------+------------------------------------------------------------------+
```

How it works...

After noting a number of required values we will be supplying as input, such as the UUID of the image that we want to boot and the UUID of the network we want to attach the instance to, we then called a tool from OpenStack Client to launch our instance. Part of that command line refers to the keypair to use. We then connect to the instance using the private key as part of that keypair generated.

How does the cloud instance know what key to use? As part of the boot scripts for this image, it makes a call back to the metaserver, which is a function of the `nova-api` and `nova-api-metadata` services. The metaserver provides a go-between that bridges our instance and the real world, which the cloud-init boot process can call and in this case, it downloads a script to inject our private key into the Ubuntu user's `.ssh/authorized_keys` file.

When a cloud instance is launched, it generates a number of useful metrics and details about that instance. This is presented by the `openstack server list` and `openstack server show` commands. The `openstack server list` command shows a convenient short version listing the ID, name, status, and IP addresses of our instance.

Stopping an instance

Sometimes, an instance will need to be stopped for a number of reasons, for example, maintenance and offline migration, yet as an OpenStack administrator, you may not want to completely destroy the instance and data associated with it. Thus, you will want to stop the instance instead.

Getting ready

To stop an instance, you will need the following:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* of the instance

How to do it...

In this example, we will be stopping the `cookbook.test` instance created in the *Booting an instance* recipe. To stop the instance, use the following steps:

1. First, we list the running instances:

```
openstack server list
```

This will bring back a list of the running instances. Consider this example:

```
+--------------------------------------+---------------+--------+-----------------+--------------------+
| ID                                   | Name          | Status | Networks        | Image Name         |
+--------------------------------------+---------------+--------+-----------------+--------------------+
| 62df5e12-e13d-4132-ac73-a17cf7d190d4 | cookbook.test | ACTIVE | public=10.1.13.3 | Ubuntu 16.04 amd64 |
+--------------------------------------+---------------+--------+-----------------+--------------------+
```

2. To stop the instance, simply issue the following command:

   `openstack server stop cookbook.test`

This command takes a few moments to complete and will not display the output if successful.

3. Now list instances to confirm the status of the instances. Here we're looking for the states of `SHUTOFF` for our `cookbook.test` instance:

   `openstack server list`

   This gives the following output:

```
+--------------------------------------+---------------+---------+-----------------+--------------------+
| ID                                   | Name          | Status  | Networks        | Image Name         |
+--------------------------------------+---------------+---------+-----------------+--------------------+
| 62df5e12-e13d-4132-ac73-a17cf7d190d4 | cookbook.test | SHUTOFF | public=10.1.13.3 | Ubuntu 16.04 amd64 |
+--------------------------------------+---------------+---------+-----------------+--------------------+
```

How it works...

To stop an instance we use the `openstack` client's `server stop` command: `openstack server stop [Name or ID]`. This tells Nova to power off the instance. This is synonymous to holding the power button on your laptop, or unplugging a server.

Deleting an instance

To complete the life cycle of an instance, you will need to delete it. Nova provides a facility for this, using the `openstack` command-line tool.

Getting ready

To delete an instance, you will need the following:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* of the instance

How to do it...

In this example, we will be deleting the `cookbook.test` instance created in the *Booting an instance* recipe used for booting. To delete the instance, use the following steps:

1. First, list the running instances:

   ```
   openstack server list
   ```

```
+--------------------------------------+------------------+--------+------------------+--------------------+
| ID                                   | Name             | Status | Networks         | Image Name         |
+--------------------------------------+------------------+--------+------------------+--------------------+
| 0001bbf6-a054-4d78-b17d-d1d389fa5777 | cookbook.test.02 | ACTIVE | public=10.1.13.5 | Ubuntu 16.04 amd64 |
| 51119fbb-fe94-4cbb-87ab-9f57cb880250 | cookbook.test    | ACTIVE | public=10.1.13.7 | Ubuntu 16.04 amd64 |
+--------------------------------------+------------------+--------+------------------+--------------------+
```

2. To delete the instance named `cookbook.test`, issue the following command:

   ```
   openstack server delete cookbook.test
   ```

   This command produces no output when successful.

3. Now list the instances again to confirm the deletion:

   ```
   openstack server list
   ```

   This will bring back an output, where the instance we have deleted is now not present:

```
+--------------------------------------+------------------+--------+------------------+--------------------+
| ID                                   | Name             | Status | Networks         | Image Name         |
+--------------------------------------+------------------+--------+------------------+--------------------+
| 0001bbf6-a054-4d78-b17d-d1d389fa5777 | cookbook.test.02 | ACTIVE | public=10.1.13.5 | Ubuntu 16.04 amd64 |
+--------------------------------------+------------------+--------+------------------+--------------------+
```

How it works...

The `openstack server delete` command, unlike suspending or shelving an instance, deletes the instance and all data within it.

Live migration

One of the key tenets of cloud is being abstracted away from hardware. However, hardware does require maintenance from time to time, or a host will need a software upgrade. Whatever the reason, a host may need to be taken offline for maintenance, ideally with little to no downtime for the instances running on the host. To accommodate this, Nova provides two methods for live migrating instances: block migration, when shared storage is not available in the environment, and the live migration flag, when instances are booted from the shared storage (where every compute host can see the same storage used by each instance for its boot volume).

Getting ready

To live migrate an instance, you will need the following information:

- The `openstack` command-line client
- The `openrc` file containing appropriate administrator credentials
- The *name* of the instance
- The *name* of the destination hypervisor

If your environment is configured so that instances boot from the shared storage, such as RBD provided by Ceph, then live migrations are almost instantaneous (depending on how busy the instance is). A feature called **block migration** is much slower. Block migration is used in environments where shared storage is not used for instances.

Migrations can only be performed by a user with admin role privileges.

How to do it...

1. First, we list the available hypervisors. We are noting where we can migrate our running instances to:

   ```
   openstack hypervisor list
   ```

   This will bring back an output like the following:

```
+----+---------------------+-----------------+---------------+-------+
| ID | Hypervisor Hostname  | Hypervisor Type | Host IP       | State |
+----+---------------------+-----------------+---------------+-------+
|  1 | compute-01.cook.book | QEMU            | 172.29.236.13 | up    |
|  2 | compute-02.cook.book | QEMU            | 172.29.236.14 | up    |
+----+---------------------+-----------------+---------------+-------+
```

2. Next, view the instance properties to identify the source host:

```
openstack server show cookbook.test
    -c name
    -c OS-EXT-SRV-ATTR:hypervisor_hostname
```

This will bring back the fields (also known as column names, as denoted by the `-c` flag) that we are interested in. Note, in this example, the instance called `cookbook.test` is running from the hypervisor called `compute-01.cook.book`:

```
+-------------------------------------+----------------------+
| Field                               | Value                |
+-------------------------------------+----------------------+
| OS-EXT-SRV-ATTR:hypervisor_hostname | compute-01.cook.book |
| name                                | cookbook.test        |
+-------------------------------------+----------------------+
```

3. To live migrate the instance *where no shared storage is available* issue the following:

```
openstack server migrate --block-migration cookbook.test
```

We do not specify a target compute host using the `--block-migration` method, instead, we let the Nova Scheduler decide on the next available hypervisor.

This command shows no output when successful.

To live migrate the instance, *with shared storage such as Ceph*, issue the following:

```
openstack server migrate cookbook.test --live compute-02.cook.book
```

We specify the target hypervisor with the `--live` flag.

This command shows no output when successful.

4. Check the status of migration:

```
openstack server show
    -c name
    -c OS-EXT-STS:task_state cookbook.test
```

As block migrations generally take longer, you may see the state as `resize_migrating`, indicating that the task of migrating is still in process:

```
+-----------------------+------------------+
| Field                 | Value            |
+-----------------------+------------------+
| OS-EXT-STS:task_state | resize_migrating |
| name                  | cookbook.test    |
+-----------------------+------------------+
```

5. Verify the migration with the following command, ensuring that the instance is now running from a different hypervisor:

```
openstack server show cookbook.test
    -c name
    -c OS-EXT-SRV-ATTR:hypervisor_hostname
```

This will bring back an output like the following:

```
+-----------------------------------+----------------------+
| Field                             | Value                |
+-----------------------------------+----------------------+
| OS-EXT-SRV-ATTR:hypervisor_hostname | compute-02.cook.book |
| name                              | cookbook.test        |
+-----------------------------------+----------------------+
```

Migrations of any kind are generally best performed when the instance is not under heavy utilization.

How it works...

Live migrations are an essential feature that enables OpenStack operators and administrators to perform maintenance of the underlying cloud infrastructure without affecting the consumers of said cloud. Additionally, the OpenStack administrator can use telemetry data from resource monitoring services (such as Ceilometer) and make live migration decisions to balance workloads across the OpenStack Cloud.

Live migration in OpenStack is handled by the libvirt drivers. Specifically, when you issue the `openstack server migrate` command, OpenStack Compute creates a connection from libvirtd on one compute host to the same process on the remote host. Once this connection is established, depending on the parameters you specified, the memory state of the instance is synchronized and the control is transferred. In the preceding example, we first specified the additional `--block-migrate` parameter, which handles the movement of the instance's disk files in the absence of shared storage, and then we showed how the `--live` flag can be used when instances are booted from shared storage.

Remember: Migrations can only be performed by users with the `admin` role.

Snapshotting an instance

Snapshotting an instance will create a Glance image of the instance at the point in time the snapshot was taken. This image can then be used to deploy additional instances of a given application, or as a bootable backup of the instance.

Getting ready

In order to create a snapshot of an instance, you require the following information:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* of the instance

How to do it...

The following commands are used to create an instance snapshot:

1. First, we list the existing images with the following command:

   ```
   openstack image list -c Name -c Status
   ```

   This will bring back a list of images like the following:

   ```
   +--------------------+--------+
   | Name               | Status |
   +--------------------+--------+
   | Ubuntu 16.04 amd64 | active |
   | cirros             | active |
   | cirros_alt         | active |
   +--------------------+--------+
   ```

2. Now list the running instances with the following command:

   ```
   openstack server list -c Name -c Status
   ```

This gives an output like the following:

Name	Status
cookbook.test.03	ACTIVE
cookbook.test.02	ACTIVE
cookbook.test	ACTIVE

3. To create the snapshot, issue the following command (note the optional shell expansion command we're using to timestamp the name of the snapshot):

```
openstack server image create
    --name cookbook.test_snapshot-$(date +"%FT%H%M%S")
    cookbook.test
```

This brings an output like the following:

Field	Value
checksum	None
container_format	None
created_at	2017-09-06T21:42:13Z
disk_format	None
file	/v2/images/f2c92002-7441-4ef2-bb09-e5ca606b9de7/file
id	f2c92002-7441-4ef2-bb09-e5ca606b9de7
min_disk	5
min_ram	0
name	cookbook.test_snapshot-2017-09-06T214209
owner	b54c75536541478f9c9ce48b0d8992c5
properties	base_image_ref='08b822ba-be44-4639-b577-45e8dd37c06d', boot_roles='heat_stack_owner,admin', image_type='snapshot', instance_uuid='94f5c32f-2412-4f5c-ba92-7874790b832a', user_id='832892c82c2a48fca3571946e4cdcce9'
protected	False
schema	/v2/schemas/image
size	None
status	queued
tags	
updated_at	2017-09-06T21:42:13Z
virtual_size	None
visibility	private

4. We can verify that the snapshot was created with the following command. Note that we limited the screenshot to just show our snapshotted image:

```
openstack image list
```

This gives an output that will show our snapshotted image:

ID	Name	Status
f2c92002-7441-4ef2-bb09-e5ca606b9de7	cookbook.test_snapshot-2017-09-06T214209	active

How it works...

Instance snapshots create a Glance image of the running instance. The snapshot can be used for backup, redistribution, or part of a continuous deployment pipeline as a build artifact. The images created with `openstack server image create --name [snapshot_name] [instance]` are bootable. You have a large degree of flexibility in how they are used.

There's more...

Instance snapshots are rather powerful. While a full exploration of the possibilities are beyond the scope of this book, the following example shows you what can be achieved using this feature: an easy way to back up all running instances.

To snapshot every instance, use the following command:

```
for instance in $(openstack server list -f value -c ID); do {
    openstack server image create \
      --name "${instance}"-$(date +"%FT%H%M%S") ${instance}
}; done
```

Warning: Snapshotting every instance is not recommended for larger environments. In addition to being time consuming, it can also consume a rather large amount of storage used by Glance, as snapshots are not sparsely created like those of the original QCOW2 image you may have used

Booting an instance from a snapshot

Now that we have created an instance snapshot, imagine that we need to go back and recover files, or revert the application to the state it was at the time the snapshot was taken. As the snapshot is stored as an OpenStack image, you can boot directly from the snapshot, and as such all the details required to boot an instance applies to booting a snapshot.

Getting ready

To start an instance from a snapshot, you will need the following:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* of the instance
- The *snapshot* to use for the instance
- The name of the *flavor* to create the instance with (this must be of equal or greater size of flavor compared with the one used in the original instance)
- The name of the *network* or networks to attach the instance to
- The *keypair* name to allow access to an instance
- The name of any *security groups* to associate with the instance

How to do it...

The instance we will boot will have the following attributes:

- Image name: `cookbook.test_snapshot-2017-09-08T163619`
- Instance name: `cookbook.test_restore`
- Flavor type: `openstack.cookbook`
- Network: `public` (UUID)
- Keypair name: `cookbook_key`
- Security groups: `ssh`

To boot from an instance snapshot, we use the same process as used when booting an instance. The commands are repeated here:

1. First, we can list the existing running instances with the following command:

   ```
   openstack server list -c Name -c Status
   ```

   This will bring back a list of `ACTIVE` running instances:

   ```
   +------------------+--------+
   | Name             | Status |
   +------------------+--------+
   | cookbook.test.03 | ACTIVE |
   | cookbook.test.02 | ACTIVE |
   | cookbook.test    | ACTIVE |
   +------------------+--------+
   ```

2. We can then boot the instance snapshot using the following command:

   ```
   openstack server create
       --flavor openstack.cookbook
       --image cookbook.test_snapshot-2017-09-08T163619
       --nic net-id=6cb5a4ce-1ea5-4817-9c34-a2212a66f394
       --security-group bd7d5e0f-538a-4d93-b123-98cc3207b3d2
       --key-name cookbook_key
     cookbook.test_restore
   ```

   This will bring back the familiar booting an instance output that has been omitted here.

3. We can now list the running instances again to show our `cookbook.test_restore` instance is `ACTIVE`:

   ```
   openstack server list -c Name -c Status
   ```

This shows our instance is now running:

```
+------------------------+--------+
| Name                   | Status |
+------------------------+--------+
| cookbook.test_restore  | ACTIVE |
| cookbook.test.03       | ACTIVE |
| cookbook.test.02       | ACTIVE |
| cookbook.test          | ACTIVE |
+------------------------+--------+
```

How it works...

As instance snapshots are stored as OpenStack images, booting from a snapshot is an identical process to booting from an existing image. There are some caveats however. As snapshots are created while an image is running, booting from them is similar to booting a server after a power failure. You must also ensure that you are using a flavor that is of equal or greater size to the original instance—for example, if an instance was originally created as an m1.large, the snapshot must be booted with an m1.large or greater flavor. Additionally, if the instance is a Windows image attached to an Active Directory domain, it may cause issues having two of the same instances running at the same time. To mitigate this, consider booting the instance onto a separate recovery network.

There are tools available to quiesce the filesystem inside an instance and provide a more consistent image that are beyond the scope of this book. However, as guidance, ensuring that a filesystem sync (to flush any disk writes) inside the running instance is recommended before a snapshot is taken.

Rescuing an instance

OpenStack Compute provides a handy troubleshooting tool with rescue mode. Should a user lose an SSH key, or otherwise not be able to boot and access an instance, say, bad iptables settings or failed network configuration, rescue mode will start a minimal instance and attach the disk from the failed instance to aid in recovery. This applies to both Windows and Linux instances as this process essentially allows the mounting of the boot volume of your failed instance as a secondary disk to the rescue instance.

Getting ready

To put an instance into rescue mode, you will need the following information:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* or *ID* of the instance

The instance we will use in this example is `cookbook.test`.

How to do it...

To put an instance into rescue mode, use the following steps:

1. First, we will put the instance into rescue mode as follows:

   ```
   openstack server rescue cookbook.test
   ```

   This will present us with a temporary password we can then use to access the rescue instance:

   ```
   +-----------+--------------+
   | Field     | Value        |
   +-----------+--------------+
   | adminPass | zmWJRw6C5XHk |
   +-----------+--------------+
   ```

2. To verify that an instance is in rescue mode, use the following command:

   ```
   openstack server show cookbook.test -c name -c status
   ```

   This will present the `status` value as `RESCUE`:

   ```
   +--------+---------------+
   | Field  | Value         |
   +--------+---------------+
   | name   | cookbook.test |
   | status | RESCUE        |
   +--------+---------------+
   ```

3. At this point, we can then access this instance, using the `root` username, and the temporary password we were given to perform operating system rescue commands on the mounted filesystem (the filesystem that is the boot volume of our original, broke instance).

When in rescue mode, the disk of the instance in rescue mode is attached as a secondary. In order to access the data on the disk, you will need to mount it. As filesystems differ between OS and deployment, showing the mount process is beyond the scope of this book.

4. To exit rescue mode, use the following command:

```
openstack server unrescue cookbook.test
```

This command will produce no output if successful.

How it works...

The `openstack server rescue` command provides a rescue environment with the disk of your instance attached. First, it powers off the named instance, then it boots the rescue environment attaching the disks of the original instance. Finally, it provides you with the login credentials for the rescue instance.

Accessing the rescue instance is done via SSH. Once logged in to the rescue instance, you can mount the disk using mount `<path to disk> /mnt`.

Once you have completed your troubleshooting or recovery, the `unrescue` command reverses this process; first, stopping the rescue environment and detaching the disk, then booting the instance as it originally was.

Shelving an instance

Somewhat unique to OpenStack Nova is the ability to *shelve* an instance. Instance shelving allows you to stop an instance without having it consume resources. A shelved instance will be retained as a bootable instance, as well as its resources assigned such as IP address, for a configurable amount of time, then deleted. This is useful as part of an instance life cycle process or to conserve resources.

Stopping versus shelving?

Stopping an instance does not free up the amount of resources still available as part of your quota as the assumption is that you will be starting that instance back up again after a short period of time. You would not be able to start a stopped instance if you didn't have any free CPUs or GBs of RAM left of your assigned quota. A stopped instance's resources are still considered *used resources* to the OpenStack Compute scheduler.

Shelving, however, frees up these resources, but still allows you, at a later date, to access the shelved instance. Quota rules and restrictions still apply when unshelving an instance, but shelving an instance will allow you to work within your allotted resource quotas while preserving your instance data.

Getting ready

To shelve an instance, the following information is required:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* or *ID* of the instance

How to do it...

To shelve an instance, the following commands are used:

1. First, we check the status of the instance:

   ```
   openstack server show cookbook.test -c name -c status
   ```

   Ensure that your instance has `status` of `ACTIVE`:

   ```
   +--------+---------------+
   | Field  | Value         |
   +--------+---------------+
   | name   | cookbook.test |
   | status | ACTIVE        |
   +--------+---------------+
   ```

2. To shelve the instance, issue the following command:

   ```
   openstack server shelve cookbook.test
   ```

This command produces no output when successful. Shelving an instance may take a few moments depending on your environment.

3. To check the status of the instance, issue the following command, noting the new status:

```
openstack server show cookbook.test -c name -c addresses -c status
```

The `status` value has changed to `SHELVED_OFFLOADED`:

```
+-----------+--------------------+
| Field     | Value              |
+-----------+--------------------+
| addresses | public=10.1.13.9   |
| name      | cookbook.test      |
| status    | SHELVED_OFFLOADED  |
+-----------+--------------------+
```

A shelved instance will retain the addresses it has been assigned.

4. To unshelve the instance and return it back to an `ACTIVE` state, simply issue the following command:

```
openstack server unshelve cookbook.test
```

This command produces no output when successful. As with shelving, the instance may take a few moments to become active, depending on your environment.

5. We can verify the state by checking the status:

```
openstack server show cookbook.test -c name -c addresses -c status
```

The `status` value has returned to `ACTIVE`:

```
+-----------+------------------+
| Field     | Value            |
+-----------+------------------+
| addresses | public=10.1.13.9 |
| name      | cookbook.test    |
| status    | ACTIVE           |
+-----------+------------------+
```

How it works...

When told to shelve an instance, OpenStack Compute will first stop the instance. It then creates an instance snapshot to retain the state of the instance. The runtime details, such as number of vCPUs, memory, and IP addresses, are retained so that the instance can be unshelved and rescheduled at a later time.

This differs from shutting down an instance, in that the resources of a shutdown instance are still reserved on the host on which it resided, so that it can be powered back on quickly. A shelved instance however, will still show in `openstack server list`, while the resources that were assigned will remain available. Additionally, as the shelved instance will need to be restored from an image, OpenStack Compute will perform placement as if the instance were new, and starting it will take some time.

Reviewing the console logs

Console logs are critical for troubleshooting the startup process of an instance. These logs are produced at boot time, before the console becomes available. Typically, when working with cloud hosted instances, accessing these can be difficult. OpenStack Compute provides a mechanism for accessing the console logs.

Getting ready

To access the console logs of an instance, the following information is required:

- The `openstack` command-line client
- The `openrc` file containing appropriate credentials
- The *name* or *ID* of the instance

For this example, we will view the last five lines of the `cookbook.test` instance.

How to do it...

To show the console logs of an instance, use the following command:

```
openstack console log show --lines 5 cookbook.test
```

This connects to the serial console output of an instance, mimicking the information as if a monitor was directly attached to a physical server:

```
[[0;32m  OK  [0m] Started udev Coldplug all Devices.
[[0;32m  OK  [0m] Started Dispatch Password Requests to Console Directory Watch.
[[0;32m  OK  [0m] Started Set console font and keymap.
[[0;32m  OK  [0m] Created slice system-getty.slice.
[[0;32m  OK  [0m] Found device /dev/ttyS0.
```

How it works...

The `openstack console log show` command collects the console logs, as if you were connected to the server via a serial port or sitting behind the keyboard and monitor at boot time. The command will, by default, return all of the logs generated to that point. To limit the amount of output, the `--lines` parameter can be used to return a specific number of lines from the end of the log.

6
Glance – OpenStack Image Service

In this chapter, we will cover the following topics:

- Introduction to OpenStack Image services
- Managing images
- Using image snapshots
- Using image metadata
- Protecting images
- Deactivating images
- Creating custom images

Introduction to OpenStack Image services

The OpenStack Image service, otherwise known as Glance, is a service that allows users to register, discover, and retrieve virtual machine images for use in an OpenStack cloud. Images made available through the OpenStack Image service can be stored in a variety of formats and backend locations, from local filesystem storage to distributed filesystems such as OpenStack Object Storage (Swift) and Ceph.

The OpenStack Image service is composed of two major components: the `glance-api` service and the `glance-registry` service. Users interface with the `glance-api` service indirectly when performing commands to create, list, delete, or otherwise manage images using OpenStack clients. The `glance-registry` service is responsible for connecting to the backend database and storing or retrieving images.

Managing images

Image management in an OpenStack environment can be achieved using the `openstack` command-line tool, the Horizon dashboard, or directly via the Glance REST-based API. Images can be sourced directly from the internet or created and manipulated with tools such as `virsh`, `virt-manager`, `growpart`, and `cloud-init`. Custom image creation will be covered later in this chapter.

Uploading images

When creating an image in OpenStack, one must provide attributes that describe the image. These attributes include the image name, the disk format, and the container format. Images can be public, private, or shared among multiple projects.

Getting ready

Images for the following examples can be downloaded from the following locations:

- Ubuntu: `https://cloud-images.ubuntu.com/xenial/current/xenial-server-cloudimg-amd64-disk1.img`
- CirrOS: `https://download.cirros-cloud.net/0.3.5/cirros-0.3.5-i386-disk.img`

Over time, image locations on the web can change and the URLs in this book may be unavailable. Feel free to replace any URL in these examples with a known, working URL.

When uploading an image, ensure that you have properly sourced your credentials or are otherwise properly authenticated.

You will need the following details, at a minimum, when uploading an image:

- Image name
- Disk format
- Image location

For our first example, the following will be used:

- Image name: `COOKBOOK_CIRROS_IMAGE`
- Disk format: `qcow2`
- Image location: `/tmp/cirros-0.3.5-i386-disk.img`

For our second example, the following will be used:

- Image name: `COOKBOOK_UBUNTU_IMAGE`
- Disk format: `qcow2`
- Image location: `/tmp/xenial-server-cloudimg-amd64-disk1.img`

How to do it...

With the OpenStack client installed on our system, we are now able to upload an image with the following steps:

1. Download the CirrOS image to the `/tmp` directory:

```
wget https://download.cirros-cloud.net/0.3.5/cirros-0.3.5-i386-
disk.img -O /tmp/cirros-0.3.5-i386-disk.img
```

2. Upload the CirrOS image:

```
openstack image create COOKBOOK_CIRROS_IMAGE \
--disk-format qcow2 \
--file /tmp/cirros-0.3.5-i386-disk.img
```

The output will resemble the following:

```
+------------------+------------------------------------------------------+
| Field            | Value                                                |
+------------------+------------------------------------------------------+
| checksum         | 7316af7358dd32ca1956d72ac2c9e147                     |
| container_format | bare                                                 |
| created_at       | 2017-10-16T12:26:38Z                                 |
| disk_format      | qcow2                                                |
| file             | /v2/images/a4b5074c-3520-439f-9f85-15e78d5d62f2/file |
| id               | a4b5074c-3520-439f-9f85-15e78d5d62f2                 |
| min_disk         | 0                                                    |
| min_ram          | 0                                                    |
| name             | COOKBOOK_CIRROS_IMAGE                                |
| owner            | 4819f952f3d0411183956113f0727b30                     |
| protected        | False                                                |
| schema           | /v2/schemas/image                                    |
| size             | 12528640                                             |
| status           | active                                               |
| tags             |                                                      |
| updated_at       | 2017-10-16T12:26:39Z                                 |
| virtual_size     | None                                                 |
| visibility       | shared                                               |
+------------------+------------------------------------------------------+
```

Repeat the previous steps for the Ubuntu image:

3. Download the Ubuntu image to the `/tmp` directory:

```
wget https://cloud-images.ubuntu.com/xenial/current/xenial-server-cloudimg-amd64-disk1.img -O /tmp/xenial-server-cloudimg-amd64-disk1.img
```

4. Upload the Ubuntu image:

```
openstack image create COOKBOOK_UBUNTU_IMAGE \
--disk-format qcow2 \
--file /tmp/xenial-server-cloudimg-amd64-disk1.img
```

How it works...

Images are created with the following syntax:

```
openstack image create IMAGE_NAME \
--disk-format DISK_FORMAT \
--file FILE_LOCATION \
[--public | --private | --shared | --community]
```

Creating an image with the OpenStack client results in the specified file being uploaded to the OpenStack image repository. The repository can be a local directory or volume specified in the Glance configuration file, or even an object store served by Swift, Ceph, Rackspace Cloud Files, or others.

The `disk-format` parameter defines the format of the virtual disk used by the image. Options include `ami`, `ari`, `aki`, `vhd`, `vmdk`, `raw`, `qcow2`, `vhdx`, `vdi`, `iso`, and `ploop`. The `qcow2` format is popular among OpenStack-based clouds running QEMU/KVM hypervisors, and it supports smaller image file sizes, copy-on-write support, and various compression and encryption technologies. The default format is `raw`.

When uploading images to a Glance store backed by Ceph, the images must be in the `raw` format else they will fail to work properly. The `qemu-img` command can be used to identify the format of an image, and it can also be used to perform the conversion of an image from one format to another.

An example of converting an image in the `qcow2` format to the `raw` format is shown here:

```
qemu-img convert -f qcow2 -O raw image.qcow2 image.raw
```

Additional examples of using the `qemu-img` command can be found in the OpenStack documentation here:

`https://docs.openstack.org/image-guide/convert-images.html`

The `file` parameter defines the local location of the image file respective to where the OpenStack client is being run. The OpenStack client uploads the image to the OpenStack image repository, where it is stored according to settings within Glance.

When specified, the `--public` option marks the image as accessible by all projects within the cloud. Alternatively, the `--private` option marks the image as accessible only by the project that created it. By default, all images created are marked as `shared` and are eligible to be shared with another project. At the time of creation, however, the image can only be seen by the project that created it. The `--shared` and `--community` options are discussed later in this chapter in the *Sharing images* recipe.

Listing images

To list the images in the OpenStack Image service repository, use the following OpenStack client command:

```
openstack image list
```

The output will resemble the following:

```
+--------------------------------------+-----------------------+--------+
| ID                                   | Name                  | Status |
+--------------------------------------+-----------------------+--------+
| a4b5074c-3520-439f-9f85-15e78d5d62f2 | COOKBOOK_CIRROS_IMAGE | active |
| d120a923-5246-4dca-8f52-51a951bffce5 | COOKBOOK_UBUNTU_IMAGE | active |
+--------------------------------------+-----------------------+--------+
```

Viewing image details

The details of individual images can be queried using the following OpenStack client command:

```
openstack image show IMAGE_NAME_OR_ID
```

The output will resemble the following:

```
root@controller-01-utility-container-e2bad038:~# openstack image show COOKBOOK_CIRROS_IMAGE
+------------------+------------------------------------------------------+
| Field            | Value                                                |
+------------------+------------------------------------------------------+
| checksum         | 7316af7358dd32ca1956d72ac2c9e147                     |
| container_format | bare                                                 |
| created_at       | 2017-10-16T12:26:38Z                                 |
| disk_format      | qcow2                                                |
| file             | /v2/images/a4b5074c-3520-439f-9f85-15e78d5d62f2/file |
| id               | a4b5074c-3520-439f-9f85-15e78d5d62f2                 |
| min_disk         | 0                                                    |
| min_ram          | 0                                                    |
| name             | COOKBOOK_CIRROS_IMAGE                                |
| owner            | 4819f952f3d0411183956113f0727b30                     |
| protected        | False                                                |
| schema           | /v2/schemas/image                                    |
| size             | 12528640                                             |
| status           | active                                               |
| tags             |                                                      |
| updated_at       | 2017-10-16T12:26:39Z                                 |
| virtual_size     | None                                                 |
| visibility       | shared                                               |
+------------------+------------------------------------------------------+
```

Deleting images

Images can be deleted from the OpenStack image repository at any time. Keep in mind, however, that depending on the version of OpenStack installed, deleting an image can have an adverse effect on the ability to migrate an instance.

OpenStack releases newer than Kilo should not be impacted by this issue.

Getting ready

When deleting an image, the following information will be necessary:

- Image name or ID

How to do it...

To delete an image in OpenStack, issue the following command:

```
openstack image delete COOKBOOK_CIRROS_IMAGE
```

No output is returned if the operation is successful. To verify that the image is no longer available, use the `openstack image list` or `openstack image show` command.

Downloading images

Images that exist in the OpenStack image repository can be downloaded at a later time and transferred to other systems.

Getting ready

To download an image from the OpenStack image repository, you must have access to the image. You will need the following details, at a minimum, to download the image:

- Image name or ID
- Destination for downloaded file

For our example, the following will be used:

- Image name: `COOKBOOK_UBUNTU_IMAGE`
- Download location: `/tmp/my_downloaded_ubuntu_image.qcow2`

How to do it...

With the OpenStack client installed on our system, we are now able to download an image from the repository with the following command:

```
openstack image save COOKBOOK_UBUNTU_IMAGE \
--file /tmp/my_downloaded_ubuntu_image.qcow2
```

No output is returned if the operation is successful. Use the `ls` command to list the downloaded file:

```
root@controller-01-utility-container-e2bad038:~# ls -la /tmp/*.qcow2
-rw-r--r-- 1 root root 290652160 Oct 16 12:43 /tmp/my_downloaded_ubuntu_image.qcow2
```

Sharing images

When an image is *private*, it is only available to the project from which that image was created or uploaded. On the other hand, when an image is *public*, it is available to all projects. The OpenStack Image service provides a mechanism whereby these private images can be shared between a subset of projects. This allows greater control over images that need to exist for different projects without making them available to all.

Sharing images among projects requires the following workflow:

- Tenant A updates an image's ability to be shared
- Tenant A shares an image with Tenant B
- Tenant B accepts or rejects the sharing attempt

Getting ready

When sharing an image, ensure that you are authenticated as an administrator or are the owner of the image. You will need the following details, at a minimum, to share the image:

- Image name or ID
- Project name or ID

For our example, the `ADMIN` project will share an image with the `FINANCE` project. The following will be used:

- Image name: `COOKBOOK_UBUNTU_IMAGE`
- Project name: `FINANCE`

For instructions on how to create projects within OpenStack, refer to the *Chapter 3*, *Keystone – OpenStack Identity Service*, chapter of this book.

How to do it...

With the OpenStack client installed on our system, we are now able to share an image with the following steps:

1. Configure the environment with the credentials of a user in the `FINANCE` project:

   ```
   source finance_openrc
   ```

2. As a user in the `FINANCE` project, view all available images:

   ```
   openstack image list
   ```

   The output will resemble the following if no images are available:

   ```
   root@controller-01-utility-container-e2bad038:~# openstack image list
   ```

   In this environment, no images are available to the `FINANCE` project.

3. Now, configure the environment with the credentials of a user in the ADMIN project:

   ```
   source openrc
   ```

4. Update the Ubuntu image and make it available for sharing:

   ```
   openstack image set COOKBOOK_UBUNTU_IMAGE --shared
   ```

   No output will be returned.

5. Share the Ubuntu image with the FINANCE project:

   ```
   openstack image add project COOKBOOK_UBUNTU_IMAGE FINANCE
   ```

   The output will resemble the following:

   ```
   +------------+--------------------------------------+
   | Field      | Value                                |
   +------------+--------------------------------------+
   | created_at | 2017-10-16T13:01:06Z                 |
   | image_id   | d120a923-5246-4dca-8f52-51a951bffce5 |
   | member_id  | 55143cf943124d04b2e02db05af907d0     |
   | schema     | /v2/schemas/member                   |
   | status     | pending                              |
   | updated_at | 2017-10-16T13:01:06Z                 |
   +------------+--------------------------------------+
   ```

At this point, the user should notify the `FINANCE` project of the sharing attempt and provide the image ID. In this example, the image ID is `d120a923-5246-4dca-8f52-51a951bffce5`.

6. Reconfigure the environment with the credentials of a user in the `FINANCE` project:

   ```
   source finance_openrc
   ```

7. Accept the sharing attempt:

   ```
   openstack image set d120a923-5246-4dca-8f52-51a951bffce5 --accept
   ```

   No output will be returned.

8. View all available images:

   ```
   openstack image list
   ```

   The output will resemble the following:

```
+--------------------------------------+----------------------+--------+
| ID                                   | Name                 | Status |
+--------------------------------------+----------------------+--------+
| d120a923-5246-4dca-8f52-51a951bffce5 | COOKBOOK_UBUNTU_IMAGE | active |
+--------------------------------------+----------------------+--------+
```

How it works...

Sharing images requires the use of the `openstack image add project` command on the part of the producer and the `openstack image set` command on the part of the consumer. The process involves communication between the producer and the consumer outside of the OpenStack API, as the image ID must be shared between them.

Producers invoke the sharing process using the following command:

```
openstack image add project <image> <project>
```

Once shared, the consumer has the ability to accept or reject the attempt using the following:

```
openstack image set <image> [--accept | --reject | --pending]
```

Marking an image as `pending` makes the image unavailable to users in the consuming project, but leaves open the possibility of making it available at a later time.

The producer can revoke the sharing of an image using the following command:

```
openstack image remove project <image> <project>
```

An alternative to marking an image as shared would be to mark its visibility as `community` using the `--community` option. This allows a user to share an image with all projects, but still requires the consuming projects to accept the image before it is returned in an image list. This reduces the administrative overhead on the part of the user sharing the image without unnecessarily cluttering the entire image list for all projects.

For additional information regarding the sharing of images, visit `https://docs.openstack.org/image-guide/share-images.html`.

Using image snapshots

In OpenStack, a snapshot is an image that reflects the state of an instance at a point in time. Snapshots are often used to backup instances or migrate instances from one cloud to another, and can even be shared with other projects like regular images.

Creating snapshots

Snapshots in OpenStack can be created using the `openstack server image create` command.

Getting ready

When creating a snapshot, ensure that you are authenticated as an administrator or are the owner of the instance. You will need the following details:

- Instance name or ID
- Image name

For our example, the following will be used:

- Instance name: `COOKBOOK_TEST_INSTANCE`
- Image: `COOKBOOK_TEST_SNAPSHOT_20170824`

How to do it...

With the OpenStack client installed on our system, we are now able to create a snapshot with the following command:

```
openstack server image create \
--name COOKBOOK_TEST_SNAPSHOT_20171016 \
COOKBOOK_TEST_INSTANCE
```

The output will resemble the following:

```
+------------------+------------------------------------------------------------------------------------------------------------------+
| Field            | Value                                                                                                            |
+------------------+------------------------------------------------------------------------------------------------------------------+
| checksum         | None                                                                                                             |
| container_format | None                                                                                                             |
| created_at       | 2017-10-16T13:11:59Z                                                                                             |
| disk_format      | None                                                                                                             |
| file             | /v2/images/9c3d3a31-a436-44c4-ba5c-7c26ae38a1de/file                                                             |
| id               | 9c3d3a31-a436-44c4-ba5c-7c26ae38a1de                                                                             |
| min_disk         | 1                                                                                                                |
| min_ram          | 0                                                                                                                |
| name             | COOKBOOK_TEST_SNAPSHOT_20171016                                                                                  |
| owner            | 4819f952f3d0411183956113f0727b30                                                                                 |
| properties       | base_image_ref='a4b5074c-3520-439f-9f85-15e78d5d62f2', boot_roles='heat_stack_owner,admin', image_type='snapshot', |
|                  | instance_uuid='d92eaf20-ee99-4e5d-9a31-f987016446ad', owner_project_name='admin', owner_user_name='admin', user_id='2ce040fef90e4ee69b1affadcb6491e0' |
| protected        | False                                                                                                            |
| schema           | /v2/schemas/image                                                                                                |
| size             | None                                                                                                             |
| status           | queued                                                                                                           |
| tags             |                                                                                                                  |
| updated_at       | 2017-10-16T13:11:59Z                                                                                             |
| virtual_size     | None                                                                                                             |
| visibility       | private                                                                                                          |
+------------------+------------------------------------------------------------------------------------------------------------------+
```

How it works...

Images snapshots are created with the following syntax:

```
openstack server image create [--name <image-name>] \
[--wait] \
<server>
```

The creation of a snapshot results in the system writing the instance's disk to a temporary file on the compute node and uploading it to the OpenStack Image service as a new image. When an instance's disk is located on a Ceph backend, the snapshot occurs on the backend itself and the compute node filesystem is not involved. The resulting snapshot/image can then be used to boot new instances within the same cloud, or it can be downloaded and copied to an offline storage system or another cloud.

The `name` parameter defines the name of the image when stored in the image repository.

The `<server>` parameter defines the name or ID of the instance from which the snapshot is taken.

When specified, the `--wait` option forces OpenStack to perform the snapshot in the foreground, meaning the CLI will be unavailable until the task is complete.

Using image metadata

Images have properties known as **metadata** that help describe the image and how it relates to other OpenStack components. The metadata that is applied to an image can be used to enable specific functionality in other OpenStack services or to determine the scheduling of an instance to a host based on CPU architectures or features, and more.

Setting image metadata

Image metadata in OpenStack can be manipulated using the `openstack image set` command. The `--property` argument can be used to set additional properties using the `key=value` pairs.

Getting ready

When setting image metadata, ensure that you are authenticated as an administrator or are the owner of the image. You will need the following details, at a minimum:

- Image name or ID
- Property name and value

For our examples, the following will be used:

- Image name: `COOKBOOK_UBUNTU_IMAGE`
- Property name and value: `architecture=m68k, os_distro=ubuntu`

How to do it...

With the OpenStack client installed on our system, we are now able to set image metadata with the following command:

```
openstack image set COOKBOOK_UBUNTU_IMAGE \
--property architecture=m68k \
--property os_distro=ubuntu
```

No output is returned if the operation is successful. However, using `openstack image show` will reveal that the properties have been set:

```
+------------------+------------------------------------------------------+
| Field            | Value                                                |
+------------------+------------------------------------------------------+
| checksum         | e6f42cddf356ae2a2633e91e20a9916b                     |
| container_format | bare                                                 |
| created_at       | 2017-10-16T12:34:21Z                                 |
| disk_format      | qcow2                                                |
| file             | /v2/images/d120a923-5246-4dca-8f52-51a951bffce5/file |
| id               | d120a923-5246-4dca-8f52-51a951bffce5                 |
| min_disk         | 0                                                    |
| min_ram          | 0                                                    |
| name             | COOKBOOK_UBUNTU_IMAGE                                |
| owner            | 4819f952f3d0411183956113f0727b30                     |
| properties       | architecture='m68k', os_distro='ubuntu'              |
| protected        | False                                                |
| schema           | /v2/schemas/image                                    |
| size             | 290652160                                            |
| status           | active                                               |
| tags             |                                                      |
| updated_at       | 2017-10-16T13:21:57Z                                 |
| virtual_size     | None                                                 |
| visibility       | shared                                               |
+------------------+------------------------------------------------------+
```

How it works...

When image metadata is set in OpenStack, the properties have an impact on how the system handles instances using that image. Properties such as `hypervisor_type` and `architecture` affect how an instance is scheduled to a host, while other properties such as `hw_disk_bus` and `hw_cdrom_bus` affect how virtual devices are connected to the instance.

In order to schedule instances based on image metadata, be sure to include the `'ImagePropertiesFilter'` filter in the `enabled_filters` list in `/etc/nova/nova.conf`. This is a default in most OpenStack installations, including the environment built in this book.

In this environment, booting an instance using the modified image requiring an architecture of `m68k` (Motorola 68000) should result in the following error:

```
NoValidHost: No valid host was found. There are not enough hosts
available.
```

Modifying the image to require `x86_64` or removing the property altogether should allow the instance to be scheduled according to other defined metadata or environmental defaults.

A current list of image metadata properties at time of writing can be found at the following location:

`https://docs.openstack.org/python-glanceclient/latest/cli/property-keys.html`

Removing image metadata

Image metadata in OpenStack can be removed using the `openstack image unset` command. The `--property` argument can be used to unset individual properties.

Getting ready

When removing image metadata, ensure that you are authenticated as an administrator or are the owner of the image. You will need the following details, at a minimum:

- Image name or ID
- Property name

For our examples, the following will be used:

- Image name: `COOKBOOK_UBUNTU_IMAGE`
- Property name: `architecture`

How to do it...

With the OpenStack client installed on our system, we are now able to unset image metadata with the following command:

```
openstack image unset COOKBOOK_UBUNTU_IMAGE \
--property architecture
```

No output is returned if the operation is successful. To verify that the property has been removed, use the `openstack image show` command.

Protecting images

Like other OpenStack objects, images and snapshots are susceptible to accidental deletion by users. By default, images are unprotected, meaning they can be deleted at any time by a user within a project. The following sections explain how an image can be protected to ensure its survival.

Protecting an image

Images can be protected using the `openstack image set` command with the `--protected` argument.

Getting ready

When protecting an image, ensure that you are authenticated as an administrator or are the owner of the image. You will need the following details, at a minimum:

- Image name or ID

For our examples, the following will be used:

- Image name: `COOKBOOK_UBUNTU_IMAGE`

How to do it...

With the OpenStack client installed on our system, we are now able to protect an image with the following command:

```
openstack image set COOKBOOK_UBUNTU_IMAGE --protected
```

No output is returned if the operation is successful. Use the `openstack image show` command to reveal the status of the image:

```
+------------------+------------------------------------------------------+
| Field            | Value                                                |
+------------------+------------------------------------------------------+
| checksum         | e6f42cddf356ae2a2633e91e20a9916b                     |
| container_format | bare                                                 |
| created_at       | 2017-10-16T12:34:21Z                                 |
| disk_format      | qcow2                                                |
| file             | /v2/images/d120a923-5246-4dca-8f52-51a951bffce5/file |
| id               | d120a923-5246-4dca-8f52-51a951bffce5                 |
| min_disk         | 0                                                    |
| min_ram          | 0                                                    |
| name             | COOKBOOK_UBUNTU_IMAGE                                |
| owner            | 4819f952f3d0411183956113f0727b30                     |
| properties       | architecture='m68k', os_distro='ubuntu'              |
| protected        | True                                                 |
| schema           | /v2/schemas/image                                    |
| size             | 290652160                                            |
| status           | active                                               |
| tags             |                                                      |
| updated_at       | 2017-10-16T13:53:14Z                                 |
| virtual_size     | None                                                 |
| visibility       | shared                                               |
+------------------+------------------------------------------------------+
```

How it works...

When an image is protected in OpenStack, users are unable to delete the image. Attempting to delete a protected image results in an error similar to the following:

```
Failed to delete image with name or ID 'COOKBOOK_UBUNTU_IMAGE': 403
Forbidden: Image d120a923-5246-4dca-8f52-51a951bffce5 is protected and
cannot be deleted. (HTTP 403)
Failed to delete 1 of 1 images.
```

Protecting an image is a useful step in ensuring that snapshots and other special images remain unharmed in a cloud shared by many users and projects.

Unprotecting an image

Images can be unprotected using the `openstack image set` command with the `--unprotected` argument.

Getting ready

When unprotecting an image, ensure that you are authenticated as an administrator or are the owner of the image. You will need the following details, at a minimum:

- Image name or ID

For our examples, the following will be used:

- Image name: `COOKBOOK_UBUNTU_IMAGE`

How to do it...

With the OpenStack client installed on our system, we are now able to unprotect an image with the following command:

```
openstack image set COOKBOOK_UBUNTU_IMAGE --unprotected
```

No output is returned if the operation is successful. Use the `openstack image show` command to reveal the status of the image.

Deactivating images

By default, images are available for use immediately upon completion of the uploading process. At times, it may be necessary to deactivate an image to prevent users from booting instances with the image, especially in cases where the image may be outdated but must be retained for archival purposes.

Deactivating an image

Images can be deactivated using the `openstack image set` command with the `--deactivate` argument.

Getting ready

When deactivating an image, ensure that you are authenticated as an administrator or are the owner of the image. You will need the following details, at a minimum:

- Image name or ID

For our examples, the following will be used:

- Image name: `COOKBOOK_UBUNTU_IMAGE`

How to do it...

With the OpenStack client installed on our system, we are now able to deactivate an image with the following command:

```
openstack image set COOKBOOK_UBUNTU_IMAGE --deactivate
```

No output is returned if the operation is successful. Use the `openstack image show` command to reveal the status of the image:

```
+------------------+------------------------------------------------------+
| Field            | Value                                                |
+------------------+------------------------------------------------------+
| checksum         | e6f42cddf356ae2a2633e91e20a9916b                     |
| container_format | bare                                                 |
| created_at       | 2017-10-16T12:34:21Z                                 |
| disk_format      | qcow2                                                |
| file             | /v2/images/d120a923-5246-4dca-8f52-51a951bffce5/file |
| id               | d120a923-5246-4dca-8f52-51a951bffce5                 |
| min_disk         | 0                                                    |
| min_ram          | 0                                                    |
| name             | COOKBOOK_UBUNTU_IMAGE                                |
| owner            | 4819f952f3d0411183956113f0727b30                     |
| properties       | architecture='m68k', os_distro='ubuntu'              |
| protected        | False                                                |
| schema           | /v2/schemas/image                                    |
| size             | 290652160                                            |
| status           | deactivated                                          |
| tags             |                                                      |
| updated_at       | 2017-10-17T13:19:46Z                                 |
| virtual_size     | None                                                 |
| visibility       | shared                                               |
+------------------+------------------------------------------------------+
```

How it works...

When an image is deactivated in OpenStack, users are unable to boot instances using the image. Attempting to boot an instance with a deactivated image results in an error similar to the following:

```
Image d120a923-5246-4dca-8f52-51a951bffce5 is not active. (HTTP 400)
(Request-ID: req-fb43f34c-982b-4eeb-abb1-76c93ebc2b5f)
```

Deactivating an image is a useful step in ensuring that images are retained but unusable. Existing instances using the image are not impacted.

Activating an image

Deactivated images can be activated using the `openstack image set` command with the `--activate` argument.

Getting ready

When activating an image, ensure that you are authenticated as an administrator or are the owner of the image. You will need the following details, at a minimum:

- Image name or ID

For our examples, the following will be used:

- Image name: `COOKBOOK_UBUNTU_IMAGE`

How to do it...

With the OpenStack client installed on our system, we are now able to activate an image with the following command:

```
openstack image set COOKBOOK_UBUNTU_IMAGE --activate
```

No output is returned if the operation is successful. Use the `openstack image show` command to reveal the status of the image.

Creating custom images

Users can create custom images of various operating systems that can be used within an OpenStack environment. Tools such as `cloud-init` can be installed in the image to provide a method of bootstrapping an instance once it has been deployed.

The use of `cloud-init` is beyond the scope of this book. More information can be found at `https://cloud-init.io`.

Getting ready

To begin, ensure that you are using an operating system that is not the OpenStack environment used throughout this book. Software packages and libraries needed to create images may conflict with the software currently installed, and could result in a broken environment. In this example, we will use a virtual machine configured with Ubuntu 16.04 LTS to create a custom CentOS 7 image.

Configuring a virtual machine with the Ubuntu 16.04 LTS operating system is beyond the scope of this book. A physical server can be used in lieu of a virtual machine, if necessary.

The following packages are prerequisites for the host building of the image:

- `qemu-kvm`
- `libvirt-bin`
- `virt-manager`

Using `apt`, install the packages with the following commands:

```
sudo apt update
sudo apt install qemu-kvm libvirt-bin virt-manager
```

How to do it...

Carry out the following steps within the virtual machine to create a custom image:

1. Change to your home directory and create a kickstart file named `ks.cfg` with the following contents:

```
cd ~/
install
text
url --url http://mirror.rackspace.com/CentOS/7/os/x86_64/
lang en_US.UTF-8
keyboard us
network --onboot yes --bootproto dhcp --noipv6
timezone --utc America/Chicago
```

```
zerombr
clearpart --all --initlabel
bootloader --location=mbr --append="crashkernel=auto"
part / --fstype=ext4 --size=1024 --grow
authconfig --enableshadow --passalgo=sha512
rootpw openstack
firewall --disable
selinux --disabled
skipx
shutdown
%packages
@core
openssh-server
openssh-clients
wget
curl
git
man
vim
ntp
%end
%post
%end
```

2. Create an empty 10 GB virtual disk to be used by the CentOS virtual machine:

```
sudo qemu-img create -f qcow2 \
/var/lib/libvirt/images/centos-7.qcow2 10G
```

3. Execute the following commands to initiate an unattended installation of the CentOS operating system:

```
sudo virt-install --virt-type qemu \
--name centos-7 \
--ram 2048 \
--location="http://mirror.rackspace.com/CentOS/7/os/x86_64/" \
--disk /var/lib/libvirt/images/centos-7.qcow2,format=qcow2 \
--network network=default \
--graphics vnc,listen=0.0.0.0 \
```

```
--noautoconsole \
--os-type=linux \
--os-variant=centos7.0 \
--initrd-inject ks.cfg \
--rng /dev/random \
--extra-args="inst.ks=file:/ks.cfg console=ttyS0,115200"
```

If the installation is being performed inside a virtual machine using nested virtualization, you may need to change `virt-type` to `qemu` from `kvm` for the machine to start properly. Use `kvm` for better performance if the image is being built without nested virtualization.

The output returned should resemble the following:

```
Starting install...
Retrieving file vmlinuz...                    | 5.1 MB  00:00:01
Retrieving file initrd.img...                 |  41 MB  00:00:07
Allocating 'centos-7.0-vm.img'                | 5.0 GB  00:00:00
Creating domain...                            |    0 B  00:00:00
Domain installation still in progress. You can reconnect to
the console to complete the installation process.
```

From the host machine, console to the newly-created CentOS virtual machine and log in as the `root` user with the `openstack` password. To escape from the console session, press **Ctrl** + **]**:

```
virsh console centos-7
```

To refresh the console, press the **Enter** key. The output will resemble the following:

```
...
[   42.133751] 8021q: 802.1Q VLAN Support v1.8
[   43.365590] dracut-initqueue[564]: RTNETLINK answers: File exists
[   45.471515] 8021q: adding VLAN 0 to HW filter on device eth0
[   56.147404] dracut-initqueue[564]: % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
[   56.164660] dracut-initqueue[564]: Dload  Upload   Total   Spent    Left  Speed
 6  317M    6 19.7M    0     0   811k      0  0:06:40  0:00:24  0:06:16  625k  0 --:--:-- --:--:-- --:--:--     0
17  317M   17 55.3M    0     0  1113k      0  0:04:51  0:00:50  0:04:05 1787k
24  317M   24 79.2M    0     0  1055k      0  0:05:07  0:01:15  0:03:50  628k
31  317M   31 98.5M    0     0   980k      0  0:05:31  0:01:41  0:03:56 1459k
...
Starting automated install.................................
Checking software selection
...
Running pre-installation scripts
.
Starting package installation process
...
```

The guest should be in the process of booting, and a live console log will be displayed. The installation process is automated and could take a while depending on the resources available to the host performing the build.

When the installation is complete, the output will resemble the following:

```
[  OK  ] Reached target Shutdown.
dracut Warning: Killing all remaining processes
Powering off.
[ 3388.611074] Power down.
```

The console session should end, and you will be returned to a prompt on the host.

4. Start the VM with the following command:

```
sudo virsh start centos-7
```

From the host machine, console to the CentOS virtual machine and log in as the `root` user with the `openstack` password. To escape from the console session, press **Ctrl** + **]**:

```
virsh console centos-7
```

To refresh the console, press the **Enter** key. The output will resemble the following:

```
Connected to domain centos-7
Escape character is ^]

CentOS Linux 7 (Core)
Kernel 3.10.0-693.el7.x86_64 on an x86_64

localhost login: root
Password:
Last login: Wed Aug 30 09:02:14 on tty1
[root@localhost ~]#
```

5. In the guest, install the `epel-release` and `cloud-init` packages with the following commands:

```
yum -y install epel-release
yum -y install cloud-init cloud-utils cloud-utils-growpart
```

6. Using a text editor, replace the guest's `cloud.cfg` file at `/etc/cloud/cloud.cfg` with the following contents:

```
users:
- default

disable_root: 1
```

```
ssh_pwauth:   0

locale_configfile: /etc/sysconfig/i18n
mount_default_fields: [~, ~, 'auto', 'defaults,nofail', '0', '2']
resize_rootfs_tmp: /dev
ssh_deletekeys:   0

ssh_genkeytypes:  ~
syslog_fix_perms: ~

cloud_init_modules:
- bootcmd
- write-files
- resizefs
- set_hostname
- update_hostname
- update_etc_hosts
- rsyslog
- users-groups
- ssh

cloud_config_modules:
- mounts
- locale
- set-passwords
- timezone
- puppet
- chef
- salt-minion
- mcollective
- disable-ec2-metadata
- runcmd

cloud_final_modules:
- rightscale_userdata
- scripts-per-once
- scripts-per-boot
- scripts-per-instance
- scripts-user
- ssh-authkey-fingerprints
- keys-to-console
- phone-home
```

```
- final-message

system_info:
 distro: rhel
 default_user:
   name: centos
   lock_passwd: True
   shell: /bin/bash
   sudo: ["ALL=(ALL) NOPASSWD: ALL"]
 paths:
   cloud_dir: /var/lib/cloud
   templates_dir: /etc/cloud/templates
 ssh_svcname: sshd
```

The contents of `cloud.cfg` are constructed in YAML, and are interpreted by `cloud-init` upon startup. In this example, the default username used to access the instance is `centos` and no password is set. Instead, an SSH key must be used and will be pushed to the instance via the OpenStack metadata service. Refer to *Chapter 5*, *Nova – OpenStack Compute*, for more information.

7. Ensure that the guest can communicate with the metadata service with the following command:

   ```
   echo "NOZEROCONF=yes" >> /etc/sysconfig/network
   ```

8. Remove persistent rules with the following command:

   ```
   rm -rf /etc/udev/rules.d/70-persistent-net.rules
   ```

9. Remove machine-specific MAC address and UUID. Edit the `/etc/sysconfig/network-scripts/ifcfg-eth0` file and remove lines starting with `HWADDR` and `UUID`:

   ```
   sed -i '/HWADDR/d' /etc/sysconfig/network-scripts/ifcfg-eth0
   sed -i '/UUID/d' /etc/sysconfig/network-scripts/ifcfg-eth0
   ```

10. After making the changes, ensure that the interface configuration file resembles the following:

```
NAME="eth0"
ONBOOT="yes"
NETBOOT="yes"
IPV6INIT="no"
BOOTPROTO="dhcp"
TYPE="Ethernet"
DEFROUTE="yes"
PEERDNS="yes"
PEERROUTES="yes"
IPV4_FAILURE_FATAL="no"
```

If you need IPv6 support, modify the interface file accordingly to ensure that instances using the image can procure their IPv6 address via DHCPv6 or SLAAC.

11. Clean up various files and directories using the following commands:

```
yum clean all
rm -rf /var/log/*
rm -rf /tmp/*
history -c
```

12. From within the guest, cleanly shutdown the guest using the following command:

```
shutdown -h now
```

Once the guest is shutdown, the console session should end and you will be returned to a prompt on the host. Transfer the disk located at `/var/lib/libvirt/images/centos-7.qcow2` to your home directory, where it can be transferred out of the host and onto a client, where the OpenStack command-line utility is installed. Using the OpenStack client, upload the image to the OpenStack image repository:

```
openstack image create MY_CENTOS_IMAGE \
--disk-format qcow2 \
--file ~/centos-7.qcow2
```

The output should resemble the following:

Field	Value
checksum	14268d0aa56d24a1336955ede3759f7b
container_format	bare
created_at	2017-10-17T18:48:28Z
disk_format	qcow2
file	/v2/images/c61095ad-6029-4a86-9af9-7cd02934b7d8/file
id	c61095ad-6029-4a86-9af9-7cd02934b7d8
min_disk	0
min_ram	0
name	MY_CENTOS_IMAGE
owner	4819f952f3d0411183956113f0727b30
protected	False
schema	/v2/schemas/image
size	1811021824
status	active
tags	
updated_at	2017-10-17T18:48:40Z
virtual_size	None
visibility	shared

For more information on building custom images, visit `https://docs.openstack.org/image-guide/create-images-manually.html`.

7
Cinder – OpenStack Block Storage

In this chapter, we will cover the following topics:

- Configuring Cinder volume services
- Creating volumes
- Attaching volumes to an instance
- Detaching volumes from an instance
- Deleting volumes
- Working with volume snapshots
- Configuring volume types
- Enabling volume encryption
- Configuring volume Quality of Service (QoS)
- Resetting volume state

Introduction

When a basic compute instance is launched, where the instance data resides on the compute host's disks for the duration of the running instance, the data written to it is not persistent after termination—meaning that any data saved on the disk will be lost when a user requests to destroy that instance. There is a solution for this in OpenStack. **Volumes** are persistent storage that you can attach to your running OpenStack Compute instances; the best analogy is that of a USB drive that you can attach to an instance. Like USB drives, you can only attach instances to one computer at a time.

There is currently an experimental feature that allows you to attach a volume to multiple instances. We do not cover it here nor recommend its usage at this time.

The OpenStack Block Storage project code name Cinder provides the interfaces and automation that allows the connection of storage volumes to OpenStack Compute instances. OpenStack Block Storage is very similar to Amazon **Elastic Block Storage** (**EBS**)—the primary difference is in how volumes are presented to the running instances. Under OpenStack Compute, one method is to dedicate a server to this purpose so that volumes that can easily be managed using an iSCSI exposes the LVM volume group, specifically named `cinder-volumes`. This is then presented over iSCSI through an OpenStack service called `cinder-volume`. The users of OpenStack interact with this service through the Cinder API. In our environment, the Cinder API service runs on the three controller servers that are then usually exposed behind a load balancer. The recipes in this chapter are shown using this same method, whereby Cinder volumes are provided by LVM and iSCSI. However, Cinder supports a wide variety of third-party storage providers by both commercial vendors and the **Open Source Software** (**OSS**) community—for example, a very popular backend provider for Cinder (instead of having a single server run the `cinder-volume` service) is **Ceph**.

The terms OpenStack Block Storage and Cinder will be used interchangeably in this chapter.

Configuring Cinder volume services

In this recipe, we will use Openstack-Ansible to deploy the Cinder service. We assume that your environment has been deployed using the recipes described in *Chapter 1*, *Installing OpenStack with Ansible*.

Getting ready

To use Cinder volumes with LVM and iSCSI, you will need a host (or hosts) running Ubuntu 16.04. Additionally, you will need to ensure the volume hosts are manageable by your Openstack-Ansible deployment host.

The following is required:

- An `openrc` file with appropriate credentials for the environment
- Access to the `openstack-ansible` deployment host
- A dedicated server to provide Cinder volumes to instances:
  - An LVM volume group, specifically named `cinder-volumes`
  - An IP address accessible from the deployment host

If you are using the lab environment that accompanies this book, `cinder-volume` is deployed to a host with the following details:

- `cinder-volume` API service host IP: `172.29.236.10`
- `cinder-volume` service host IP: OpenStack management calls over `172.29.236.100` and storage traffic over `172.29.244.100`

How to do it...

To deploy the `cinder-volume` service, first we will create a YAML file to describe how to deploy Cinder. Then we will use `openstack-ansible` to deploy the appropriate services.

To configure a Cinder LVM host that runs the `cinder-volume` service, perform the following steps on the deployment host:

1. First, edit the `/etc/openstack_deploy/openstack_user_config.yml` file to add the following lines, noting that we are using both the address of the API and storage networks:

```
---
storage-infra_hosts:
  controller-01:
    ip: 172.29.236.10

storage_hosts:
  cinder-volume:
    ip: 172.29.236.100
    container_vars:
      cinder_backends:
        limit_container_types: cinder_volume
        lvm:
          volume_group: cinder-volumes
          volume_driver: cinder.volume.drivers.lvm.LVMVolumeDriver
          volume_backend_name: LVM_iSCSI
          iscsi_ip_address: "172.29.244.100"
```

More nodes can be added to the `storage-infra_hosts` section to match your environment. Note that we refer to the `storage_hosts` and `storage-infra_hosts` IPs from the container network (`172.29.236`), and we will present the actual iSCSI volume to an instance over the storage network (`172.29.244`).

2. Now edit the `/etc/openstack_deploy/user_variables.yml` file to contain the following lines. The defaults for an OpenStack-Ansible deployment are shown here; so edit to suit your environment if necessary:

```
## Cinder iscsi
cinder_iscsi_helper: tgtadm
cinder_iscsi_iotype: fileio
cinder_iscsi_num_targets: 100
cinder_iscsi_port: 3260
```

3. When using LVM, as we are here, we must tell OpenStack-Ansible to *not* deploy the service onto a container (by setting `is_metal: true`). Ensure that the `/etc/openstack_deploy/env.d/cinder.yml` file has the following contents:

```
---
container_skel:
  cinder_volumes_container:
    properties:
      is_metal: true
```

4. As we are specifically choosing to install Cinder with the `LVMVolumeDriver` service as part of this example, we must ensure that the host (or hosts) that has been set as running the `cinder-volume` service, has a **volume group** created, named very specifically `cinder-volumes`, *before* we deploy Cinder. Carry out the following steps to create this important volume group. The following example simply assumes that there is an extra disk, at `/dev/sdb`, that we will use for this purpose:

```
pvcreate /dev/sdb
vgcreate cinder-volumes /dev/sdb
```

   This will bring back an output like the following:

```
Physical volume "/dev/sdb1" successfully created
Volume group "cinder-volumes" successfully created
```

Tip: The creation of a `cinder-volumes` logical **Volume Group** (**VG**) is only required to be created because we are choosing the `volume_driver` type of `cinder.volume.drivers.lvm.LVMVolumeDriver`. Other backends are available to Cinder that do not require this step to be performed.

5. We can now deploy our Cinder service with the `openstack-ansible` command:

```
cd /opt/openstack-ansible/playbooks
openstack-ansible os-cinder-install.yml
```

Note that the Ansible output has been omitted here. On success, Ansible will report that the installation has been successful or will present you information about which step failed.

Did you use more than one `storage-infra_hosts`? These are the Cinder API servers. As these run behind a load balancer, ensure that you have updated your load balancer VIPs with the IP addresses of these new nodes. The Cinder service runs on port `8776`. If you are running HAProxy that was installed using OpenStack-Ansible, you must also run the following command:

```
openstack-ansible haproxy-install.yml
```

6. Verify that the Cinder services are running with the following checks from an OpenStack client or one of the utility containers from one of the controller nodes, as shown here:

```
lxc-attach --name $(lxc-ls -1 | grep util)
source openrc
cinder service-list
```

This will give back an output like the following:

```
+------------------+----------------------------------------------------+------+---------+-------+----------------------------+-----------------+
| Binary           | Host                                               | Zone | Status  | State | Updated_at                 | Disabled Reason |
+------------------+----------------------------------------------------+------+---------+-------+----------------------------+-----------------+
| cinder-scheduler | controller-01-cinder-scheduler-container-18ca83af | nova | enabled | up    | 2017-09-22T20:15:29.000000 | -               |
| cinder-volume    | cinder-volume@lvm                                  | nova | enabled | up    | 2017-09-22T20:15:29.000000 | -               |
+------------------+----------------------------------------------------+------+---------+-------+----------------------------+-----------------+
```

How it works...

In order for us to use a host as a `cinder-volume` server, we first needed to ensure that the logical volume group (LVM VG) named `cinder-volumes` has been created.

Tip: You are able to rename the volume group to something other than `cinder-volumes`; however, there are very few reasons to do so. If you do require this, ensure that the `volume_group:` parameter in `/etc/openstack_deploy/openstack_user_config.yml` matches your LVM Volume Group name that you create.

Once that has been done, we will configure our OpenStack-Ansible deployment to specify our `cinder-volume` server (as denoted by the `storage_hosts` section) and the servers that run the API service (as denoted by the `storage-infra_hosts` section). We will then use `openstack-ansible` to deploy the packages onto the controller hosts and our nominated Cinder volume server.

Note that if you are using multiple networks or VLAN segments, to configure your OpenStack-Ansible deployment accordingly. The storage network presented in this book should be used for this iSCSI traffic (that is to say, when a volume attaches to an instance), and it is separate from the container network that is reserved for API traffic and interservice traffic.

In our example, `cinder-volume` uses iSCSI as the mechanism for attaching a volume to an instance; `openstack-ansible` then installs the packages that are required to run iSCSI targets.

OpenStack-Ansible provides the additional benefit of deploying any changes required to support Cinder. This includes creating the service within Keystone and configuring Nova to support volume backends.

Creating volumes

Now that we have created a Cinder volume service, we can now create volumes for use by our instances. We do this under our client environment using the Python-OpenStack client with the `python-cinderclient` library; so we are creating volumes specific to our project (tenant).

Refer to *Chapter 2*, *The OpenStack Client*, for more information on installing and configuring the OpenStack client.

Getting ready

To create a volume, the following is required:

- The `openstack` command line client
- An `openrc` file with appropriate credentials for the environment
- The desired *name* for the volume
- The desired *size* in GiB for the volume

For our example, these are the following:

- Volume name: `cookbook.volume`
- Size: 10 GiB

How to do it...

Carry out the following steps to create a volume:

We simply create the volume that we will attach to our instance with the following command:

```
openstack volume create
    --size 10
    --description "Cookbook Volume"
    cookbook.volume
```

On completion, the command returns the following output:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| consistencygroup_id | None                                 |
| created_at          | 2017-09-22T20:25:16.943250           |
| description         | Cookbook Volume                      |
| encrypted           | False                                |
| id                  | bb7a7d2c-8069-4924-a18e-8fb398584a5d |
| migration_status    | None                                 |
| multiattach         | False                                |
| name                | cookbook.volume                      |
| properties          |                                      |
| replication_status  | None                                 |
| size                | 10                                   |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | available                            |
| type                | None                                 |
| updated_at          | None                                 |
| user_id             | 7d79ffa6b2614377b08339be7eb62af9     |
+---------------------+--------------------------------------+
```

How it works...

Creating volumes is very straightforward. Using the `openstack` client, we supply the `volume` context and the `create` action with the following syntax:

```
openstack volume create
    --size size_GiB
    --description "meaningful description"
    volume_name
```

Here, `volume_name` can be any arbitrary name with no spaces.

As we are using a server that runs the `cinder-volume` service, we can see the actual LVM volumes on `cinder-volumes`, using the usual LVM tools, as follows (ensure that you are logged in as `root` on the server we specified as storage host):

```
lvdisplay cinder-volumes

  --- Logical volume ---
  LV Path                /dev/cinder-volumes/volume-bb7a7d2c-8069-4924-
a18e-8fb398584a5d
  LV Name                volume-bb7a7d2c-8069-4924-a18e-8fb398584a5d
  VG Name                cinder-volumes
  LV UUID                XqZY27-Ei32-EEdC-3UtE-MR6e-2EKw-XCvIGt
  LV Write Access        read/write
  LV Creation host, time cinder-volume, 2017-09-22 20:25:17 +0000
  LV Status              available
  # open                 0
  LV Size                10.00 GiB
  Current LE             2560
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           252:2
```

There's more...

By default, Cinder volumes operate like a physical disk, insofar as they can only be attached to one instance at a time. However, for workloads that require a disk be shared between instances, you can pass the `--multi-attach` flag when creating the volume to enable the volume to be attached to more than once instance:

```
openstack volume create
    --multi-attach
    --description "description"
    --size size_GiB
    volume_name
```

A word of warning

This feature is quite new and considered experimental for production environments, and is not a replacement for a shared storage service such as NFS. NFS supports locking that allows multiple clients to read and write to the same mount point. Multi-attach block storage does not support locking. Therefore, a valid use case could be a scenario where a master/slave service is employed and where the data should only be written to by only one instance at a time, but there is a benefit to having the data immediately available on failover.

Attaching volumes to an instance

Now that we have a usable volume, we can attach it to any instance. We'll do this using the `openstack server volume add` command.

Getting ready

To attach a volume to an instance, you will need the following:

- The `openstack` command-line client
- An `openrc` file with appropriate credentials for the environment
- The *name* or *ID* of the *volume* to attach
- The *name* or *ID* of the *instance* to attach the volume to

For our example, these values are as follows:

- Volume: `cookbook.volume`
- Instance: `cookbook.test`

How to do it...

Carry out the following steps to attach a volume to an instance using the `openstack` client:

1. First, let's list running instances to get the ID of our instance:

```
openstack server list -c Name -c ID -f table
```

An example showing our running instance called `cookbook.test` is shown here:

```
+--------------------------------------+---------------+
| ID                                   | Name          |
+--------------------------------------+---------------+
| 90654098-477f-417f-b102-453c0f1f9119 | cookbook.test |
+--------------------------------------+---------------+
```

2. Now list the available volumes to get the ID of our volume:

```
openstack volume list
```

This shows the information we need about our volume:

```
+--------------------------------------+-----------------+-----------+------+-------------+
| ID                                   | Name            | Status    | Size | Attached to |
+--------------------------------------+-----------------+-----------+------+-------------+
| daeddd6c-908a-4ea8-8632-d93d198c2621 | cookbook.volume | available |   10 |             |
+--------------------------------------+-----------------+-----------+------+-------------+
```

3. Using the *instance* and *volume* name or IDs, we'll attach the volume to the instance as follows:

```
openstack server add volume
    cookbook.test
    cookbook.volume
    --device /dev/vdc
```

This command produces no output when successful.

Note that the `--device` option is not always honored, depending on the operating system and image type. Always check which device the target instance operating system assigns to the new volume before performing any actions on it.

Tip: Volume or server names do not have to be unique in OpenStack; where volume and server names are not unique, replace the names with the IDs assigned instead. In the preceding example, this would achieve the same goal:

```
openstack server add volume
    90654098-477f-417f-b102-453c0f1f9119
    daeddd6c-908a-4ea8-8632-d93d198c2621
    --device /dev/vdb
```

4. Now we will perform actions inside our running instance. Log in to the instance and verify that the volume is now attached:

```
lsblk
```

This will list the block devices available to our instance. Here we can see that our volume, attached as `/dev/vdb`, is available but not mounted:

```
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0     11:0    1  558K  0 rom
vda    253:0    0  2.2G  0 disk
`-vda1 253:1    0  2.2G  0 part /
vdb    253:16   0   10G  0 disk
```

5. We should see `10G` of space available for use by the running instance. As this is a new volume, this is like adding a fresh disk to a system. We need to format it for use and then mount it as part of your filesystem:

```
sudo mkfs.ext4 /dev/vdb
sudo mkdir /mnt1
sudo mount /dev/vdb /mnt1
```

Tip: Volumes created with Cinder are persistent storage volumes; formatting the volume (disk) is only required once. Should you need to re-attach this data volume to another instance in the future, do not reformat this drive!

6. We should now see the newly attached disk available at `/mnt1`:

```
df -h
```

This will show output like the following:

```
Filesystem Size Used Avail Use% Mounted on
udev       238M 0    238M  0%   /dev
tmpfs      49M  1.8M 48M   4%   /run
/dev/vda1  2.1G 843M 1.3G  41%  /
tmpfs      245M 0    245M  0%   /dev/shm
tmpfs      5.0M 0    5.0M  0%   /run/lock
tmpfs      245M 0    245M  0%   /sys/fs/cgroup
tmpfs      49M  0    49M   0%   /run/user/1000
/dev/vdb    9.8G 23M  9.2G   1%   /mnt1
```

How it works...

Attaching a *new* Cinder volume is very much like plugging in an unformatted USB stick into your own computer. When it needs to first be used, it must be formatted for use.

On subsequent uses of this disk (when connecting to other instances in the future), you wouldn't need to perform this formatting step.

Under the `openstack` client, the `server add volume` option takes the following syntax:

```
openstack server add volume
    instance_ID
    volume_ID
    --device /dev/device_ID
```

`instance_ID` is the ID returned from `openstack server list` for the instance that we want to attach the volume to.

`volume_ID` is the ID of the volume returned from `openstack volume list`.

`device_ID` is the device that will be created on our instance that we use to mount the volume. Remember that this parameter can sometimes be ignored; so, see this as a hint to pass to a running instance only.

Detaching volumes from an instance

Ordinarily, Cinder volumes can only be attached to one instance at a time. Thus, you need to detach it from one instance before attaching it to another. To detach a volume, we will use another OpenStack client command called `openstack server remove volume`.

Getting ready

To detach a volume from an instance, you will need the following:

- The `openstack` command-line client
- An `openrc` file with appropriate credentials for the environment
- The name or ID of the *volume* to detach
- The name or ID of the *instance* to detach the volume from

For our example, these values are as follows:

- Volume: `cookbook.volume`
- Instance: `cookbook.test`
- Mount point: `/mnt1`

How to do it...

Carry out the following steps to detach a volume to an instance using the `openstack` client:

1. List running instances to get the ID of our instance:

```
+--------------------------------------+----------------+
| ID                                   | Name           |
+--------------------------------------+----------------+
| 90654098-477f-417f-b102-453c0f1f9119 | cookbook.test  |
+--------------------------------------+----------------+
```

2. List the volumes that are available and `in-use` in our environment:

   ```
   openstack volume list
   ```

   This will bring back an output like the following. Note that the information provided shows you if a volume is in use and what instance it is attached to:

```
+--------------------------------------+----------------+--------+------+---------------------------------------+
| ID                                   | Name           | Status | Size | Attached to                           |
+--------------------------------------+----------------+--------+------+---------------------------------------+
| daeddd6c-908a-4ea8-8632-d93d198c2621 | cookbook.volume | in-use |   10 | Attached to cookbook.test on /dev/vdb |
+--------------------------------------+----------------+--------+------+---------------------------------------+
```

3. We now need to perform actions inside the running instance. Connect to this instance and verify that the volume is mounted:

   ```
   df -h
   ```

   This will show an output like the following:

```
Filesystem Size Used Avail Use% Mounted on
udev       238M 0    238M  0%   /dev
tmpfs      49M  1.8M 48M   4%   /run
/dev/vda1  2.1G 843M 1.3G  41%  /
tmpfs      245M 0    245M  0%   /dev/shm
tmpfs      5.0M 0    5.0M  0%   /run/lock
tmpfs      245M 0    245M  0%   /sys/fs/cgroup
tmpfs      49M  0    49M   0%   /run/user/1000
/dev/vdb    9.8G 23M  9.2G   1%   /mnt1
```

4. Now unmount the `/mnt1` volume:

   ```
   sudo umount /mnt1
   ```

   (Verify that it has been unmounted by running `df -h` again).

5. Exit the guest, and back on our OpenStack client, detach the volume from the instance with the following command:

   ```
   openstack server remove volume cookbook.test cookbook.volume
   ```

6. This command doesn't produce any output on success, so view the volume list again to verify that the volume has been detached from the `cookbook.test` instance:

   ```
   openstack volume list
   ```

   This shows the volume is available and not attached to any instance:

```
+--------------------------------------+-----------------+-----------+------+-------------+
| ID                                   | Name            | Status    | Size | Attached to |
+--------------------------------------+-----------------+-----------+------+-------------+
| daeddd6c-908a-4ea8-8632-d93d198c2621 | cookbook.volume | available |   10 |             |
+--------------------------------------+-----------------+-----------+------+-------------+
```

How it works...

Detaching a Cinder volume from an instance is similar to the steps you would take when removing a USB stick from a computer. First, we need to unmount it from the instance so the operating system doesn't complain about an unexpected removal of some storage. Next under the `openstack` client, the `server remove volume` option takes the following syntax:

```
openstack server remove volume instance_name_or_id volume_name_or_id
```

Deleting volumes

At some point, you will no longer need the volumes you have created. To remove the volumes from the system permanently, so they are no longer available, we simply pull out another tool from the OpenStack client, the `volume delete` option.

Getting ready

To delete a volume, you will need the following:

- The `openstack` command-line client
- An `openrc` file with appropriate credentials for the environment
- The name or ID of the *volume* to delete

For our example, these values are as follows:

- Volume: `cookbook.volume`

Warning: This will remove the volume and any data stored on it, so ensure that this is the correct action you want to perform before continuing.

You can only delete a volume that isn't currently attached to an instance.

How to do it...

To delete a volume using the OpenStack client, carry out the following steps:

1. First, we list the volumes available to identify the volume we want to delete, with the following command:

   ```
   openstack volume list
   ```

   This shows that the volume is available and not attached to any instance:

```
+--------------------------------------+-----------------+-----------+------+-------------+
| ID                                   | Name            | Status    | Size | Attached to |
+--------------------------------------+-----------------+-----------+------+-------------+
| daeddd6c-908a-4ea8-8632-d93d198c2621 | cookbook.volume | available |   10 |             |
+--------------------------------------+-----------------+-----------+------+-------------+
```

2. We will now use the volume name or ID to delete this from the system, with the following command:

   ```
   openstack volume delete cookbook.volume
   ```

This command produces no output when successful.

As with attaching and detaching volumes, an ID or name can be used. Best practice is to use the ID to avoid any discrepancies and delete the wrong volume.

How it works...

How the actual volume is deleted depends largely on the Cinder volume driver. In the *Configuring Cinder volume services* recipe of this chapter, we used `cinder.volume.drivers.lvm.LVMVolumeDriver`. In this case, deleting images removes the LVM volume from use within our system.

There's more...

OpenStack Cinder volumes can be snapshotted, in which case, the `openstack volume delete` command will produce an error message like the following:

```
Invalid volume: Volume status must be available or error or error_
restoring or error_extending or error_managing and must not be migrating,
attached, belong to a group, have snapshots or be disassociated from
snapshots after volume transfer. (HTTP 400) (Request-ID: req-2b82e7fc-
0cb4-403f-8dd8-35d99a0f6c6c)
```

To delete a volume, and all of its snapshots, pass the `--purge` flag to the `openstack volume delete` command.

Be careful with this. It is a one-way operation, and `openstack volume delete` does not prompt for confirmation. Additionally, the command produces no output when successful.

Working with volume snapshots

Cinder volume **snapshots** provide a way to nondisruptively copy a volume; allowing for in-situ volume backups to be taken. It also enables more advanced backup features and provides the ability to boot an instance from a given snapshot or from a point in time.

In this section, we will show you how to create a snapshot, mount a volume based off a snapshot, refresh a snapshot, and delete a given snapshot.

Getting ready

To work with Cinder volume snapshots, you will need the following:

- The `openstack` command-line client
- An `openrc` file with appropriate credentials for the environment
- The name or ID of the *volume* to delete

How to do it...

To create a snapshot, the volume must be first detached from an instance:

1. First, list your current volumes:

```
+--------------------------------------+-----------------+-----------+
| ID                                   | Name            | Status    |
+--------------------------------------+-----------------+-----------+
| daeddd6c-908a-4ea8-8632-d93d198c2621 | cookbook.volume | available |
+--------------------------------------+-----------------+-----------+
```

If the volume you wish to snapshot has a status of `in-use`, you will need to detach it using the instructions in the *Detaching volumes from an instance* recipe earlier in this chapter.

2. As our volume is the correct state of `available`, we will proceed to create a snapshot of the volume using the `openstack volume snapshot create` command:

```
openstack volume snapshot create
    --volume cookbook.volume
    cookbook.snapshot
```

This will produce an output like the following, showing a state of `creating`:

```
+-------------+--------------------------------------+
| Field       | Value                                |
+-------------+--------------------------------------+
| created_at  | 2018-01-02T15:57:37.061775           |
| description | None                                 |
| id          | 8acfb2bd-1911-4885-9490-179959a60d51 |
| name        | cookbook.snapshot                    |
| properties  |                                      |
| size        | 10                                   |
| status      | creating                             |
| updated_at  | None                                 |
| volume_id   | daeddd6c-908a-4ea8-8632-d93d198c2621 |
+-------------+--------------------------------------+
```

3. Once the snapshot is complete, you can reattach the original volume using the *Attaching volumes to an instance* recipe, in this chapter and continue operations.
4. If using snapshots as part of an ongoing test / validation process, or part of a backup scheme, you may want to update the snapshot with fresh data. To do this, we use the `cinder snapshot-reset-state`, which produces no output if successful:

```
openstack volume snapshot list -c ID -c Name -c Status -f table
cinder snapshot-reset-state cookbook.snapshot
```

Note the use of the `cinder` command-line tool, as opposed to `openstack`.

5. You are unable to use a snapshot directly; to use a snapshot as a volume to attach to an instance, you first need to create a new volume based off this snapshot. To do this, carry out the following:

```
openstack volume create
    --snapshot cookbook.snapshot
    newcookbook.volume
```

This will produce the following output. Note `snapshot_id` that was used and the fact that we didn't specify a size:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| consistencygroup_id | None                                 |
| created_at          | 2018-01-02T16:09:09.528566           |
| description         | None                                 |
| encrypted           | False                                |
| id                  | dba5520f-1e0b-4368-821b-03639a7f0725 |
| migration_status    | None                                 |
| multiattach         | False                                |
| name                | newcookbook.volume                   |
| properties          |                                      |
| replication_status  | None                                 |
| size                | 1                                    |
| snapshot_id         | c74a9ba9-10a1-4567-b744-381c3ac1f4cd |
| source_volid        | None                                 |
| status              | creating                             |
| type                | None                                 |
| updated_at          | None                                 |
| user_id             | b01e337fd00f42b195ee8287eb6d8334     |
+---------------------+--------------------------------------+
```

6. We can confirm that our new cookbook volume, based off the snapshot, is now available by viewing the volume list again:

```
openstack volume list -c Name -c ID -c Status -f table
```

This produces a list of our volumes and their state:

```
+--------------------------------------+--------------------+-----------+
| ID                                   | Name               | Status    |
+--------------------------------------+--------------------+-----------+
| dba5520f-1e0b-4368-821b-03639a7f0725 | newcookbook.volume | available |
| 47ad8896-22fa-4120-ae71-940db069346d | cookbook.volume    | available |
+--------------------------------------+--------------------+-----------+
```

7. Finally, you will want to delete snapshots at some point. To do this, use the `openstack volume snapshot delete` command as follows:

```
openstack volume snapshot delete cookbook.snapshot
```

   Confirm with `openstack volume snapshot list` the list the remaining snapshots available.

How it works...

Cinder volume snapshots provide a flexible way to clone volumes for snapshot type backups, attaching to other instances, and more. The `cinder snapshot` commands we used here, specifically `openstack volume snapshot create`, `openstack volume snapshot list`, `cinder snapshot-reset-state`, and `openstack volume snapshot delete`, instruct `cinder` to work with the storage driver to perform snapshot specific actions—create, list, update, and delete, respectively. The specific implementation of snapshot depends on the underlying driver.

Also note that you cannot use a snapshot directly with an instance. You must first create a new volume based on the snapshot of your choice. This has the following syntax:

```
openstack volume create
    --snapshot snapshot_name_or_id
    new_volume_name
```

Configuring volume types

Volume types in Cinder are a label or identifier that is selected during volume creation. Typically, volume types correspond with some attribute of the volumes, for example, `SSD`, `High IOPS`, and `Encrypted`. We will use this with the next recipe to define further features of our Cinder service.

Getting ready

To create a volume type, you will need the following:

- An `openrc` file with appropriate credentials for the environment (you must be an administrator)
- The `openstack` command-line client
- The name of the volume type to create. For our example, we will create the `"High IOPS"` volume type as a contrived example type that would refer to a block storage device dedicated to High IOPS.

How to do it...

We will use the `openstack volume type` command to operate on Cinder volume types. To create a volume type, use the following steps:

1. First, list the existing volume types:

   ```
   openstack volume type list
   ```

   This will bring back an output like the following. Here we have the default set when we installed our LVM service:

   ```
   +--------------------------------------+------+-----------+
   | ID                                   | Name | Is Public |
   +--------------------------------------+------+-----------+
   | 27aaadd0-907c-4db9-8314-ffcadf840d4c | lvm  | True      |
   +--------------------------------------+------+-----------+
   ```

2. Create the new volume type:

   ```
   openstack volume type create
       --description "The High IOPS volume type is QOS applied to 500
   IOPS"
       "High IOPS"
   ```

   This will bring back an output like the following:

   ```
   +-------------+-------------------------------------------------------+
   | Field       | Value                                                 |
   +-------------+-------------------------------------------------------+
   | description | The High IOPS volume type is QOS applied to 500 IOPS  |
   | id          | 419d30c5-e030-48a0-a232-38ebb4c5f2d1                  |
   | is_public   | True                                                  |
   | name        | High IOPS                                             |
   +-------------+-------------------------------------------------------+
   ```

3. Confirm that the new volume type is available by displaying the list of volume types:

   ```
   openstack volume type list
   ```

   This will now bring back our additional volume type:

   ```
   +--------------------------------------+-----------+-----------+
   | ID                                   | Name      | Is Public |
   +--------------------------------------+-----------+-----------+
   | 419d30c5-e030-48a0-a232-38ebb4c5f2d1 | High IOPS | True      |
   | 27aaadd0-907c-4db9-8314-ffcadf840d4c | lvm       | True      |
   +--------------------------------------+-----------+-----------+
   ```

4. To use this new type when creating a volume, we will use the `--type` flag as follows:

```
openstack volume create
    --size 10
    --type "High IOPS"
    --description "High IOPS Volume"
    highiops.volume
```

5. We will verify the type in the output:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| consistencygroup_id | None                                 |
| created_at          | 2018-01-02T16:42:10.820348           |
| description         | High IOPS Volume                     |
| encrypted           | False                                |
| id                  | 978780e6-49ca-4a6a-b242-cdc4a181a4d3 |
| migration_status    | None                                 |
| multiattach         | False                                |
| name                | highiops.volume                      |
| properties          |                                      |
| replication_status  | None                                 |
| size                | 1                                    |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| type                | High IOPS                            |
| updated_at          | None                                 |
| user_id             | b01e337fd00f42b195ee8287eb6d8334     |
+---------------------+--------------------------------------+
```

How it works...

The `openstack volume type create` command has only one mandatory parameter, `name`. The `name` parameter allows users to define different volume types or identifiers. While typically based on some attribute of the storage, such as department, storage backend, or QOS level, as these are labels, as long as they're well understood, the name can be arbitrary.

Additionally, the `--description` flag can be used to provide more detail on the volume type. The `--private` flag allows you to restrict visibility from the public. The `--project` flag allows the selective sharing of a private volume type with other named projects.

We will then take advantage of these types (with further configuration of Cinder to specify that a type of storage, which is associated with a particular volume type) by issuing the `--type` flag to the `volume create` command:

```
openstack volume create
    --size size_GiB
    --description "meaningful description"
    --type "Type"
    volume_name
```

Enabling volume encryption

Cinder can manage the **encryption** of volumes, and it happens transparent to the guest. Encryption is enabled on a *volume type* level.

Getting ready

Encryption can be enabled either when creating a new volume type or added to an existing volume type that has no volumes in use. To enable volume encryption, you will need the following:

- An `openrc` file with appropriate credentials for the environment
- The `openstack` command-line client
- The name of the *volume type*
- Name of the *encryption provider*
- Encryption control location
- Encryption key size
- Encryption cipher

For our example, these will be as follows:

- Name: `Cookbook Encrypted Volumes`
- Encryption provider: `nova.volume.encryptors.luks.LuksEncryptor`
- Encryption control location: `front-end`
- Encryption key size: `256`
- Encryption cipher: `aes-xts-plain64`

The encryption-specific values you choose will be based on what is available in your particular environment. A detailed discussion of these values is beyond the scope of this book.

How to do it...

To enable volume encryption as a new volume type, the following command is used:

```
openstack volume type create
    --description "LUKS Encrypted volumes"
    --encryption-provider nova.volume.encryptors.luks.LuksEncryptor
    --encryption-control-location front-end
    --encryption-key-size 256
    --encryption-cipher aes-xts-plain64
    "Encrypted"
```

We would then use this `"Encrypted"` volume type when creating a volume as follows:

```
openstack volume create
    --size 1
    --type "Encrypted"
    --description "An encrypted volume"
    encrypted.volume
```

How it works...

Volumes are configured as a volume type, and thus, additional parameters are passed to `openstack volume type create`:

- The `--encryption-provider` flag let's Cinder know which provider will perform the encryption. As with storage backends, there are a number of providers available. Refer to the OpenStack documentation for a current list.
- The `--encryption-control-location` parameter tells Cinder where the encryption will be handled. In our case, `front-end` means that Nova will be handling the encryption.
- Next, the `--encryption-key-size` parameter specifies the size of the key used. `256` was selected for the example as to not crush lab performance. The encryption provider and cipher you choose will provide specific recommendations.
- Finally, `--encryption-cipher` specifies which cipher to use. You can use `cryptsetup benchmark` to get a list of available options and an idea as to how they will perform.

Configuring volume Quality of Service (QoS)

Another feature provided by Cinder is the ability to define and manage classes of service for volumes. Like volume encryption, **Quality of Service** (**QoS**) in Cinder is configured by volume type. By default, you can define values for minimum, maximum, and burst IOPS.

Getting ready

To configure volume QoS, you will need the following information:

- An `openrc` file with appropriate credentials for the environment (you need to be an administrator)
- The `openstack` command-line client
- The name or ID of the *volume type*
- The *consumer* of the QoS policy
- Values for the following:
  - Minimum IOPS
  - Maximum IOPS
  - Burst IOPS

Only one of the three (min, max, and burst) needs to be provided to define a minimal QoS policy.

For our example, these will be as follows:

- Name: `High IOPS`
- Consumer: `both`
- Maximum IOPS: `500`

How to do it...

For Cinder volumes to use QoS, an administrator needs to perform two steps: first, define the specification, and second, to associate the specification with a volume type.

To create and assign a QoS specification, use the following steps:

1. First, list the existing QoS specifications:

   ```
   openstack volume qos list --print-empty
   ```

   We currently don't have any QoS defined:

   ```
   +----+------+----------+--------------+------------+
   | ID | Name | Consumer | Associations | Properties |
   +----+------+----------+--------------+------------+
   +----+------+----------+--------------+------------+
   ```

2. Define a new QoS specification with the following:

   ```
   openstack volume qos create
       --consumer both
       --property maxIOPS=500
       "High IOPS"
   ```

   This will bring back the following output:

   ```
   +------------+--------------------------------------+
   | Field      | Value                                |
   +------------+--------------------------------------+
   | consumer   | both                                 |
   | id         | 6c19d357-71fb-4be7-bede-2590b2512067 |
   | name       | High IOPS                            |
   | properties | maxIOPS='500'                        |
   +------------+--------------------------------------+
   ```

3. Confirm our QoS specification was created by listing the QoS available:

   ```
   openstack volume qos list -c Name -c Consumer -c Properties -f
   table
   ```

   This will bring back the following output:

   ```
   +-----------+----------+---------------+
   | Name      | Consumer | Properties    |
   +-----------+----------+---------------+
   | High IOPS | both     | maxIOPS='500' |
   +-----------+----------+---------------+
   ```

4. We now associate the policy to a volume type as follows:

   ```
   openstack volume qos associate "High IOPS" "High IOPS"
   ```

This command produces no output when successful.

How it works...

QoS specifications are defined within Cinder using the `openstack volume qos` set of commands. When creating a QoS specification, you can specify where QoS is applied as well as the value to set it at. Currently, you can specify a static set of minimum, maximum, and burst IOPS, and a scaling set of IOPS.

The static values break down as follows:

- **Minimum IOPS**: This is the number of IOPs *guaranteed* to a volume
- **Maximum**: This is the ceiling for IOPS on a volume
- **Burst IOPS**: This is the maximum IOPS over a short period

Scaling IOPs then, define the amount to change the static values for each additional gigabyte of volume size.

Resetting volume state

During the ongoing operation of your OpenStack cloud, you will occasionally encounter trouble where a Cinder volume will get stuck in an odd state. During the course of writing this chapter, the authors had a number of volumes getting stuck in the `attaching` status, after an instance failed to boot from them. This looks like the following:

```
openstack volume list -c Name -c Status -f table
```

```
+---------------------------+-----------+
| Name                      | Status    |
+---------------------------+-----------+
| cookbook.volume.boot.test | attaching |
| cookbook.boot.volume      | attaching |
| cookbook.test             | available |
+---------------------------+-----------+
```

The cause of this was some fat-fingered typing while creating the boot from volume section!

Getting ready

To reset the status on a Cinder volume, you will need the following:

- An `openrc` file with appropriate credentials for the environment
- The `openstack` command-line client
- The `cinder` command-line client
- The *name* or *ID* of the volume

For the example that follows, we will be resetting the following volumes to available:

- `cookbook.boot.volume`
- `cookbook.volume.boot.test`

How to do it...

Resetting the status of a Cinder volume is done with the `cinder` command. Here the `openstack` set of commands covers most operations you will commonly need to do, and the `cinder` command provides additional admin functionality, such as `reset-state`.

As the `reset-state` command only manipulates the database without regard to the actual status, it should be used with care.

To reset the status of a Cinder volume, carry out the following steps:

1. First, list your Cinder volumes and statuses

   ```
   openstack volume list -c Name -c Status -f table
   ```

   This will bring back a list of the volumes OpenStack knows about:

```
+---------------------------+-----------+
| Name                      | Status    |
+---------------------------+-----------+
| cookbook.volume.boot.test | attaching |
| cookbook.boot.volume      | attaching |
| cookbook.test             | available |
+---------------------------+-----------+
```

2. Use the `cinder` client to reset the state of the volumes:

```
cinder reset-state 23e1e006-a753-403c-ad8f-27e98444f71e --state
available
cinder reset-state e934c45f-6e2f-431f-8457-7e84f6cee876 --state
available
```

When successful, this command produces no output.

3. Confirm the new status:

```
openstack volume list -c Name -c Status -f table
```

This will show that the state has been reset to `available`:

```
+--------------------------+-----------+
| Name                     | Status    |
+--------------------------+-----------+
| cookbook.volume.boot.test | available |
| cookbook.boot.volume      | available |
| cookbook.test             | available |
+--------------------------+-----------+
```

How it works...

The `cinder reset-state` command operates directly on the `cinder` database, regardless of the actual status of the volume. The `--state` flag we used allows us to change the status of a volume that might be stuck in a particular state. There are two additional flags that allow you to change the attachment status and migration status, respectively:

```
cinder help reset-state
usage: cinder reset-state
    [--type ]
    [--state ]
    [--attach-status ]
    [--reset-migration-status]
    [ ...]
```

This tool explicitly updates the entity state in the `cinder` database. Being a database change only, this has no impact on the true state of the entity and may not match the actual state. This can render an entity unusable in the case of changing to the `available` state.

8
Swift – OpenStack Object Storage

In this chapter, we will cover the following topics:

- Introduction – OpenStack Object Storage
- Creating object containers
- Deleting object containers
- Uploading objects
- Uploading large objects
- Downloading objects
- Deleting objects
- Container ACLs

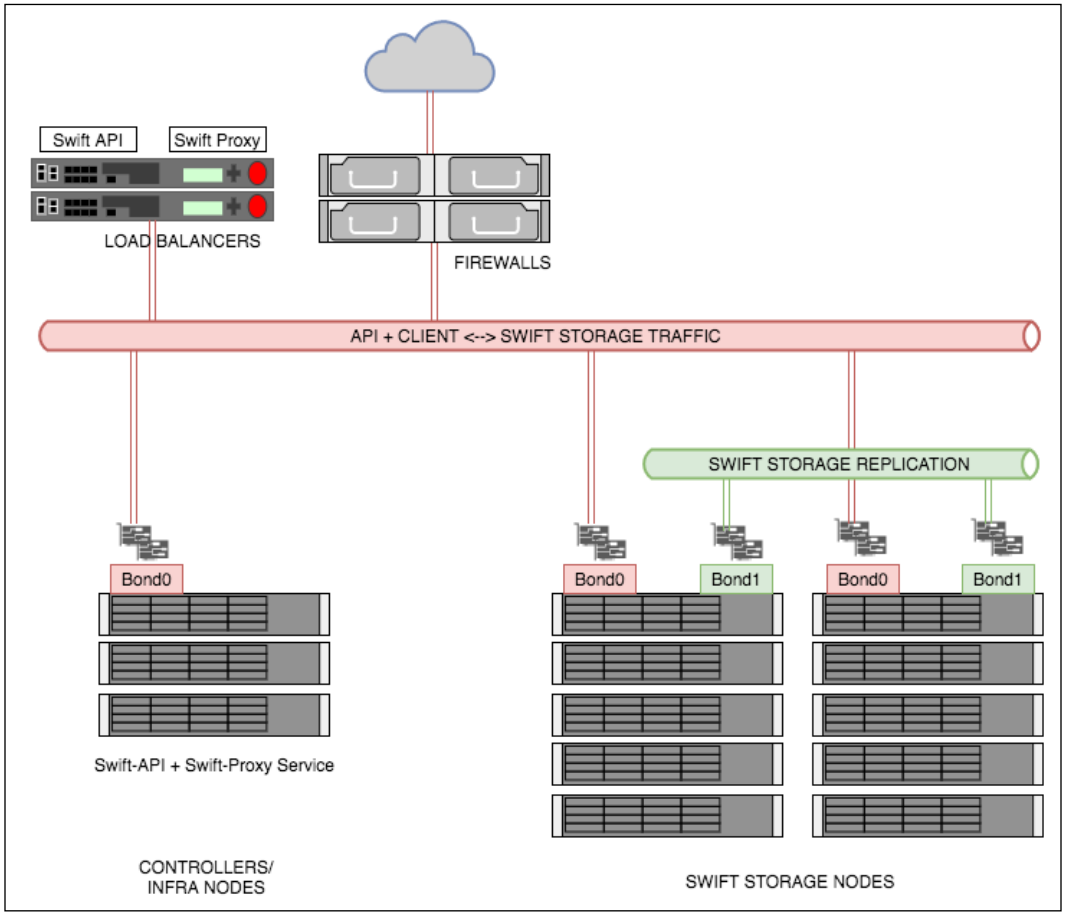
Introduction – OpenStack Object Storage

OpenStack Object Storage, also known as **Swift**, is the service that allows for massively scalable and highly redundant storage on commodity hardware. This service is implemented by Rackspace as Cloud Files, and it is also analogous to Amazon's S3 storage service and managed in a similar way under OpenStack. With OpenStack Object Storage, we can store many objects of virtually unlimited size—restricted only by the available hardware—and grow our environment as needed, to accommodate our storage. The highly redundant nature of OpenStack Object Storage is ideal for archiving data (such as logs and backup archives) as well as for providing a storage system that OpenStack Compute can use for virtual machine instance images.

The architecture of OpenStack Object Storage is straightforward. There is the API service that runs on the controller nodes. Then there are Swift proxy services that can either be deployed onto the controller nodes, or separated onto their own dedicated servers—depending on your requirements, followed by the actual storage nodes that store the data. The storage nodes are the servers that are designed to scale out as your requirements for storage grows.

Scaling the characteristics and design of Swift is beyond the scope of this book.

A typical, simplified view of the architecture is show here:



Essentially, requests (for example, to upload or download an object) are sent to the API through the load balance pool, and then the data is sent through to the physical storage nodes by the `Swift-Proxy` service. The `Swift-Proxy` service would fetch and store the data back to and from the end user.

As the name states, OpenStack Object Storage operates on what is known as objects. Objects can be anything from a file, to an object named as a complete folder and filename – to OpenStack Object Storage, anything it sees, regardless of the filename, is still one single object. Objects are stored in **Containers**. A great analogy is a bucket. In a bucket, you can store anything that would fit in it, from grains of sand to tools you have laying in your shed!

Creating object containers

To get started using OpenStack Object Storage, we must first create a **container**. A container in this case is quite similar to a folder on Windows or Linux file directory. However, containers cannot be nested, though deep structures can be created in a fashion similar to the nested folder structure using both container and object names (pseudo folders) when we come to uploading the objects that are stored in these containers. Names we assigned containers and objects are analogous to labels that allow us to interpret as folder structures through the use of a `/` character in these labels.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client as described in *Chapter 2*, *The OpenStack Client*, and can access the OpenStack environment as a user with the `swiftoperator` privileges.

We will use the `developer` user created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client*, with the `cookbook4` password; we have also granted this user with the `swiftoperator` privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

To create a Swift `container` on our environment, follow these steps:

1. To create a container, execute the following command:

```
openstack container create books
```

2. Once the container is created, you will see the following output:

```
+--------------------------------------+-----------+----------------------------------+
| account                              | container | x-trans-id                       |
+--------------------------------------+-----------+----------------------------------+
| AUTH_402e8fe274c143ea91fe905a1b8c7614 | books     | tx4fed0bc15e47450b82835-005a238985 |
+--------------------------------------+-----------+----------------------------------+
```

3. To view all the available containers, type this command:

   `openstack container list`

   This will give an output like the following:

```
+-------+
| Name  |
+-------+
| books |
+-------+
```

4. Once the container is created, we can set additional details or **metadata** on it by executing the following command:

   `openstack container set books --property title=cookbooks`

5. To view container details, type the following command:

   `openstack container show books`

   This will give the following output:

```
+--------------+----------------------------------------+
| Field        | Value                                  |
+--------------+----------------------------------------+
| account      | AUTH_402e8fe274c143ea91fe905a1b8c7614 |
| bytes_used   | 0                                      |
| container    | books                                  |
| object_count | 0                                      |
| properties   | Title='cookbooks'                      |
+--------------+----------------------------------------+
```

6. As described in the introduction, names that we assigned to containers and objects are very much like labels; so, to create a pseudo-folder in the container, use `/` delimiter in the container name:

   `openstack container create books/chapter1`

This will return output like the following:

```
+---------------------------------------+---------------+-------------------------------------+
| account                               | container     | x-trans-id                          |
+---------------------------------------+---------------+-------------------------------------+
| AUTH_402e8fe274c143ea91fe905a1b8c7614 | books/chapter1 | txa712d8af947d49fab6337-005a3566fd |
+---------------------------------------+---------------+-------------------------------------+
```

7. To view details of the pseudo-folder, execute the `container show` command by including the full pseudo-folder name:

   `openstack container show books/chapter1`

   This will give the information like the following:

```
+--------------+---------------------------------------+
| Field        | Value                                 |
+--------------+---------------------------------------+
| account      | AUTH_402e8fe274c143ea91fe905a1b8c7614 |
| bytes_used   | None                                  |
| container    | books/chapter1                        |
| object_count | None                                  |
+--------------+---------------------------------------+
```

How it works...

In OpenStack Object Storage, users with the `admin` or `swiftoperator` privileges can utilize the Object Storage service. To do so, we first must create **containers**, where the objects will be stored. Containers may not be nested, though we may create pseudo-folders using the `/` delimiter in the container name. To create a container, follow this command-line syntax:

```
openstack container create container_name
```

To list available containers, use the following command:

```
openstack container list
```

We can also set metadata on each container. Use the following command for setting additional info on a container:

```
openstack container set container_name
    --property key=value
```

Multiple pairs of metadata may be set on each container.

To view container details, execute this command:

```
openstack container show <container_name>
```

Deleting object containers

Deleting OpenStack Object Storage containers is quite simple. Any container can be deleted, for as long as they are empty.

In this case, the OpenStack CLI has different behavior from the Swift CLI tool. The Swift CLI will *delete container with all of its* contents, while the OpenStack CLI will not. In this example, we are using the OpenStack CLI.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with the `swiftoperator` privileges. We will use the `developer` user created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client* with the `cookbook4` password. We have also granted this user the `swiftoperator` privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

To delete a Swift container in our environment, execute the following steps:

1. First, list the available containers:

   ```
   openstack container list
   ```

   This will give an output like the following:

   ```
   +--------+
   | Name   |
   +--------+
   | books  |
   +--------+
   ```

2. View container details to make sure that it is empty:

   ```
   openstack container show books
   ```

This will show an output like the following:

```
+--------------+----------------------------------------+
| Field        | Value                                  |
+--------------+----------------------------------------+
| account      | AUTH_402e8fe274c143ea91fe905a1b8c7614  |
| bytes_used   | 0                                      |
| container    | books                                  |
| object_count | 0                                      |
| properties   | Title='cookbooks'                      |
+--------------+----------------------------------------+
```

3. Now delete the container using the following command:

```
openstack container delete books
```

There is no output from this command.

How it works...

In OpenStack Object Storage, empty containers can be deleted by a user that created them. First, we must make sure that the container is empty by viewing its details:

```
openstack container show container_name
```

Then, delete the container with the following command:

```
openstack container delete container_name
```

Verify that container is gone by listing available containers:

```
openstack container list
```

Uploading objects

Once we have created one or more containers, we can start uploading objects to them. While OpenStack Object Storage does not support nested containers, we can simulate folders or file directories with object names. This presents similar structure to the pseudo-folders we used in container names and both achieve similar end goals to a user expecting a tree-like structure to their object storage use.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with the `swiftoperator` privileges. We will use the `developer` user created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client*, with the `cookbook4` password; we have also granted this user `swiftoperator` privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack command-line client.

How to do it...

To upload objects to an OpenStack Object Storage container, follow the following steps:

1. We will upload the `intro.txt` file into the `books` container:

   ```
   openstack object create books intro.txt
   ```

   This will give the following output:

   ```
   +-----------+-----------+----------------------------------+
   | object    | container | etag                             |
   +-----------+-----------+----------------------------------+
   | intro.txt | books     | 0ed4362d6b074cccabc41cff7c2cb283 |
   +-----------+-----------+----------------------------------+
   ```

2. List objects in a container by providing the container name:

   ```
   openstack object list books
   ```

   This will give an output like the following:

   ```
   +-----------+
   | Name      |
   +-----------+
   | chapter1  |
   | intro.txt |
   +-----------+
   ```

Here `chapter1` is the *pseudo-folder* we created in the *Creating object containers* recipe and it shows as an object.

3. We can upload an object into the pseudo-folder called `books/chapter1` as follows:

   `openstack object create books/chapter1 swift.txt`

   This will give an output like the following:

```
+-----------+----------------+----------------------------------+
| object    | container      | etag                             |
+-----------+----------------+----------------------------------+
| swift.txt | books/chapter1 | 922ce68107fcfb7744dc2e2aee589ee6 |
+-----------+----------------+----------------------------------+
```

4. To list objects in a container, issue the following against the container name:

   `openstack object list books`

   This will list the objects available under this container:

```
+--------------------+
| Name               |
+--------------------+
| chapter1           |
| chapter1/swift.txt |
| intro.txt          |
+--------------------+
```

5. To list all objects in the pseudo-folder container name, use a `prefix` flag:

   `openstack object list books --prefix chapter1/`

```
+--------------------+
| Name               |
+--------------------+
| chapter1/swift.txt |
+--------------------+
```

You can use `--prefix` to list any object that begins with the prefixed letters.

6. To list all top-level objects in a container, use a **delimiter** flag. In this example, the delimiter is `/`:

   `openstack object list books --delimiter /`

```
+-----------+
| Name      |
+-----------+
| chapter1  |
| intro.txt |
+-----------+
```

7. To show information about an object, issue the following command:

```
openstack object show books chapter1/swift.txt
```

This will give information about that object:

```
+----------------+------------------------------------------+
| Field          | Value                                    |
+----------------+------------------------------------------+
| account        | AUTH_402e8fe274c143ea91fe905a1b8c7614    |
| container      | books                                    |
| content-length | 32757                                    |
| content-type   | text/plain                               |
| etag           | 922ce68107fcfb7744dc2e2aee589ee6         |
| last-modified  | Sat, 16 Dec 2017 21:44:07 GMT            |
| object         | chapter1/swift.txt                       |
+----------------+------------------------------------------+
```

How it works...

In order to store files in OpenStack Object Storage, **objects** have to be uploaded to a **container**. Containers cannot be nested; however, pseudo-folders or pseudo-directories can be created using the `/` delimiter. Once a container is created, you can upload files by using the following command:

```
openstack object create container_name object_1 [object_2 …]
```

Multiple objects can be uploaded with one command.

To list objects that were uploaded to a container, execute the following command:

```
openstack object list container_name
```

When listing objects in a container, enable filtering using the `--prefix` and `--delimiter` flags.

By default, a maximum of 10,000 objects will be listed. Use pagination or the `--all` flag to view more than the default number of objects.

To view details of individual object, use the following command:

```
openstack object show container_name object_name
```

Regular object size is limited in Swift. By default, only objects 5 GB or smaller can be uploaded with the `openstack object create` command. Refer to the *Uploading large objects* recipe on how to upload large objects.

Uploading large objects

Every OpenStack Object Storage cluster has a limit on how large the uploaded objects can be. Usually that limit is set to 5 GB, though each cluster can have its own limit. However, this doesn't mean that you are limited to uploading only 5 GB or smaller objects to OpenStack Object Storage. Swift provides large object support via already configured and deployed middleware by splitting up large objects into smaller parts. There are two types of large object support: dynamic and static.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with the `swiftoperator` privileges. We will use the `developer` user created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client* with the `cookbook4` password. We also granted this user with the `swiftoperator` privileges.

Since the OpenStack CLI does not provide all the functionality required through the individual OpenStack project client, we will need to use the Swift CLI for this recipe. Ensure that you have the Swift command-line client installed. If you do not, install it:

```
pip install python-swiftclient
```

How to do it..

To upload large objects to OpenStack Object Storage container, follow these steps:

1. First, we can list available containers with the following command:

   ```
   swift list
   ```

   This will give a list of the containers we have:

   ```
   books
   ```

2. To upload a large file, we will specify the *segment size* of the objects in *bytes* with the `-S` flag:

   ```
   swift upload -S 25000 books nova.txt
   ```

This will show the object being split into **segments** in the following output:

```
nova.txt segment 1
nova.txt segment 0
nova.txt segment 5
nova.txt segment 2
nova.txt segment 3
nova.txt segment 4
nova.txt
```

In this example, the segment size is 25000 bytes, and our 128 K `nova.txt` file was split into 6 segments before being uploaded to the Swift cluster.

3. Verify that the file has been uploaded by listing the available objects in the container:

```
swift list books
```

This shows our `nova.txt` file available:

```
nova.txt
```

Even though the file was split into 6 parts for uploading, it appears as one file in Swift's `books` container. The segments are listed separately in a new container.

4. To view the available containers, issue this command:

```
swift list
```

This gives us our containers with a new one automatically created:

```
books
books_segments
```

You will notice that a new `books_segments` container has been created automatically.

5. List objects in the `books_segments` container to view individual objects:

```
swift list books_segments
```

This shows our segmented file objects:

```
nova.txt/1512361523.073429/131024/25000/00000000
nova.txt/1512361523.073429/131024/25000/00000001
nova.txt/1512361523.073429/131024/25000/00000002
nova.txt/1512361523.073429/131024/25000/00000003
nova.txt/1512361523.073429/131024/25000/00000004
nova.txt/1512361523.073429/131024/25000/00000005
```

Do not modify or delete these objects individually.

6. View the details of the uploaded object in the `books` container using the `stat` command:

```
swift stat books nova.txt
```

This gives a number of details about our `nova.txt` file in the `books` container:

```
               Account: AUTH_402e8fe274c143ea91fe905a1b8c7614
             Container: books
                Object: nova.txt
          Content Type: text/plain
        Content Length: 131024
         Last Modified: Sun, 17 Dec 2017 06:36:32 GMT
                  ETag: "cd0ef1b80a6261f0b6b7db9efa739938"
              Manifest: books_segments/nova.txt/
                        1512361523.073429/131024/25000/
           Meta Mtime: 1512361523.073429
         Accept-Ranges: bytes
           X-Timestamp: 1513492591.13264
            X-Trans-Id: tx4fa6e7a4e53a459a86430-005a3611ad
X-Openstack-Request-Id: tx4fa6e7a4e53a459a86430-005a3611ad
```

Take a look at the `manifest` field in the information provided. The `manifest` field will include the `container` details that was created for individual segments, the original file size, and segment size.

How it works

Since the OpenStack CLI does not provide large file upload support at the time of this recipe's writing, we will use the Swift command-line client. In order to store large files in OpenStack Object Storage, objects have to be split up or segmented before uploading to a container. This segmentation is done for us by specifying the `-S` flag in the `swift upload` command:

```
swift upload -S size_in_bytes container large_object
```

When uploading large files, Swift automatically creates a new container and pseudo-folders for individual segments of the uploaded object. Individual segments may be listed the same way as regular objects; however, do not manipulate them directly.

Downloading objects

Once objects have been uploaded to a container, one may also want to download them. In this recipe, we will show you how to download objects to your local disk.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with the `swiftoperator` privileges. We will use the `developer` user created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client* with the `cookbook4` password. We have also granted this user the `swiftoperator` privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use the OpenStack command-line client.

How to do it...

To download objects from a Swift cluster, carry out the following steps:

1. List the available objects in a container:

   ```
   openstack object list books
   ```

   This will give a list of objects in our `books` container:

   ```
   +---------------------+
   | Name                |
   +---------------------+
   | chapter1            |
   | chapter1/swift.txt  |
   | intro.txt           |
   | nova.txt            |
   +---------------------+
   ```

2. To download the desired file, for example, `intro.txt`, issue the following command:

   ```
   openstack object save books intro.txt
   ```

   There is no output to this command. The file will be saved to the current directory.

3. We can also download a file and specify the destination file:

   ```
   openstack object save books nova.txt --file /tmp/nova.txt
   ```

How it works...

In order to download files from the OpenStack Object Storage, issue the following command:

```
openstack object save container object --file file_name
```

The `--file` flag is optional. If not specified, the object will be saved to the current directory.

Large objects do not require special treatment when downloading; the same command will work on large and small objects.

Deleting objects

Deleting objects from OpenStack Object Storage is fairly trivial, whether for small or large files.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with the `swiftoperator` privileges. We will use the `developer` user created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client*, with the `cookbook4` password. We have also granted this user the `swiftoperator` privileges.

Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use the OpenStack command-line client.

How to do it...

To delete objects from our Swift cluster, carry out the following steps:

1. List available objects in a container:

   ```
   openstack object list books
   ```

   This will give a list of objects in our `books` container:

```
+--------------------+
| Name               |
+--------------------+
| chapter1           |
| chapter1/swift.txt |
| intro.txt          |
| nova.txt           |
+--------------------+
```

2. Delete the object from a container:

```
openstack object delete books intro.txt
```

There will be no output from this command.

3. List objects in the `books` container after deletion:

```
openstack object list books
```

The container listing will show that `intro.txt` is no longer present:

```
+--------------------+
| Name               |
+--------------------+
| chapter1           |
| chapter1/swift.txt |
| nova.txt           |
+--------------------+
```

4. To delete *all objects* from a container and *then delete container*, issue the following command:

```
openstack container delete -r books
```

There is no output to this command.

How it works

In order to delete files from the OpenStack Object Storage cluster, specify the container and object that should be deleted with the following command:

```
openstack object delete container object
```

Use the `-r` flag for deleting container and all its contents.

Container ACLs

OpenStack Object Storage containers are usually owned by the user that created them. However, through Swift's **ACLs** (**Access Control Lists**), containers can be made accessible to different OpenStack users or made completely public. The owner of the container can set specific read and write rules. The read and write rules must be set separately and have to be enabled explicitly on each container. The owner of the container can make the container completely public or set rules based on the project, user, or rule set.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment as a user with the `swiftoperator` privileges and an `admin` user. We will use the `developer` user created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client*, with the `cookbook4` password. We have also granted this user the `swiftoperator` privileges.

Since the OpenStack CLI does not provide all the functionality available through the individual OpenStack project client, we will need to use Swift CLI for this recipe. Ensure that you have the Swift command-line client installed. If you do not, install it:

```
pip install python-swiftclient
```

How to do it...

To view and modify ACLs on containers, follow the following steps:

1. First, view existing ACLs on a container, if any:

   ```
   swift stat books
   ```

   This gives the information about our container called `books`:

   ```
                  Account: AUTH_402e8fe274c143ea91fe905a1b8c7614
                Container: books
                  Objects: 3
                    Bytes: 32764
                 Read ACL:
                Write ACL:
                  Sync To:
                 Sync Key:
             Accept-Ranges: bytes
          X-Storage-Policy: default
             Last-Modified: Mon, 18 Dec 2017 06:09:45 GMT
               X-Timestamp: 1512278405.11522
                X-Trans-Id: tx484e741deb754fdb86f7a-005a375e4c
              Content-Type: text/plain; charset=utf-8
   X-Openstack-Request-Id: tx484e741deb754fdb86f7a-005a375e4c
   ```

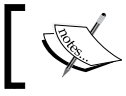
2. In our example, there are no read or write ACLs set yet. Let's set a `read` ACL to make the `books` container *public*:

```
swift post books --read-acl ".r:*,.rlistings"
```

There is no output from this command.

3. To make the `books` container *writable* by everybody, issue the following:

```
swift post books --write-acl "*:*"
```



There is no output from this command.

4. Now check the details on the books container again with the stat command:

```
swift stat books
```

We can see that the `Read ACL` and `Write ACL` fields have been populated:

```
               Account: AUTH_402e8fe274c143ea91fe905a1b8c7614
             Container: books
               Objects: 3
                 Bytes: 32764
              Read ACL: .r:*,.rlistings
             Write ACL: *:*
               Sync To:
              Sync Key:
          Accept-Ranges: bytes
             X-Trans-Id: txc0d0d64ed54e48989f3f6-005a3760ba
      X-Storage-Policy: default
         Last-Modified: Mon, 18 Dec 2017 06:22:56 GMT
           X-Timestamp: 1512278405.11522
          Content-Type: text/plain; charset=utf-8
X-Openstack-Request-Id: txc0d0d64ed54e48989f3f6-005a3760ba
```

5. Since operating *world-writable* and *readable* containers are not very good security practice, we can remove the ACLs from the container. To remove the *read* ACL, issue this command:

```
swift post -r "" books
```

6. To remove the *write* ACL, use this command:

```
swift post -w "" books
```

7. If you need to share your container with another user in your OpenStack environment, you can set permissions based on the *project* and *user*. In our example, we will set the `books` container's access to be readable by everyone in the `admin` project:

```
swift post -r "admin:*" books
```

   The asterisk (`*`) after `:` indicates that all users in the `admin` project will have access to the `books` container.

8. Now check the details of the `books` container:

```
swift stat -v books
```

   This will produce output like the following:

```
                  URL: http://172.29.236.100:8080/v1/
                       AUTH_402e8fe/books
           Auth Token: gAAAAABaODQ8R93x7kW46CW_u9ZS3
              Account: AUTH_402e8fe274c143ea91fe905a1b8c7614
            Container: books
              Objects: 3
                Bytes: 32764
             Read ACL: admin:*
            Write ACL:
              Sync To:
             Sync Key:
        Accept-Ranges: bytes
           X-Trans-Id: tx20b0d0d8394b4b0a81cba-005a38343c
      X-Storage-Policy: default
         Last-Modified: Mon, 18 Dec 2017 21:24:27 GMT
           X-Timestamp: 1512278405.11522
          Content-Type: text/plain; charset=utf-8
X-Openstack-Request-Id: tx20b0d0d8394b4b0a81cba-005a38343c
```

   Notice the `URL` of the container in the details. Anybody wishing to access this container will need to pass the `URL` field as a parameter.

9. As an `admin` user, test the access to the `books` container:

```
swift
    --os-storage-url http://172.29.236.100:8080/v1/AUTH_402e8fe/
books list
```

This will give objects from our shared container at the specified URL:

```
chapter1
chapter1/swift.txt
intro.txt
```

In our example, the `admin` user is part of the `admin` project and therefore is able to access the `books` container via the `--os-storage-url` flag.

How it works...

Containers can be shared with other users by setting *read* and *write* **ACLs** on them. Currently, the ACLs functionality is not available in the OpenStack client, so we are using the Swift CLI in our examples.

There are two types of ACLs that can be set on a container, *read* and *write*, and they have to be set individually.

Set read ACL with the following command:

```
swift post -r "project:user" container
```

Set write ACL as follows:

```
swift post -w "project:user" container
```

Here both the project and user can be substituted with a wild card (`*`).

To make a container completely public, use the following commands:

```
swift post --read-acl ".r:*,.rlistings" container
swift post --write-acl "*:*" container
```

With the `.r:*` and `.rlistings` elements set, the `books` container is publicly accessible. The `.r*` element allows access to the objects in a container, and `.rlistings` allows listing of the container's content.

With write ACL set to `"*:*"`, the container can be updated by anybody, so use it with care.

The `-r` and `--read-acl` commands as well as `-w` and `--write-acl` are the short and long forms of the same flag. That is, `-r` and `--read-acl` are interchangeable as well as `-w` and `--write-acl`.

Once access to containers is enabled for other users, find the URL of a container with the following command:

```
swift stat -v container | grep URL
```

To access another user's container once access been enabled, use this command:

```
swift --os-storage-url URL list
```

If you are always accessing the same storage URL, it can be set as the `OS_STORAGE_URL` environment variable.

9
OpenStack Orchestration Using Heat and Ansible

In this chapter, we will cover the following topics:

- Introduction – orchestrating with OpenStack
- Creating your first stack with Heat
- Launching your stack with Heat
- Viewing the resources and output of a stack created with Heat
- Deleting a Heat stack
- Updating a Heat stack
- Installing and configuring Ansible for OpenStack
- Using Ansible to launch instances
- Using Ansible to orchestrate software installation
- Using Ansible to orchestrate software installations across multiple instances
- Using Ansible to fully orchestrate the creation of a web server and load balancer stack

Introduction – orchestrating with OpenStack

OpenStack is chosen as a platform for many reasons, but one that frequently tops the list is orchestration. Without an element of orchestration in your OpenStack environment, you have a powerful turbo engine car that is just used for the school-run. As with any cloud environment, there are various tools to help with your orchestrated workloads, but out of the box, OpenStack provides Heat, the orchestration engine.

With Heat, you can define rich environments in a template, such as a multi-tier web application, which allows users consistency in launching these relatively complex deployments. I view the Heat orchestration templates (known as **HOT** (**Heat Orchestration Template**) – get it?) as a recipe that is written in **YAML** (**Yet Another Markup Language**). You define your ingredients that make up the environment. In a cooking recipe, this would be listing the amount of chocolate, flour, and sugar that is required for something like a cake. In a HOT ()file, this is the parameters section. You define the flavor of instances, the images used, and what networks the instances should launch against.

Like any good recipe, you can override these defaults—so if you fancy experimenting with jam instead of chocolate sauce, or varying the amount of sugar required—you can adjust these. Also, with Heat you do this in an environment file. This file is laid out as though you're assigning values to the parameters. For example, an input parameter in the HOT file might be `image_name`, and in an environment file, you assign `image_name=ubuntu-image`.

The next section in the HOT file is the largest and most complex as it is the method section of the recipe – the "how all the ingredients create a cake" section. In Heat, this is the start of the resources section. The resources section describes how the instances interact with each other. For example, a load balancer might get created during the Heat run that includes three web servers. The load balancer resources would have a method to attach these three unknown web servers to the load balance pool for that resource to be complete.

The final section is the output section. In a cooking recipe, you'd be taking the cake out of the oven. In OpenStack, this will be the end result—your multi-tier web application. However, with OpenStack and the nature of launching a number of instances into an environment that may not have even had a network created at that point, it would be hard to know the IP addresses that have been used, which may be needed in order to access the deployed stack. So, for this, we have the output section where a user of OpenStack can interrogate the stack and get useful information so that the launched stack is usable.

However, we are not limited to using the Heat orchestration engine within OpenStack when it comes to instance and application life cycle management. As introduced in *Chapter 1*, *Installing OpenStack with Ansible*, Ansible is a great example of a platform independent tool that we can use to help orchestrate tasks. Using Ansible modules specific to the target environment, such as OpenStack, we can launch instances and perform software installations in a structured way that may suit users of both OpenStack and other cloud environments.

The basic structure of Ansible is quite straightforward. In its simplest form, it has a notion of playbooks, plays, and tasks. If you study the OpenStack-Ansible playbooks that were described in *Chapter 1*, *Installing OpenStack with Ansible* you'll get an insight into advanced features not covered in this chapter, which allow you to extend your OpenStack-based cloud environments into fully orchestrated masterpieces!

Creating your first stack

With Heat, we can create a wide variety of templates from spinning up basic instances, to creating complete environments for an application. In this section, we will show the basics of Heat by spinning up an instance and attaching it to an existing Neutron network, and assigning a floating IP to it. Heat templates describe the resources being used, the type and size of the instances, the network an instance will be attached to, among other pieces of information required to run that environment.

In this section, we will show you how to use a HOT file to spin up two web servers running Apache, connected behind a third instance running HAProxy acting as the load balancer.

Getting ready

Ensure that you are logged onto a correctly configured OpenStack client and can access the OpenStack environment. Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use Heat.

How to do it...

In this section, we will download a HOT file called `cookbook.yaml`, which will describe our instance and the network to attach it to:

1. First, we download the HOT file from the Cookbook GitHub repository:

   ```
   wget -O cookbook.yaml
   https://raw.githubusercontent.com/OpenStackCookbook/
   OpenStackCookbook/master/cookbook.yaml
   ```

2. Heat takes input parameters from the command line, or from an environment file, which get passed to the template. These parameters are seen at the top of the HOT file, as shown here:

   ```
   parameters:
     key_name:
       type: string
       description: Name of keypair to assign to servers
     image:
       type: string
   ```

```
    description: Name of image to use for servers
  flavor:
    type: string
    description: Flavor to use for servers
  public_net_id:
    type: string
    description: >
      ID of public network for which floating IP addresses will be
allocated
  private_net_id:
    type: string
    description: ID of private network into which servers get
deployed
  private_subnet_id:
    type: string
    description: ID of private sub network into which servers get
deployed
```

3. As can be seen, we expect to pass in various parameters when we launch this template. Ensure that we have these details by running the following commands:

   `openstack keypair list`

   `openstack image list`

   `openstack flavor list`

   `openstack network list`

   The `openstack network list` output may look like the following:

```
+--------------------------------------+-------------+--------------------------------------+
| ID                                   | Name        | Subnets                              |
+--------------------------------------+-------------+--------------------------------------+
| 2da8979e-dcf8-4eb8-b207-f33bfce4a15a | GATEWAY_NET | a1b0e5e8-9f91-428a-aa69-2e41ee7a1c7a |
| 78a5a119-c27a-41c4-8310-5c04d3a6bc31 | private-net | 3cee2bb9-5673-4a6e-bb1e-8cb66be066b2 |
+--------------------------------------+-------------+--------------------------------------+
```

4. With the information at hand, we create an environment file that will be used to store our parameters that we will pass to the HOT file when we launch the stack. Create `cookbook-env.yaml` in the same directory as `cookbook.yaml` with the following contents based on the output of the previous commands (adjust to suit your environment):

```
parameters:
  key_name: demokey
  image: xenial-image
  flavor: m1.tiny
  public_net_id: 2da8979e-dcf8-4eb8-b207-f33bfce4a15a
  private_net_id: 78a5a119-c27a-41c4-8310-5c04d3a6bc31
  private_subnet_id: 3cee2bb9-5673-4a6e-bb1e-8cb66be066b2
```

How it works...

Heat Orchestration Templates (HOT) are YAML files that describe our environment, or "Stacks" as they're known. The basic templates generally have the following structure:

- `description:`
- `parameters:`
- `resources:`
- `outputs:`

The `description:` section has a number of words that helps a user understand what is expected to occur when the template is used.

The `parameters:` section defines the input variables, for example, the type of image(s) to be used, the network(s) to attach the instances on, and the key pair name to associate with the instances. Parameters are arbitrary and can contain any information needed to execute the template properly. The parameters: section works directly with the information found in the accompanying environment file (as specified by the `--environment` parameter). Each parameter must either have a default value or be specified in the environment file for the stack to launch successfully.

The `resources:` section is usually the biggest section as it describes the environment. It can describe the instances that will be used, the naming of them, which networks to attach, and essentially how all of the elements relate to each other and how the environment is orchestrated. Explanations of how best to write these resources are beyond the scope of this book.

The `outputs:` section refers to the "return" values from running the stack. For example, a user will need to know how to access a particular stack that has just been created. Random IPs and hostnames can all be assigned as normal operation of running stacks, so being able to interrogate the right information in order to access the environment is a must.

Launching your stack with Heat

To launch a Heat stack we need three things: a *name* for the stack, the *template* (HOT) that describes the deployment, and finally, the *environment file* that fills in the blanks of the input parameters.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment. Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack.

If you get the `openstack: 'stack' is not an openstack` command, refer to `openstack --help`.

Ensure that the `python-heatclient` package is installed:

```
sudo -H pip install python-heatclient
```

Also ensure that you have downloaded the example `cookbook.yaml` Heat template and have created the environment file, as described in the previous recipe.

How to do it...

In this section, we will download a HOT file called `cookbook.yaml`, which will describe our instance and the network to attach it to:

1. We will launch the stack with the following commands:

```
openstack stack create myStack
    --template cookbook.yaml
    --environment cookbook-env.yaml
```

Tip: You can use the `-t` flags instead of `--template`, and `-e` instead of `--environment`.

This will produce an output like the following:

```
+--------------------+--------------------------------------------------------------------------------------------+
| Field              | Value                                                                                      |
+--------------------+--------------------------------------------------------------------------------------------+
| id                 | 17061d1f-e885-4051-9957-781049e27b7b                                                       |
| stack_name         | myStack                                                                                    |
| description        | HOT template to deploy two instances running Apache, and an extra instance running        |
|                    | HA Proxy that load balances traffic between then assigns a floating IP addresses  to      |
|                    | the HA Proxy instance from the Public Network                                              |
|                    |                                                                                            |
| creation_time      | 2017-07-19T13:10:59Z                                                                       |
| updated_time       | None                                                                                       |
| stack_status       | CREATE_IN_PROGRESS                                                                         |
| stack_status_reason | Stack CREATE started                                                                      |
+--------------------+--------------------------------------------------------------------------------------------+
```

2. To view a list of stacks, execute the following command:

```
openstack stack list
```

This will bring back a list of stacks currently running:

```
+--------------------------------------+------------+-----------------+----------------------+--------------+
| ID                                   | Stack Name | Stack Status    | Creation Time        | Updated Time |
+--------------------------------------+------------+-----------------+----------------------+--------------+
| 17061d1f-e885-4051-9957-781049e27b7b | myStack    | CREATE_COMPLETE | 2017-07-19T13:10:59Z | None         |
+--------------------------------------+------------+-----------------+----------------------+--------------+
```

Note **Stack Status**. A successful launch is when it is marked as **CREATE_COMPLETE**.

How it works...

Launching a stack is simple. We will specify the HOT file with the `--template` parameter, and then we will specify the inputs that get described in the template in a file that we specify with the `--environment` parameter.

The syntax is as follows:

```
openstack stack create nameOfStack
    --template template.yaml
    --environment template-env.yaml
```

Note that the name of the stack must be unique in your project.

Viewing the resources and output of a stack created with Heat

A stack is an orchestrated set of services, where the user launching the stack shouldn't care too much about what IP addresses were assigned. However, the application stack has been launched to serve a purpose and therefore it is helpful to know how to access it! To access the environment, the user interrogates the "outputs" of the stack, which were defined as part of the template. In this example, we are concerned about how to access the website running behind the HAProxy server. The HAProxy server has been assigned a floating IP from the GATEWAY_NET network, and it is assumed that this is how the application will be accessed.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment. Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use Heat.

How to do it...

To view the application stack and get information about how to access it, carry out the following steps:

1. You can view more details of a stack using the following command:

   ```
   openstack stack show myStack
   ```

   This will bring back a number of details about the created stack:

```
+----------------------+--------------------------------------------------------------------------------------+
| Field                | Value                                                                                |
+----------------------+--------------------------------------------------------------------------------------+
| id                   | 17061d1f-e885-4051-9957-781049e27b7b                                                 |
| stack_name           | myStack                                                                              |
| description          | HOT template to deploy two instances running Apache, and an extra instance running |
|                      | HA Proxy that load balances traffic between then assigns a floating IP addresses   |
|                      | to the HA Proxy instance from the Public Network                                     |
|                      |                                                                                      |
| creation_time        | 2017-07-19T13:10:59Z                                                                 |
| updated_time         | None                                                                                 |
| stack_status         | CREATE_COMPLETE                                                                      |
| stack_status_reason  | Stack CREATE completed successfully                                                  |
| parameters           | OS::project_id: 052b66ad725e4ab2952f8c052fc577d9                                     |
|                      | OS::stack_id: 17061d1f-e885-4051-9957-781049e27b7b                                   |
|                      | OS::stack_name: myStack                                                              |
|                      | flavor: m1.tiny                                                                      |
|                      | image: xenial-image                                                                  |
|                      | key_name: demokey                                                                    |
|                      | private_net_id: 78a5a119-c27a-41c4-8310-5c04d3a6bc31                                 |
|                      | private_subnet_id: 3cee2bb9-5673-4a6e-bb1e-8cb66be066b2                              |
|                      | public_net_id: 2da8979e-dcf8-4eb8-b207-f33bfce4a15a                                  |
|                      |                                                                                      |
| outputs              | - description: Floating IP address of haproxy in public network                      |
|                      |   output_key: haproxy_public_ip                                                      |
|                      |   output_value: 192.168.100.108                                                      |
|                      | - description: IP address of webserver2 in private network                           |
|                      |   output_key: webserver2_private_ip                                                  |
|                      |   output_value: 10.10.10.11                                                          |
|                      | - description: IP address of webserver1 in private network                           |
|                      |   output_key: webserver1_private_ip                                                  |
|                      |   output_value: 10.10.10.5                                                           |
|                      |                                                                                      |
| links                | - href: http://172.29.236.10:8004/v1/052b66ad725e4ab2952f8c052fc577d9/stacks/mySta |
|                      | ck/17061d1f-e885-4051-9957-781049e27b7b                                              |
|                      |   rel: self                                                                          |
|                      |                                                                                      |
| parent               | None                                                                                 |
| disable_rollback     | True                                                                                 |
| deletion_time        | None                                                                                 |
| stack_user_project_id | 5a2420d4ed924825ae465dc0bebc54af                                                    |
| capabilities         | []                                                                                   |
| notification_topics  | []                                                                                   |
| stack_owner          | None                                                                                 |
| timeout_mins         | None                                                                                 |
| tags                 | None                                                                                 |
+----------------------+--------------------------------------------------------------------------------------+
```

2. A section in the template references *outputs*. Outputs allow a user to interrogate these values so that they can access the running stack. Without this, the user would have to do more digging into the running systems to find out what IP addresses were assigned to the instances that make up the stack. To see a list of outputs associated with our running stack, execute the following command:

   ```
   openstack stack output list myStack
   ```

   This will bring back the following output:

   ```
   +----------------------+-------------------------------------------------+
   | output_key           | description                                     |
   +----------------------+-------------------------------------------------+
   | haproxy_public_ip    | Floating IP address of haproxy in public network |
   | webserver2_private_ip | IP address of webserver2 in private network     |
   | webserver1_private_ip | IP address of webserver1 in private network     |
   +----------------------+-------------------------------------------------+
   ```

3. To view a particular value, such as the public IP (*floating IP*) assigned to our HAProxy instance, we can access the websites that are running on private addresses behind the load balancer. To do so, issue the following command:

   ```
   openstack stack output show myStack haproxy_public_ip
   ```

   This gives the IP address that we would then use to access this particular service set up as a stack:

   ```
   +--------------+-------------------------------------------------+
   | Field        | Value                                           |
   +--------------+-------------------------------------------------+
   | description  | Floating IP address of haproxy in public network |
   | output_key   | haproxy_public_ip                               |
   | output_value | 192.168.100.108                                 |
   +--------------+-------------------------------------------------+
   ```

4. In this example application stack, we can then use the `http://192.168.100.108/` address, which will send the request to either of the web servers that are running, configured as part of this HAProxy load balancer demonstration.

How it works...

A stack is designed to take a number of inputs, performs some actions, and produces a number of instances running an application ready for service. However, this hands-off approach means a lot of decisions are automatically dictated by OpenStack—predominantly because the instances get served from DHCP-enabled subnets. In order to find out the state of the stack and information about how to access the stack, a user would interrogate the **outputs**, which were described in the template. In the example template, the output section looks like the following:

```
outputs:
  webserver1_private_ip:
    description: IP address of webserver1 in private network
    value: { get_attr: [ webserver1, first_address ] }
  webserver2_private_ip:
    description: IP address of webserver2 in private network
    value: { get_attr: [ webserver2, first_address ] }
  haproxy_public_ip:
    description: Floating IP address of haproxy in public network
    value: { get_attr: [ haproxy_floating_ip, floating_ip_address ] }
```

The outputs that have more information are labelled as follows:

- `webserver1_private_ip`
- `webserver2_private_ip`
- `haproxy_public_ip`

In this recipe, we specifically targeted `haproxy_public_ip`, as this is how we would access the web service that we created. We issued the following command to do this:

```
openstack stack output show myStack haproxy_public_ip
```

Deleting a Heat stack

To delete a running Heat stack we will make a simple call as shown in this recipe.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment. Refer to *Chapter 2*, *The OpenStack Client* for details of setting up your environment to use OpenStack.

How to do it...

In this section, we'll show how to delete a stack.

1. To delete a running stack named `myStack`, issue the following command:

   ```
   openstack stack delete myStack
   ```

2. You will be prompted to confirm this deletion, as shown here. Type `y` to continue destroying the stack:

   ```
   vagrant@openstack-client:~$ openstack stack delete myStack
   Are you sure you want to delete this stack(s) [y/N]? y
   ```

3. You can check on the status of the deletion by listing the created stacks:

   ```
   openstack stack list
   ```

   This will bring back an empty list if there is no stack to show or the following during deletion:

```
+--------------------------------------+------------+----------------------------------+--------------------+----------------------+----------------------+
| ID                                   | Stack Name | Project                          | Stack Status       | Creation Time        | Updated Time         |
+--------------------------------------+------------+----------------------------------+--------------------+----------------------+----------------------+
| 0f2ba0b6-a047-4e31-9ec9-9ccb0e9ca687 | myStack    | 31d0c145fad24a53a7838f2629465dab | DELETE_IN_PROGRESS | 2017-12-14T10:33:21Z | 2017-12-14T15:40:03Z |
+--------------------------------------+------------+----------------------------------+--------------------+----------------------+----------------------+
```

How it works...

In very much the same way that we can launch a stack easily, deleting one is achieved by simply specifying which stack we want to destroy and using the `stack delete` command.

If you prefer to destroy a stack without confirmation, use the following syntax:

```
openstack stack delete nameOfStack -y
```

Updating a Heat stack

Our running Stack is based on templates, so this allows us to modify our application stack by altering the inputs. If we wanted to change the size of a flavor, or the key used, you can trigger a rebuild of the instances in the stack by altering the inputs and issuing the `stack update` command.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment. Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack.

How to do it...

In this section, we'll modify the environment file to change a flavor from `m1.tiny` to `m1.large`. (Ensure that you have a valid flavor with this name before continuing!)

1. We first edit the environment file, called `cookbook-env.yaml`, to show the changes we want to make to our running stack:

```
parameters:
  key_name: demokey
  image: xenial-image
  flavor: m1.large
  public_net_id: 2da8979e-dcf8-4eb8-b207-f33bfce4a15a
  private_net_id: 78a5a119-c27a-41c4-8310-5c04d3a6bc31
  private_subnet_id: 3cee2bb9-5673-4a6e-bb1e-8cb66be066b2
```

2. Ensure that the stack is running without issues by viewing the resources:

```
openstack stack show myStack
```

   This will bring back an output like the following:

```
+----------------------+--------------------------------------------------------------------------------------+
| Field                | Value                                                                                |
+----------------------+--------------------------------------------------------------------------------------+
| id                   | 17061d1f-e885-4051-9957-781049e27b7b                                                 |
| stack_name           | myStack                                                                              |
| description          | HOT template to deploy two instances running Apache, and an extra instance running |
|                      | HA Proxy that load balances traffic between then assigns a floating IP addresses   |
|                      | to the HA Proxy instance from the Public Network                                     |
|                      |                                                                                      |
| creation_time        | 2017-07-19T13:10:59Z                                                                 |
| updated_time         | None                                                                                 |
| stack_status         | CREATE_COMPLETE                                                                      |
| stack_status_reason  | Stack CREATE completed successfully                                                  |
| parameters           | OS::project_id: 052b66ad725e4ab2952f8c052fc577d9                                     |
|                      | OS::stack_id: 17061d1f-e885-4051-9957-781049e27b7b                                   |
|                      | OS::stack_name: myStack                                                              |
|                      | flavor: m1.tiny                                                                      |
|                      | image: xenial-image                                                                  |
|                      | key_name: demokey                                                                    |
|                      | private_net_id: 78a5a119-c27a-41c4-8310-5c04d3a6bc31                                 |
|                      | private_subnet_id: 3cee2bb9-5673-4a6e-bb1e-8cb66be066b2                              |
|                      | public_net_id: 2da8979e-dcf8-4eb8-b207-f33bfce4a15a                                  |
|                      |                                                                                      |
| outputs              | - description: Floating IP address of haproxy in public network                      |
|                      |   output_key: haproxy_public_ip                                                      |
|                      |   output_value: 192.168.100.108                                                      |
|                      | - description: IP address of webserver2 in private network                           |
|                      |   output_key: webserver2_private_ip                                                  |
|                      |   output_value: 10.10.10.11                                                          |
|                      | - description: IP address of webserver1 in private network                           |
|                      |   output_key: webserver1_private_ip                                                  |
|                      |   output_value: 10.10.10.5                                                           |
|                      |                                                                                      |
| links                | - href: http://172.29.236.10:8004/v1/052b66ad725e4ab2952f8c052fc577d9/stacks/mySta   |
|                      | ck/17061d1f-e885-4051-9957-781049e27b7b                                              |
|                      |   rel: self                                                                          |
|                      |                                                                                      |
| parent               | None                                                                                 |
| disable_rollback     | True                                                                                 |
| deletion_time        | None                                                                                 |
| stack_user_project_id | 5a2420d4ed924825ae465dc0bebc54af                                                    |
| capabilities         | []                                                                                   |
| notification_topics  | []                                                                                   |
| stack_owner          | None                                                                                 |
| timeout_mins         | None                                                                                 |
| tags                 | None                                                                                 |
+----------------------+--------------------------------------------------------------------------------------+
```

3. We will now use the updated environment file to modify the running stack:

```
openstack stack update myStack --existing
```

Tip: We're using the parameter **`--existing`** to avoid specifying the template and environment file again.

This will bring back an output like the following, showing the update has started:

```
+---------------------+--------------------------------------------------------------------------------------------+
| Field               | Value                                                                                      |
+---------------------+--------------------------------------------------------------------------------------------+
| id                  | 17061d1f-e885-4051-9957-781049e27b7b                                                       |
| stack_name          | myStack                                                                                    |
| description         | HOT template to deploy two instances running Apache, and an extra instance running        |
|                     | HA Proxy that load balances traffic between then assigns a floating IP addresses  to      |
|                     | the HA Proxy instance from the Public Network                                              |
|                     |                                                                                            |
| creation_time       | 2017-07-19T13:10:59Z                                                                       |
| updated_time        | 2017-07-20T10:06:09Z                                                                       |
| stack_status        | UPDATE_IN_PROGRESS                                                                         |
| stack_status_reason | Stack UPDATE started                                                                       |
+---------------------+--------------------------------------------------------------------------------------------+
```

4. Also we can view the state of the stack once this has been completed, to show the reflected change:

```
openstack stack show myStack
```

This will bring back an output like the following (note that the flavor has changed from `m1.tiny` to `m1.large`). Also note that the IP addresses have not changed:

```
+-----------------------+--------------------------------------------------------------------------------------------+
| Field                 | Value                                                                                      |
+-----------------------+--------------------------------------------------------------------------------------------+
| id                    | 17061d1f-e885-4051-9957-781049e27b7b                                                       |
| stack_name            | myStack                                                                                    |
| description           | HOT template to deploy two instances running Apache, and an extra instance running        |
|                       | HA Proxy that load balances traffic between then assigns a floating IP addresses          |
|                       | to the HA Proxy instance from the Public Network                                           |
|                       |                                                                                            |
| creation_time         | 2017-07-19T13:10:59Z                                                                       |
| updated_time          | 2017-07-20T10:06:09Z                                                                       |
| stack_status          | UPDATE_COMPLETE                                                                            |
| stack_status_reason   | Stack UPDATE completed successfully                                                        |
| parameters            | OS::project_id: 052b66ad725e4ab2952f8c052fc577d9                                           |
|                       | OS::stack_id: 17061d1f-e885-4051-9957-781049e27b7b                                         |
|                       | OS::stack_name: myStack                                                                    |
|                       | flavor: m1.large                                                                           |
|                       | image: xenial-image                                                                        |
|                       | key_name: demokey                                                                          |
|                       | private_net_id: 78a5a119-c27a-41c4-8310-5c04d3a6bc31                                       |
|                       | private_subnet_id: 3cee2bb9-5673-4a6e-bb1e-8cb66be066b2                                    |
|                       | public_net_id: 2da8979e-dcf8-4eb8-b207-f33bfce4a15a                                        |
|                       |                                                                                            |
| outputs               | - description: Floating IP address of haproxy in public network                            |
|                       |   output_key: haproxy_public_ip                                                            |
|                       |   output_value: 192.168.100.108                                                            |
|                       | - description: IP address of webserver2 in private network                                 |
|                       |   output_key: webserver2_private_ip                                                        |
|                       |   output_value: 10.10.10.11                                                                |
|                       | - description: IP address of webserver1 in private network                                 |
|                       |   output_key: webserver1_private_ip                                                        |
|                       |   output_value: 10.10.10.5                                                                 |
|                       |                                                                                            |
| links                 | - href: http://172.29.236.10:8004/v1/052b66ad725e4ab2952f8c052fc577d9/stacks/mySta         |
|                       | ck/17061d1f-e885-4051-9957-781049e27b7b                                                    |
|                       |   rel: self                                                                                |
|                       |                                                                                            |
| parent                | None                                                                                       |
| disable_rollback      | True                                                                                       |
| deletion_time         | None                                                                                       |
| stack_user_project_id | 5a2420d4ed924825ae465dc0bebc54af                                                           |
| capabilities          | []                                                                                         |
| notification_topics   | []                                                                                         |
| stack_owner           | None                                                                                       |
| timeout_mins          | None                                                                                       |
| tags                  | None                                                                                       |
+-----------------------+--------------------------------------------------------------------------------------------+
```

How it works...

The OpenStack Orchestration service, Heat, is designed to follow a template to provide a running service to end users. Everything is automated from start to finish. This crucial feature allows us to update a running stack, effectively redeploying the stack with updates, which runs through the fully automated routine to restore the service, on the same IP addresses, but with the required changes.

The syntax for updating the stack is as follows:

```
openstack stack update nameOfStack
    --environment updatedEnvironmentFile.yaml
    --template originalStackTemplate.yml
```

In our example, we omitted the `--environment` and `--template` parameters as we made the required change to our stack directly in the environment file used originally. This allowed for a simpler syntax:

```
openstack stack update nameOfStack --existing
```

Installing and configuring Ansible for OpenStack

Ansible has relatively few prerequisites that are not installed on most Linux- and macOS-based systems. However, there are a few steps to follow before we can use Ansible for managing our OpenStack environment.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment. Refer to *Chapter 2*, *The OpenStack Client*, for details of setting up your environment to use OpenStack.

The version of Ansible 2.x requires Python 2.6 or 2.7. Most modern Linux distributions and macOS/OS X have this already installed. If you were able to successfully execute the `openstack` commands as described in *Chapter 2*, *The OpenStack Client*, then you're good to go here.

You may need to install Shade. Shade is a simple client library for interacting with OpenStack clouds. Red Hat and CentOS environments don't have this installed by default. Install it with the following command:

```
sudo pip install shade
```

Be aware that Shade may pull in other dependencies that may break your environment. It is suggested that you use a **virtual environment** (**venv**) to avoid this issue.

How to do it...

As we're performing this on our client machine, ensure that you have the necessary permissions to install software. When ready, carry out the following steps depending on your chosen operating system.

Ubuntu

For Ubuntu, we can use the Ansible PPA as follows:

1. First, ensure that we can add **PPA** (**Personal Package Archives**) by installing the following tool:

   ```
   sudo apt-get install software-properties-common
   ```

2. Next, we will add the PPA:

   ```
   sudo apt-add-repository ppa:ansible/ansible
   ```

3. Finally, we will run the installation:

   ```
   sudo apt-get update
   sudo apt-get install ansible
   ```

macOS/OS X (and for those wanting to use pip)

For macOS, we can use pip as follows:

1. Ensure that `pip` is available:

   ```
   sudo easy_install pip
   ```

2. Next, use `pip` to install Ansible:

   ```
   sudo pip install ansible
   ```

Verifying the installation

To verify the installation, issue the following command:

```
ansible --version
```

This should produce an output like the following:

```
ansible 2.3.2.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = Default w/o overrides
  python version = 2.7.6 (default, Oct 26 2016, 20:30:19) [GCC 4.8.4]
```

How it works...

In order for us to be able to use Ansible to manage our OpenStack environment, we must ensure that we have a good working Ansible set up. The preceding steps merely helped us install Ansible onto our client using the tools available for that operating system.

Using Ansible to launch instances

Launching an instance using Ansible is a convenient, platform agnostic method. While we have to specify how to do this for OpenStack, as a particular task, an Ansible playbook could be extended to allow a user to use the same Ansible command to launch an instance on any cloud. This recipe is a very basic introduction to the use of Ansible with OpenStack.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment that has Ansible installed.

How to do it...

Ansible executes tasks in what is known as a playbook. In this example, we will create a simple task that launches a specific instance called `cookbook1`:

1. The first step is to create the Ansible playbook for our tasks that will launch our instance. Create the following file called `launch-instance.yml` on your client, in a directory of your choosing:

```
- name: Launch instance on OpenStack
  hosts: localhost
  gather_facts: false
  tasks:
  - name: Deploy an instance
    os_server:
       state: present
       name: cookbook1
       image: xenial-image
```

```
key_name: demokey
timeout: 200
flavor: m1.tiny
network: private-net
verify: false
```

2. Once that has been described, we will simply run that particular task using the `ansible-playbook` command, as follows:

   ```
   source openrc
   ansible-playbook launch-instance.yml
   ```

   This will bring back the familiar Ansible output like the following:

```
 [WARNING]: provided hosts list is empty, only localhost is available


PLAY [Launch instance on OpenStack] ********************************************

TASK [Deploy an instance] ******************************************************
changed: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=1    changed=1    unreachable=0    failed=0
```

This is a `localhost` task, so the warning can be ignored.

3. We can verify that an instance was launched in OpenStack by viewing the server listing as follows:

   ```
   openstack server list
   ```

   This will bring back an output like the following:

```
+--------------------------------------+-----------+--------+-----------------------------------------+--------------+
| ID                                   | Name      | Status | Networks                                | Image Name   |
+--------------------------------------+-----------+--------+-----------------------------------------+--------------+
| 3cc3f42e-be89-40a6-878f-4afe81a9d984 | cookbook1 | ACTIVE | private-net=10.10.10.10, 192.168.100.101 | xenial-image |
+--------------------------------------+-----------+--------+-----------------------------------------+--------------+
```

Note that the task automatically assigned a public floating IP address from the `GATEWAY_NET` network. This is an important detail as Ansible can do much more than just launch instances. If we want to be able to install and configure instances, Ansible must be able to SSH from the client to the running instance. Private tenant networks are generally not accessible; therefore, Ansible would use the public routed network to access the instance, just like you would if you were to SSH to it.

How it works...

Launching an instance using Ansible uses the `os_server` Ansible module. This is available from Ansible 2.0 onwards. The `os_server` module takes a number of parameters that describe the usual parameters you would expect when launching instances using the command line.

Note that we didn't specify any authentication details as part of the task. This is because this module interprets our shell environment variables, just like we would use when executing OpenStack client tools.

You will notice that one of the entries in the task denotes the following:

```
os_server:
  state: present
```

This has a specific intent in Ansible, as Ansible is designed to give state consistency of running that task, despite how many times that task may get run. This simple statement basically says that this instance must be present. If not, it will launch that instance. Once it has launched, it satisfies that predicate. In the Ansible output, you will see that the overall run says **ok=1 changed=1**. This means that it changes the state of this environment. In other words, it launched the instance (which is a change of state).

However, if we run the task again, we get the following subtle change in the output denoting that the task didn't need to run to satisfy the fact the instance needed to be "present" (that is, running):

```
 [WARNING]: provided hosts list is empty, only localhost is available

PLAY [Launch instance on OpenStack] ********************************************

TASK [Deploy an instance] ******************************************************
ok: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=1    changed=0    unreachable=0    failed=0
```

Note that the task executed successfully, but didn't need to change anything as denoted by the **changed=0** output in the **PLAY RECAP** line.

See also

- Visit `http://docs.ansible.com/ansible/latest/os_server_module.html` for more information
- For those who do not want to source environment variables into their playbooks using the `source openrc` method, visit `https://docs.openstack.org/shade/latest/` to set up a `clouds.yaml` cloud environment file

Using Ansible to orchestrate software installation

Launching an instance using Ansible doesn't provide a user much beyond consistency, in that a playbook describes the end state of the environment: every time a user runs the task, it will either need to launch that specific instance to ensure that it is present, or it will skip that task because the instance is already running. However, we can achieve a lot more with Ansible beyond just launching a virtual machine. In this recipe, we will launch another instance that will install and start Apache.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment that has Ansible installed.

How to do it...

Ansible executes playbooks of tasks. In this example, we will extend the task that launches a specific instance to allow Ansible to then connect to that instance and install Apache:

1. We start off by extending the Ansible playbook introduced in the previous recipe to include the creation of *security group rules* (this is to ensure that Ansible can access to the instance on port `22`, and we are ultimately able to access the running web server on port `80`). Create the file called `orchestrate-instance.yml` with the following contents:

```
- name: Launch instances on OpenStack
  hosts: localhost
  gather_facts: false

  tasks:
    - name: Create ansible security group
      os_security_group:
        state: present
        name: ansible
        verify: false
    - name: Create a rule to allow SSH connections
      os_security_group_rule:
        security_group: ansible
        protocol: tcp
        port_range_min: 22
        port_range_max: 22
        remote_ip_prefix: 0.0.0.0/0
```

```
      verify: false
  - name: Create webserver security group
    os_security_group:
      state: present
      name: webserver
      verify: false
  - name: Create rule to allow http connections
    os_security_group_rule:
      security_group: webserver
      protocol: tcp
      port_range_min: 80
      port_range_max: 80
      remote_ip_prefix: 0.0.0.0/0
      verify: false
```

2. Once we have our **security groups** configured, we can then ensure that these security group rules are included in the **task** that launches the instance. We also ensure that we include a new entry called `register`. This allows us to set a variable associated with this instance that we can then refer to in other tasks. Carry on editing this file with the following tasks:

```
  - name: Deploy an instance
    os_server:
      state: present
      name: cookbook1
      image: xenial-image
      key_name: demokey
      timeout: 200
      flavor: m1.tiny
      network: private-net
      security_groups: default,ansible,webserver
      verify: false
    register: nova_cookbook
```

3. Next, we will add in a task that adds this particular instance to an internal in-memory inventory that we can then access later on in the playbook. As part of this inventory, we're telling Ansible that when the particular inventory item is accessed, when Ansible wants to connect to it (via `ssh`), it will use a particular IP address (in our case, the public floating IP). Carry on editing the file and add this next entry as shown as follows. As we are using YAML, ensure that the spacing matches for each element. For example, this additional `- name: Add instance to Inventory` block must match the same column as the previous block `- name: Deploy an instance`, as it is a task that is part of the same play:

```
  - name: Add instance to Inventory
    add_host: name=cookbook1 groups=webservers
```

```
                ansible_ssh_host={{ nova_cookbook.server.accessIPv4
}}
```

4. Next, we will add in a new play that tells Ansible to wait for this instance to complete its boot process. As Ansible uses SSH to perform its tasks, it makes sense to only continue when the SSH daemon is running and accepting connections. Ensure that your private key running the Ansible task matches the public key portion described in `key_name:`.

Note that this is a new play and task, so ensure that this entry begins at the start of the line at column 0.

```
- name: Wait for port 22 to be ready
  hosts: webservers
  gather_facts: False
  tasks:
    - local_action: wait_for port=22 host="{{ ansible_ssh_host }}"
search_regex=OpenSSH delay=10
```

5. This final set of tasks perform the steps inside the running instance; this next task performs the installation of Apache on this running instance. Ansible knows to operate on this instance because this set of tasks is performed against the `webservers` group of hosts. We registered this new group, in the in-memory inventory, in the task in step 2.

The `pre_tasks:` section is *optional* and may not be needed in all circumstances. The example in this book was created using an Ubuntu 16.04 image. Ubuntu 16.04 doesn't install Python 2 by default, however this particular Ansible `apt` module, that will ultimately install Apache, expects to execute Python 2 code in order to work. So, we do an initial `raw` command, that doesn't execute any Python, to run some shell script to set some things up for us. This example is also further complicated if you are using the accompanying Vagrant environment, which the instances do not have direct access to the internet. So, as part of the `pre_tasks:` section, we also configure a proxy server for APT to use.

```
- hosts: webservers
  remote_user: ubuntu
  become: yes
  gather_facts: no
  pre_tasks:
    - name: Set APT proxy
      raw: echo "Acquire::http::Proxy
\"http://192.168.1.20:3128\";" > /etc/apt/apt.conf
```

```
    - name: 'install python2'
      raw: sudo apt-get -y install python-simplejson
  tasks:
    - name: Ensure Apache is installed
      apt: name=apache2 state=latest
    - name: Ensure that Apache is started
      service: name=apache2 state=started
```

Hint: This playbook can also be found at `https://raw.githubusercontent.com/OpenStackCookbook/vagrant-openstack/master/orchestrate-instance.yml`.

Once this file has been created, save and exit. Then run the following commands which will launch an instance and install Apache:

```
source openrc
```

```
ansible-playbook orchestrate-instance.yml
```

This will bring back the familiar Ansible output like the following:

```
 [WARNING]: provided hosts list is empty, only localhost is available


PLAY [Launch instance on OpenStack] ********************************************

TASK [Deploy an instance] ******************************************************
ok: [localhost]

TASK [Add instance to Inventory] ***********************************************
changed: [localhost]

PLAY [Wait for port 22 to be ready] ********************************************

TASK [wait_for] ****************************************************************
ok: [cookbook1 -> localhost]

PLAY [webservers] **************************************************************

TASK [Set APT proxy] ***********************************************************
changed: [cookbook1]

TASK [install python2] *********************************************************
changed: [cookbook1]

TASK [Ensure Apache is installed] **********************************************
changed: [cookbook1]

TASK [Ensure that Apache is running] *******************************************
ok: [cookbook1]

PLAY RECAP *********************************************************************
cookbook1                  : ok=5    changed=3    unreachable=0    failed=0
localhost                  : ok=2    changed=1    unreachable=0    failed=0
```

How it works...

What we did here was extending the original, simple *playbook* that launched a single instance, and adding in subsequent *tasks* that allow us to install some software onto that instance once it has completed its boot process. The important details of how this is achieved are described here.

In the first *play*, named `Launch instance on OpenStack`, we first configure some tasks to set up our security group rules. By default, there are no incoming connections allowed, and Ansible uses SSH to perform its tasks, so we have to at least ensure that TCP port 22 is open. We also configure rules appropriate for the service that we are installing. In this case, we are running Apache, so we open up TCP port 80. Once the security groups are configured, we then have a *task* named `Deploy an instance`. We ensure this instance is launched with the appropriate security groups that we have just configured, and we also register that instance in a variable named `nova_cookbook`.

Ansible uses an **inventory** of data to allow subsequent plays and tasks to access details that Ansible have performed in your environment, and so the next *task*, named `Add instance to Inventory` places a *host* named `cookbook1` into a **host group** called `webservers`. And for this particular *host*, named `cookbook1`, in the group `webservers`, we are setting the variable that Ansible would use to access that instance as the floating IP address assigned: `(ansible_ssh_host={{ nova_cookbook.server.accessIPv4 }}`. As you can see, we are using our registered variable, `nova_cookbook`, to access some information that Ansible has stored about that instance. It is important that we use the floating IP because the private tenant network is not routable from our client, and therefore Ansible would not be able to connect to perform the Apache install.

The next *play* named `Wait for port 22 to be ready` basically has a *task* that waits for SSH to be running. This signals that the instance is ready for use, and therefore we are able to SSH into this to run further Ansible commands.

The last one is the *play* that has *tasks* that perform the install of Apache. As described earlier, we have placed an optional set of `pre_tasks` into this section to overcome the fact that Ubuntu 16.04 doesn't come with the necessary prerequisite Python packages needed for Ansible to run. We have also set an optional APT proxy here too, so feel free to remove and adjust this section according to the image and environment you are operating with.

The last set of *tasks* in this *play* basically ensure that Apache is installed and runs. Here you could then add in additional tasks to pull in Apache configuration data from GitHub, or install additional packages, thus completing the set up of this instance from a single Ansible command.

Using Ansible to orchestrate software installations across multiple instances

So far, we created playbooks that first launched an instance, and then we extended this in the previous recipe to subsequently install Apache onto the running instance. This recipe describes a playbook that can launch any number of instances, and install Apache onto each of those instances.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment that has Ansible installed.

How to do it...

We will extend the previous recipe's playbook to add flexibility to include a variable number of instances.

1. The basic structure was provided in the previous recipe, so the only **play** we need to adjust is the first one that launches the instances, called `Launch instances on OpenStack`. This complete *play* is shown as follows, where we introduce a variable called `count`, which we have set to `2`, and we also introduce the `with_sequence` section, which forms our loop that will execute that *task* the specified number of *count* times. Note that we also include the count value as part of the instance name:

```
- name: Launch instances on OpenStack
  hosts: localhost
  gather_facts: false

  vars:
    count: 2

  tasks:
    - name: Create ansible security group
      os_security_group:
        state: present
        name: ansible
        verify: false
    - name: Create a rule to allow SSH connections
      os_security_group_rule:
```

```
    security_group: ansible
    protocol: tcp
    port_range_min: 22
    port_range_max: 22
    remote_ip_prefix: 0.0.0.0/0
    verify: false
- name: Create webserver security group
  os_security_group:
    state: present
    name: webserver
    verify: false
- name: Create a rule to allow http connections
  os_security_group_rule:
    security_group: webserver
    protocol: tcp
    port_range_min: 80
    port_range_max: 80
    remote_ip_prefix: 0.0.0.0/0
    verify: false
- name: Deploy an instance
  os_server:
    state: present
    name: cookbook{{ item }}
    image: xenial-image
    key_name: demokey
    timeout: 200
    flavor: m1.tiny
    network: private-net
    verify: false
register: nova_cookbook
with_sequence:
    count={{ count }}

- name: Add instance to Inventory
  add_host: name="{{ item.server.name }}" groups=webservers
         ansible_ssh_host="{{ item.server.accessIPv4 }}"
with_items: "{{ nova_cookbook.results }}"
```

2. The next set of plays copy what was described in the previous recipe, such as waiting for the instance's SSH to be available and subsequently installing Apache, and is shown here for completeness:

```
- name: Wait for port 22 to be ready
  hosts: webservers
  gather_facts: False
  tasks:
    - local_action: wait_for port=22 host="{{ ansible_ssh_host }}"
search_regex=OpenSSH delay=10

- hosts: webservers
  remote_user: ubuntu
  become: yes
  gather_facts: no
```

The `pre_tasks:` section is optional. Your use will vary on any implicit restrictions imposed on the image or environment you are using. This was described in the previous recipe on its use here:

```
    pre_tasks:
      - name: Set APT proxy
        raw: echo "Acquire::http::Proxy
  \"http://192.168.1.20:3128\";" > /etc/apt/apt.conf
      - name: 'install python2'
        raw: sudo apt-get -y install python-simplejson

  tasks:
      - name: Ensure Apache is installed
        apt: name=apache2 state=latest
      - name: Ensure that Apache is started
        service: name=apache2 state=started
```

3. Assuming the file that you have created is called `multi-orchestrate-instances.yml`, you execute this with the following:

```
source openrc
ansible-playbook multi-orchestrate-instances.yml
```

This will bring back an output like the following. This produces more output than the other plays so far, so only the last part is shown:

```
TASK [wait_for]
*********************************************************************************
ok: [cookbook1 -> localhost]
ok: [cookbook2 -> localhost]

PLAY [webservers]
*********************************************************************************

TASK [Set APT proxy]
*********************************************************************************
changed: [cookbook2]
changed: [cookbook1]

TASK [install python2]
*********************************************************************************
changed: [cookbook1]
changed: [cookbook2]

TASK [Ensure Apache is installed]
*********************************************************************************
changed: [cookbook1]
changed: [cookbook2]

TASK [Ensure that Apache is started]
*********************************************************************************
ok: [cookbook1]
ok: [cookbook2]

PLAY RECAP
*********************************************************************************
cookbook1                  : ok=5    changed=3    unreachable=0    failed=0
cookbook2                  : ok=5    changed=3    unreachable=0    failed=0
localhost                  : ok=11   changed=2    unreachable=0    failed=0
```

How it works...

We have included a few extra items in this playbook that extends the previous playbook that installs Apache to a newly launched instance. These are described here:

```
vars:
   count: 2
```

We introduce a variable called `count`, to which we assign the value `2`. This variable is limited to the scope of this particular play (named `Launch instances on OpenStack`). This variable is used to form a loop, as indicated by this attribute assigned to the `os_server` call:

```
with_sequence:
          count={{ count }}
```

This basically states: run the `os_server` module task when `count = 1`, and when `count = 2`. As we are in a sequence, we have access to the value of `count`, in a variable called `item`. We use this to append to the `name` variable of the instance allowing us to end up with `cookbook1` and `cookbook2` with the following syntax:

```
name: cookbook{{ item }}
```

Using Ansible to fully orchestrate the creation of a web server and load balancer stack

The previous recipes launched instances into an existing environment, including existing networks, images and keys, for example. However, using Ansible for orchestration of OpenStack environments brings a full suite of modules that can be used to operate more than just Nova. For example, we can use Ansible to control Glance, Neutron, Cinder, and so on.

In this recipe, we only assume that a user is able to authenticate into a project. We don't assume that any networks exist, or even any images exist. We can get Ansible's view of the world to ensure that images and networks are present, and if not—create them.

Note that this recipe is intended to introduce you to the wonderful world of Ansible. The example is to show the creation of a stack from start to finish. Optimizing Ansible playbooks is beyond the scope of this book.

Getting ready

Ensure that you are logged on to a correctly configured OpenStack client and can access the OpenStack environment that has Ansible installed.

How to do it...

Carry out the following steps to launch an environment that ensures an image is available for use, sets the correct security groups, creates new networks and routers, and finally installs Apache onto two web servers:

1. We can assume a blank OpenStack project, but we want to ensure that we have the appropriate running instances and services. We first include tasks as part of our `Create OpenStack Cloud Environment` play that first downloads an Ubuntu 16.04 image and then loads into OpenStack. Start off by creating the `full-stack.yml` file with the following contents:

```
- name: Create OpenStack Cloud Environment
  hosts: localhost
```

```
  gather_facts: false

  vars:
    webserver_count: 2

  tasks:
    - name: Download Ubuntu 16.04 Xenial
      get_url:
        url: http://releases.ubuntu.com/16.04/ubuntu-16.04.3-
server-amd64.img
        dest: /tmp/ubuntu-16.04.img

    - name: Ensure Ubuntu 16.04 Xenial Image Exists
      os_image:
        name: xenial-image
        container_format: bare
        disk_format: qcow2
        state: present
        filename: /tmp/ubuntu-16.04.img
        verify: false
```

2. Next, we will create the *private tenant network and router*. We only assume that a shared provider network already exists. In this instance, we assume this provider network—that provides floating IP addresses—is called `GATEWAY_NET`. Carry on editing the file to include the following *tasks* as part of the same `Create OpenStack Cloud Environment` play:

```
    - name: Create the cookbook network
      os_network:
        state: present
        name: cookbook_network
        external: false
        shared: false
        verify: false
      register: cookbook_network

    - name: Create the test subnet
      os_subnet:
        state: present
        network_name: "{{ cookbook_network.id }}"
        name: cookbook_subnet
        ip_version: 4
        cidr: 192.168.0.0/24
        gateway_ip: 192.168.0.1
        enable_dhcp: yes
```

```
      dns_nameservers:
        - 192.168.1.20
      verify: false
    register: cookbook_subnet

- name: Create the test router
  os_router:
    state: present
    name: cookbook_router
    network: GATEWAY_NET
    external_fixed_ips:
      - subnet: GATEWAY_SUBNET
    interfaces:
      - cookbook_subnet
    verify: false
```

Optional (if using the Vagrant environment). Currently, the `os_router` module is unable to insert static routes into a router, so if you have a requirement to utilize static routes into your Ansible controlled playbooks, a workaround is to execute an `openstack` command instead as shown as follows. If you are using the *Vagrant* lab that accompanies this book, you may need to provide static routes to your router in order for traffic to flow from the physical host to the VirtualBox/VMware environment. If you require this, add in the following *task* (edit to suit your environment). In this instance, the physical host running the Vagrant environment has an IP of 192.168.100.1 and allows traffic to flow from the physical host to the instances that have a floating IP from a `192.168.100.0/24` provider network. This provider network here is the `GATEWAY_NET` referred to in this example:

```
      - name: Insert routes into router
        command: openstack router set --route destination
=192.168.1.0/24,gateway=192.168.100.1 cookbook_router
      register: cookbook_router_route
```

3. Next, we configure the security groups. Remember that Ansible uses SSH to connect to servers, and by default, cloud images prevent any incoming connections. So one of the rules should be for allowing incoming SSH connections. We also need to configure security group rules for the intended service that is to run on the instances. In this case, we're running Apache and HAProxy on TCP Port 80, so that also needs to be set up here:

```
- name: Create ansible security group
  os_security_group:
    state: present
    name: ansible
    verify: false
```

```
- name: Create rule to allow SSH connections
  os_security_group_rule:
    security_group: ansible
    protocol: tcp
    port_range_min: 22
    port_range_max: 22
    remote_ip_prefix: 0.0.0.0/0
    verify: false

- name: Create webserver security group
  os_security_group:
    state: present
    name: webserver
    verify: false
- name: Create rule to allow http connections
  os_security_group_rule:
    security_group: webserver
    protocol: tcp
    port_range_min: 80
    port_range_max: 80
    remote_ip_prefix: 0.0.0.0/0
    verify: false
```

As we are running Ansible from one specific host, we can further secure the Ansible SSH security group rule by limiting the access from a single IP address, and not a general 0.0.0.0/0 range.

4. We can now launch the instances. Carry on editing this file to add in the tasks to launch multiple web servers and a single HAProxy instance as follows. Note that the chosen network, image, and security groups match what we created in the preceding tasks:

```
- name: Deploy Webserver Instances
  os_server:
    state: present
    name: webserver{{ item }}
    image: xenial-image
    key_name: demokey
    timeout: 200
    flavor: m1.tiny
    network: cookbook_network
    security_groups: default,ansible,webserver
    verify: false
  register: nova_webservers
```

```
    with_sequence:
      count={{ webserver_count }}

  - name: Add webservers to Inventory
    add_host: name="{{ item.server.name }}" groups=webservers
          ansible_ssh_host="{{ item.server.accessIPv4 }}"
    with_items: "{{ nova_webservers.results }}"

  - name: Deploy HAProxy Instance
    os_server:
      state: present
      name: haproxy
      image: xenial-image
      key_name: demokey
      timeout: 200
      flavor: m1.tiny
      network: cookbook_network
      security_groups: default,ansible,webserver
      verify: false
    register: nova_haproxy
    with_sequence:
      count=1

  - name: Add HAProxy to Inventory
    add_host: name="{{ item.server.name }}" groups=haproxy
          ansible_ssh_host="{{ item.server.accessIPv4 }}"
    with_items: "{{ nova_haproxy.results }}"
```

5. As we have seen, we need to wait for SSH to be available before Ansible should continue, so we add in a *wait* until this is so. Note that we apply this play to both the `webservers` and `haproxy` *hosts* groups:

```
- name: Wait for port 22 to be ready
  hosts: webservers:haproxy
  gather_facts: False
  tasks:
    - local_action: wait_for port=22 host="{{ ansible_ssh_host }}"
search_regex=OpenSSH delay=10
```

6. With the instances up and running, the final tasks are concerned with installation and configuration of the services that run on the instances. We will first install Apache on our web servers. This play is applied to our `webservers` hosts, so these tasks will run for each of them:

```
- name: Configure Web Servers
  hosts: webservers
```

```
  remote_user: ubuntu
  become: yes
  gather_facts: False

pre_tasks:
    - name: Set APT proxy
      raw: echo "Acquire::http::Proxy
\"http://192.168.1.20:3128\";" > /etc/apt/apt.conf
    - name: 'install python2'
      raw: sudo apt-get -y install python-simplejson

tasks:
    - name: Ensure Apache is installed
      apt: name=apache2 state=latest
    - name: Ensure that Apache is started
      service: name=apache2 state=started
```

7. As our plays have `gather_facts` set to `False` (because the *image* we're using doesn't have the Python libraries expected to run Ansible out of the box, which would mean that the Ansible task that gathers facts about a running instance would immediately fail), we will launch a separate task in a new play that just populates the Ansible inventory of facts, knowing we installed the prerequisite Python 2 libraries in the preceding play. We need this information to populate the HAProxy configuration file later on:

```
- name: Gathering facts about webservers
  hosts: webservers
  remote_user: ubuntu
  become: yes
  tasks:
    - name: Gathering facts
      setup:
```

8. We can now install and configure HAProxy. This gets applied to our *haproxy* group of hosts (of which there is only one server, named *haproxy*). As part of this play, we will reference a HAProxy configuration file template. We will create this in the next step.

Ansible provides pre-defined, and extensively tested roles that can be used for the installation of software. The guide below is for example purposes only. In reality, you would utilize roles from `https://galaxy.ansible.com/`.

Carry on building out this `full-stack.yml` playbook file with the following contents:

```
- name: Configure HAProxy
  hosts: haproxy
  remote_user: ubuntu
  become: yes
  gather_facts: False

  pre_tasks:
    - name: Set APT proxy
      raw: echo "Acquire::http::Proxy
\"http://192.168.1.20:3128\";" > /etc/apt/apt.conf
    - name: 'install python2'
      raw: sudo apt-get -y install python-simplejson

  tasks:
    - name: Update apt cache
      apt: update_cache=yes cache_valid_time=3600

    - name: Install haproxy
      apt: name=haproxy state=present

    - name: Enable init script
      replace: dest='/etc/default/haproxy'
            regexp='ENABLED=0'
            replace='ENABLED=1'

    - name: Update HAProxy config
      template: src=templates/haproxy.cfg.j2
            dest=/etc/haproxy/haproxy.cfg
      notify:
        - restart haproxy

  handlers:
    - name: restart haproxy
      service: name=haproxy state=restarted
```

9. Before we can continue executing our playbook, we need to create the HAProxy configuration template file as specified in the previous step. The configuration pointed to a template file named `haproxy.cfg.j2` in the `templates` directory. Create this directory from the current working directory where you are editing the `full-stack.yml` file:

```
mkdir templates/
vi templates/haproxy.cfg.j2
```

10. Populate the `haproxy.cfg.j2` file with the following contents:

```
global
    log 127.0.0.1 local0 notice
    maxconn 2000
    user haproxy
    group haproxy
    daemon

defaults
    log  global
    mode  http
    option  httplog
    option  dontlognull
    retries 3
    option redispatch
    timeout connect  5000
    timeout client  10000
    timeout server  10000

listen {{haproxy_app_name}}
    bind *:80
    mode {{haproxy_mode}}
    stats {{haproxy_enable_stats}}
    {% if haproxy_enable_stats == 'enable' %}
    stats uri /haproxy?stats
    stats realm Strictly\ Private
   {% endif %}
    balance {{haproxy_algorithm}}
    option httpclose
    option forwardfor
    {% for host in groups['webservers'] %}
    server {{ hostvars[host].inventory_hostname }} {{
hostvars[host]['ansible_all_ipv4_addresses'][0] }} check
    {% endfor %}
```

The `server {{ hostvars[host]...check` line is all on a single line.

11. The `templates/haproxy.cfg.j2` file also refers to some variables that we have not yet declared to Ansible, such as `haproxy_app_name` and `haproxy_algorithm`. These are in a `group_var` file specific to our `haproxy` group. To create this group variable file, we need to create a `group_vars/haproxy` directory, with a file called `main.yml` that lists these variables. From the same directory as our `full-stack.yml` file, carry out the following steps:

```
mkdir -p group_vars/haproxy
vi group_vars/haproxy/main.yml
```

12. Populate the `group_vars/haproxy/main.yml` file with the following contents:

```
---
haproxy_app_name: myapp
haproxy_mode: http
haproxy_enable_stats: enable
haproxy_algorithm: roundrobin
```

13. We are now ready to run our `full-stack.yml` playbook to create our environment similar to the one described by the Heat example. Execute the following commands:

```
source openrc
ansible-playbook full-stack.yml
```

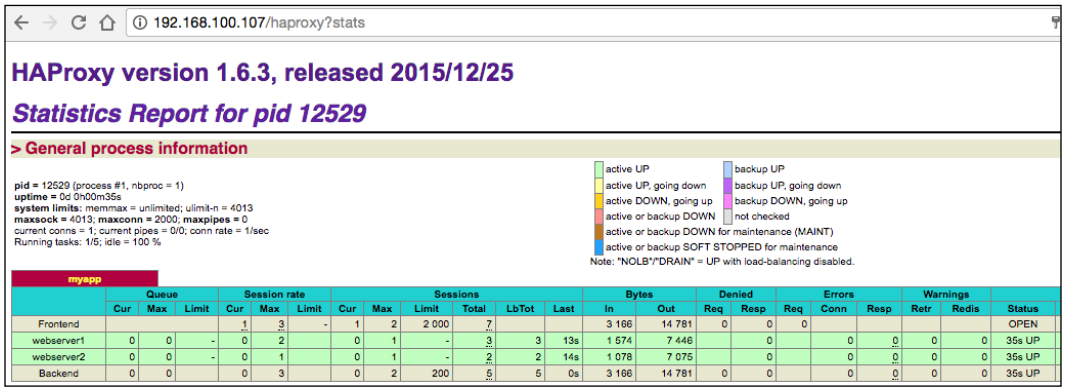
This will produce an output similar to the following. Due to this being a longer playbook, only the last part is shown:

```
PLAY [Configure HAProxy] *******************************************************

TASK [Set APT proxy] ***********************************************************
changed: [haproxy]

TASK [install python2] *********************************************************
changed: [haproxy]

TASK [Update apt cache] ********************************************************
ok: [haproxy]

TASK [Install haproxy] *********************************************************
changed: [haproxy]

TASK [Enable init script] ******************************************************
ok: [haproxy]

TASK [Update HAProxy config] ***************************************************
changed: [haproxy]

RUNNING HANDLER [restart haproxy] **********************************************
changed: [haproxy]

PLAY RECAP *********************************************************************
haproxy                    : ok=8    changed=5    unreachable=0    failed=0
localhost                  : ok=13   changed=2    unreachable=0    failed=0
webserver1                 : ok=7    changed=2    unreachable=0    failed=0
webserver2                 : ok=7    changed=2    unreachable=0    failed=0
```

We are also able to verify that these instances are running, and the addresses it has assigned by viewing an `openstack server list` output:

```
+--------------------------------------+-----------+--------+-----------------------------------------------------+--------------+--------+
| ID                                   | Name      | Status | Networks                                            | Image        | Flavor |
+--------------------------------------+-----------+--------+-----------------------------------------------------+--------------+--------+
| 9c27666d-462f-42d4-a896-c7b5d3caa29b | haproxy   | ACTIVE | cookbook_network=192.168.0.9, 192.168.100.107       | xenial-image | m1.tiny |
| 320b57e6-b449-4f16-b36a-c67f522d98b5 | webserver2 | ACTIVE | cookbook_network=192.168.0.3, 192.168.100.105       | xenial-image | m1.tiny |
| e7147424-2450-4959-98b3-1bd26e874921 | webserver1 | ACTIVE | cookbook_network=192.168.0.6, 192.168.100.101       | xenial-image | m1.tiny |
+--------------------------------------+-----------+--------+-----------------------------------------------------+--------------+--------+
```

14. Finally, we can test our setup by visiting the HAProxy server floating IP address, as shown here. Here we can view the HAProxy stats:



How it works...

What we did here was to methodically build out a playbook, called `full-stack.yml`, that carries out the following steps:

- Download and install Ubuntu 16.04 image if necessary
- Create our web server and ansible SSH security groups
- Create a private tenant network, router and configure the router
- Launch two web server instances and one HAProxy instance
- For each web server, install Apache
- For the HAProxy server, install HAProxy and configure the configuration file, auto-populated with information that Ansible knows about from launching the web servers (such as what IP address was assigned to each of them)

In this recipe, we introduced a few notable items: Ansible *facts*, the *Jinja2* configuration template file (the `haproxy.cfg.j2` file), and `group_vars`.

The Ansible facts and HAProxy `group_var` variables were used to populate the HAProxy configuration file called `templates/haproxy.cfg.j2`. This file mostly looks like a normal `haproxy.cfg` file, but has elements that are applicable to Jinja2, which Ansible interprets. Of particular interest is the `haproxy.cfg` file that has the load balance pool member lines, that ordinarily look like the following basic construct:

```
server webserver1 192.168.0.6 check
server webserver2 192.168.0.3 check
```

When we launch our instances into OpenStack, we have no idea what IP address they will be assigned, however Ansible does this with its *fact gathering*. If we take a look at the same line in our template, we get the following:

```
{% for host in groups['webservers'] %}
server {{ hostvars[host].inventory_hostname }} {{ hostvars[host]
['ansible_all_ipv4_addresses'][0] }} check
{% endfor %}
```

This line has some static text at the beginning and end, denoted by `server` and `check`. They're the same `server` and `check` text, as we can see in our final output.

The magic of the template and Ansible is what we can do with the loop that surrounds this line and `hostvars` we can access. The loop says:

For each of the `hosts` in the `webservers` *group* (recall that we registered the web server instances into this specific inventory group), get the value of `hostvars[host].inventory_hostname` and `hostvars[host]['ansible_all_ipv4_addresses'][0]` (from the **gathered facts**). This last variable takes the first entry in the `ansible_all_ipv4_addresses` dict, which is our internal IP of our instance.

The result is the output shown on the previous page, which lists the web servers that HAProxy can access in the load balance pool.

We aren't restricted to the variables that Ansible has gathered. We have specified a `group_var` variable file that specifies the following:

```
---
haproxy_app_name: myapp
haproxy_mode: http
haproxy_enable_stats: enable
haproxy_algorithm: roundrobin
```

This is referenced directly in the template file in the following places. This allows us to add some static elements that are user-configurable, but allows us to maintain a flexible, environment agnostic set of playbooks.

```
listen {{haproxy_app_name}}
   bind *:80
   mode {{haproxy_mode}}
   stats {{haproxy_enable_stats}}
   {% if haproxy_enable_stats == 'enable' %}
   stats uri /haproxy?stats
   stats realm Strictly\ Private
   {% endif %}
   balance {{haproxy_algorithm}}
```

10
Using OpenStack Dashboard

In this chapter, we will cover the following topics:

- Introduction – OpenStack Dashboard
- Using OpenStack Dashboard for key management
- Using OpenStack Dashboard to manage Neutron networks and routers
- Using OpenStack Dashboard for security group management
- Using OpenStack Dashboard to launch instances
- Using OpenStack Dashboard to delete instances
- Using OpenStack Dashboard to add new projects
- Using OpenStack Dashboard for user management
- Using OpenStack Dashboard with LBaaS
- Using OpenStack Dashboard with OpenStack Orchestration

Introduction – OpenStack Dashboard

Managing our OpenStack environment through a command-line interface allows us to have complete control of our cloud environment, but having a web-based interface that operators and administrators can use to manage their environments and instances makes this process easier. The OpenStack Dashboard, known as **Horizon**, provides a graphical, web-based, user interface. Horizon is a web service that runs from an Apache installation, using Python's **Web Service Gateway Interface** (**WSGI**) and Django, a rapid development web framework.

With the OpenStack Dashboard installed, we can manage all the core components of our OpenStack environment. If you are using OpenStack Ansible playbooks, Horizon gets installed using the `os-horizon-install.yml` playbook.

Using OpenStack Dashboard for key management

SSH key pairs allow users to connect to their Linux instances without requiring passwords, and it is the default access mechanism for almost all Linux images that you will use for OpenStack. Users can manage their own key pairs through the OpenStack Dashboard. Usually, this is the first task a new user has to do when given access to our OpenStack environment.

Getting ready

Load a web browser, point it to our OpenStack Dashboard address at `http://192.168.100.117/`, and log in as a user in the `default` domain, such as the `developer` user, created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client*, with the `cookbook4` password. The URL for our dashboard is same as the public load balancer IP as discussed in *Chapter 1*, *Installing OpenStack with Ansible*. If you need to find out at what URL your Horizon is, use public IP from the OpenStack catalog list as described in *Chapter 3*, *Keystone – OpenStack Identity Service*.

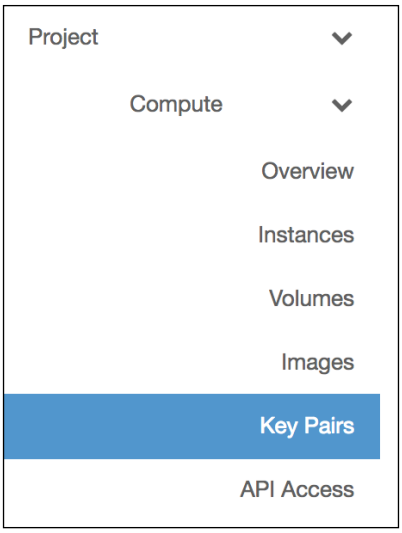
How to do it...

Management of the logged-in user's key pairs is achieved with the steps discussed in the following sections:

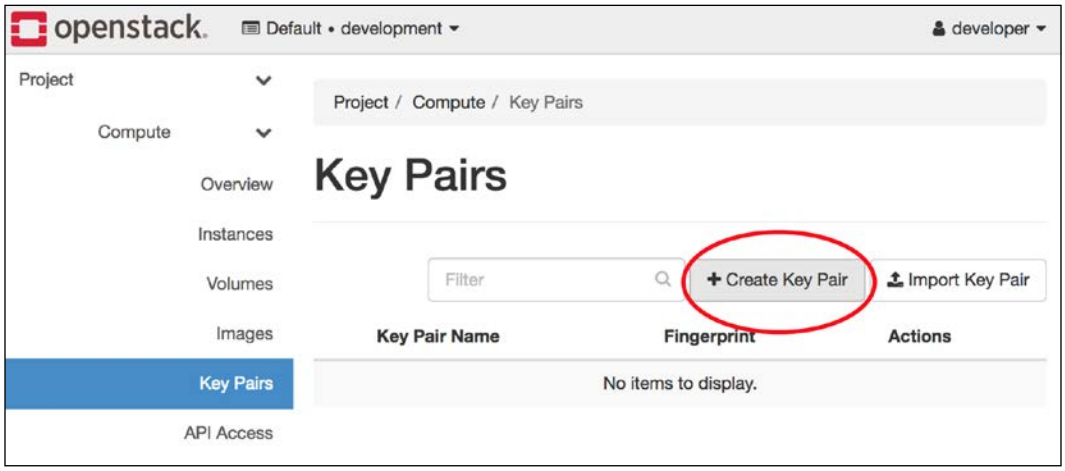
Adding key pairs

Key pairs can be added by performing the following steps:

1. A new key pair can be added to our system by clicking on the **Key Pairs** tab in the **Project** | **Compute** section:

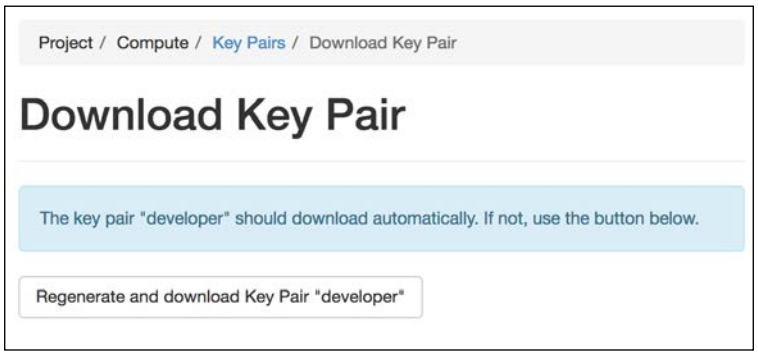


2. We will now see a screen allowing key pair management. In the right-hand corner of the screen, there is a **Create Key Pair** button. Click on this button to create a new key pair:



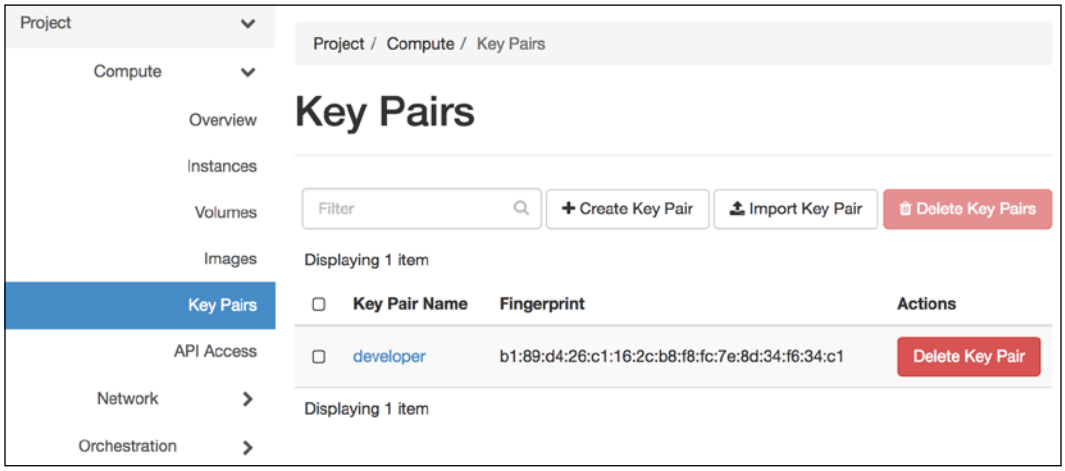
3. On the **Create Key Pair** screen, type in a meaningful name (for example, `developer`) ensuring that there are no spaces in the name, and then, click on the **Create Key Pair** button:

4. Once the new key pair is created, the private key portion of our key pair will automatically download. If not, click on the **Regenerate and download Key Pair "developer"** button:



A private SSH key cannot be recreated, so keep this safe and store it safely and appropriately on the filesystem.

5. Click on the **Key Pairs** link to return to our list of key pairs. We will now see the newly created key pair listed. When launching instances, we can select this new key pair and gain access to it only using the private key that we have stored locally:

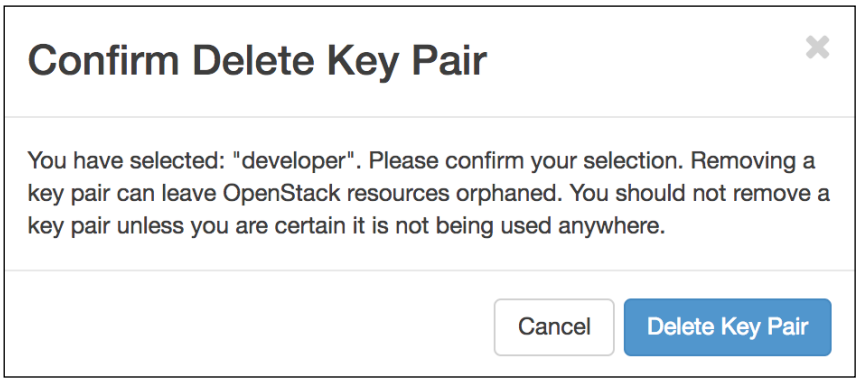


Deleting key pairs

Key pairs can be deleted by performing the following steps:

1. When key pairs are no longer required, we can delete them from our OpenStack environment. To do so, click on the **Key Pairs** tab on the left of our screen.
2. We will then be presented with a screen allowing access to key pair management. Under **Key Pairs**, there will be a list of key pairs that we can use to access our instances. To delete a key pair from our system, click on the **Delete Key Pair** button for the key pair that we want to delete:

3. We will be presented with a confirmation dialog box:



Once we click on the **Delete Key Pair** button, the key pair will be deleted.

Importing key pairs

If you have your own key pairs that you use to access other systems, these can be imported into our OpenStack environment so that you can continue to use them for accessing instances within our OpenStack Compute environment. To import key pairs, perform the following steps:

1. We can import key pairs that have been created in our traditional Linux-based environments into our OpenStack setup. If you don't have one already, run the following from your Linux-based or other Unix-based host:

   ```
   ssh-keygen -t rsa -f cookbook.key
   ```

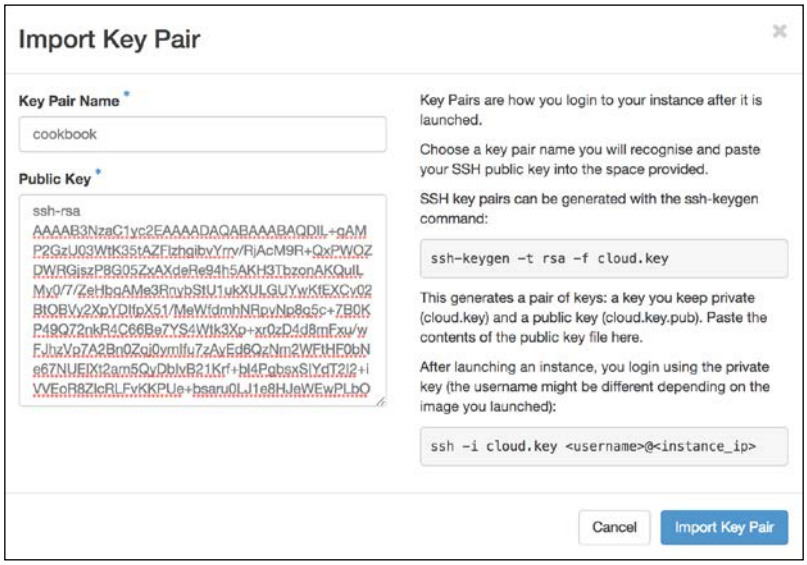
2. This will produce the following two files on our client:

   `cookbook.key`

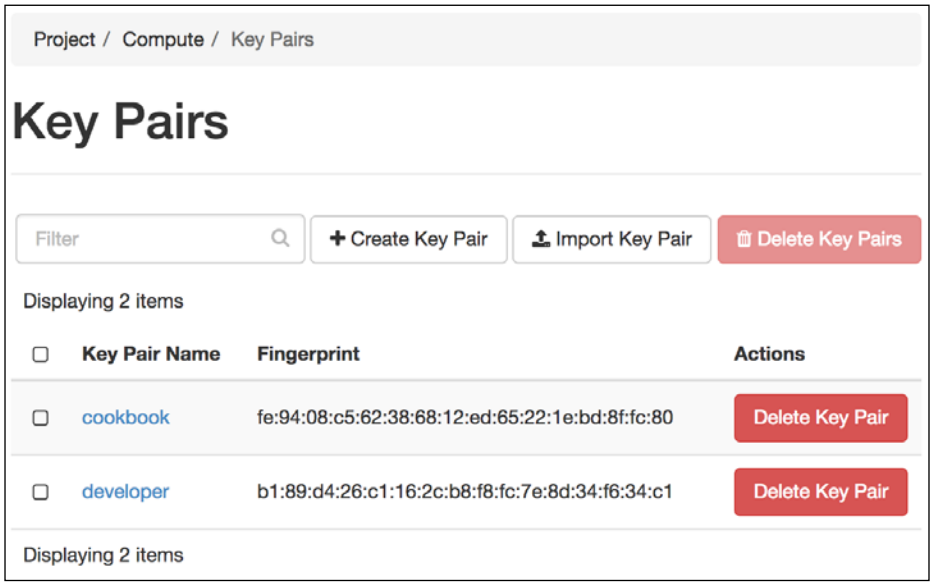
   `cookbook.key.pub`

3. The `cookbook.key` file is our private key and has to be protected as it is the only key that matches the public portion of the `cookbook.key.pub` key pair.
4. We can import this public key to use in our OpenStack environment so that when an instance is launched, the public key is inserted into our running instance. To import the public key, ensure that you're at the **Access & Security** screen, and then in **Key pairs**, click on the **Import Key Pair** button:

5. We are presented with a screen that asks us to name our key pair and paste the contents of our public key. So name the key pair, and then copy and paste the contents of the public key into the space—for example, the contents of `cookbook.key.pub`. Once entered, click on the **Import Key Pair** button:



Once completed, we see the list of key pairs available for that user, including our imported key pair:



How it works...

Key pair management is important as it provides a consistent and secure approach for accessing our running instances. Allowing the user to create, delete, and import key pairs to use within their projects enables them to create more secure systems.

The OpenStack Dashboard allows a user to create key pairs easily. The user must ensure, though, that the private key that they download, is kept secure.

While deleting a key pair is simple, the user must remember that deleting a private key that is associated with running instances will remove access to the running system. Deleting a key pair from the dashboard will not delete keys from instances that are already running. Every key pair created is unique regardless of the name. The name is simply a label, but the unique fingerprint of the key is required and cannot be recreated.

Importing key pairs has the advantage that we can use our existing secure key pairs that we have been using outside of OpenStack, within our new private cloud environment. This provides a consistent user experience when moving from one environment to another.

Using OpenStack Dashboard to manage Neutron networks and routers

The OpenStack Dashboard has the ability to view, create, and edit Neutron networks, which makes managing complex software-defined networks much easier. Certain functions, such as creating shared networks and provider routers, require a user to be logged in to the OpenStack Dashboard as a user with admin privileges, but any user can create private networks. To help with managing complex software-defined networks, the OpenStack Dashboard provides means to graphically view and update the network topography.

Getting ready

Load a web browser, point it to our OpenStack Dashboard address at `http://192.168.100.117/`, and log in as a user in the `default` domain, such as the `developer` user, created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client*, with the `cookbook4` password. The URL for our dashboard is same as the public load balancer IP as discussed in *Chapter 1*, *Installing OpenStack with Ansible*. If you need to find out at what URL your Horizon is, use public IP from the OpenStack catalog list as described in *Chapter 3*, *Keystone – OpenStack Identity Service*.

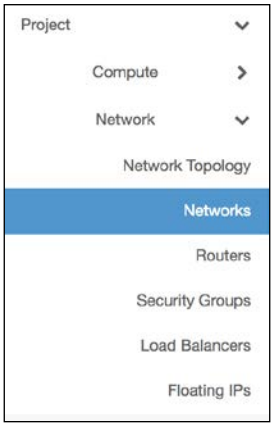
How to do it

This recipe will walk you through creating and deleting networks using the OpenStack Dashboard:

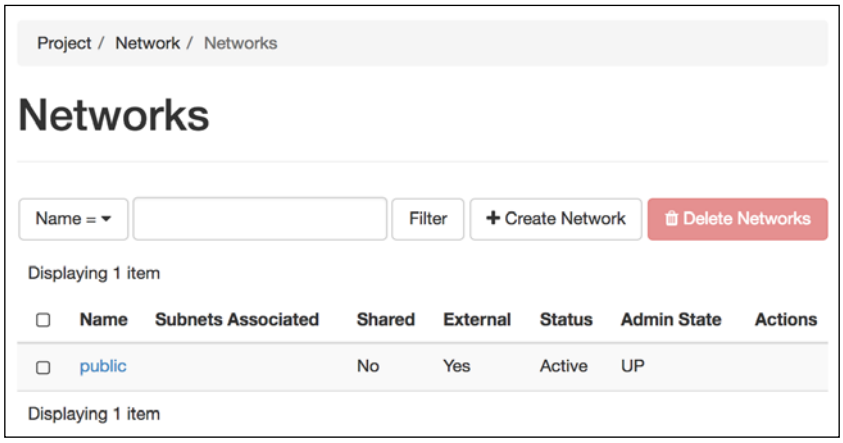
Creating networks

To create a private network for the logged in user, carry out the following steps:

1. To manage networks within our OpenStack Dashboard, select the **Networks** tab as shown in the following screenshot:

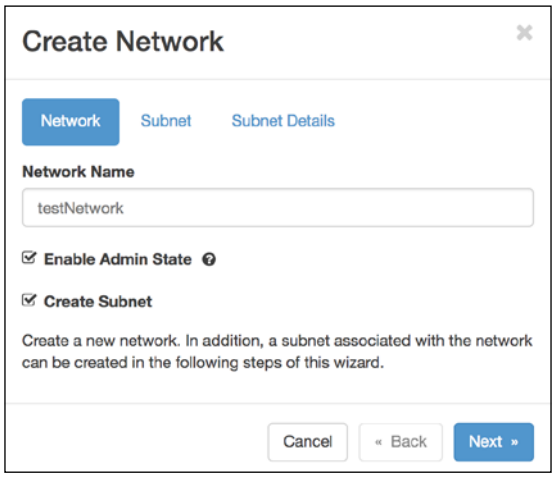


2. When this has been selected, we will be presented with a list of networks that we can assign to our instances:

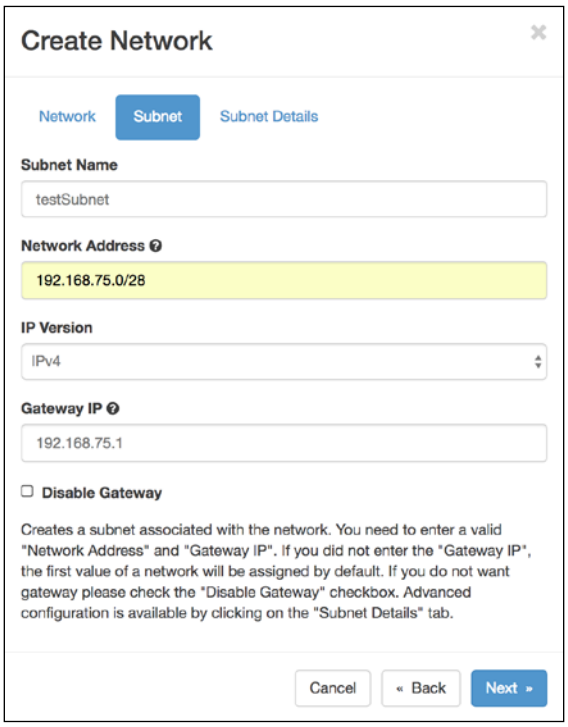


3. To create a new network, click on the **Create Network** button.

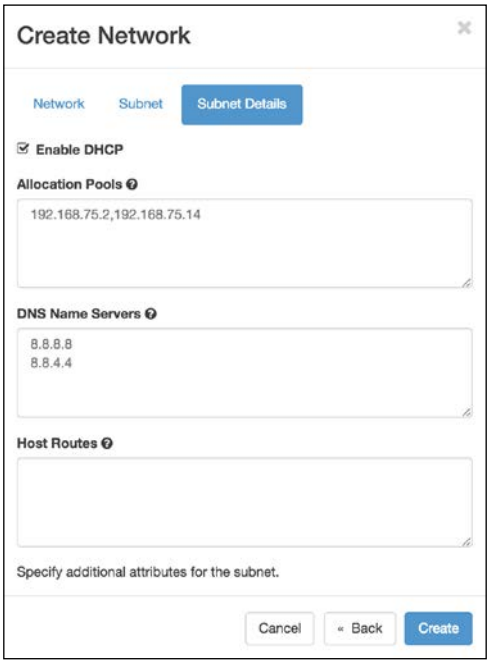
4. We will be presented with a dialog box that first asks us to name our network:



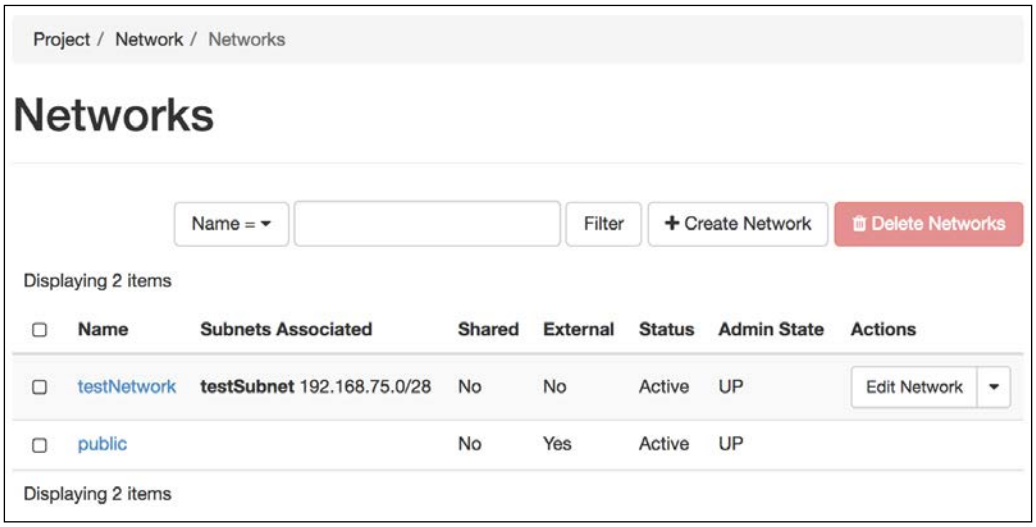
5. After choosing a name, and keeping the admin state set to enabled (which means our network will be on and available for instances to connect to) we then assign a subnet to it by selecting the **Subnet** tab or clicking on the **Next** button:



6. After filling in details for our subnet, we select the **Subnet Details** tab that allows us to configure details such as DHCP range, DNS, and any additional routes that we want when a user chooses that network:



7. After filling in all the details, clicking on the **Create** button makes this network available to the users of our project and takes us back to the list of available networks:

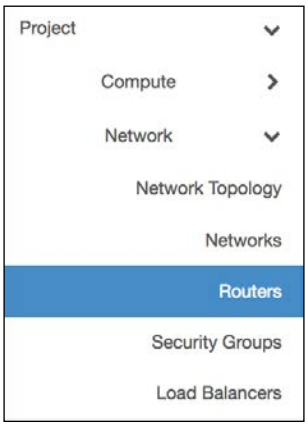


Creating routers

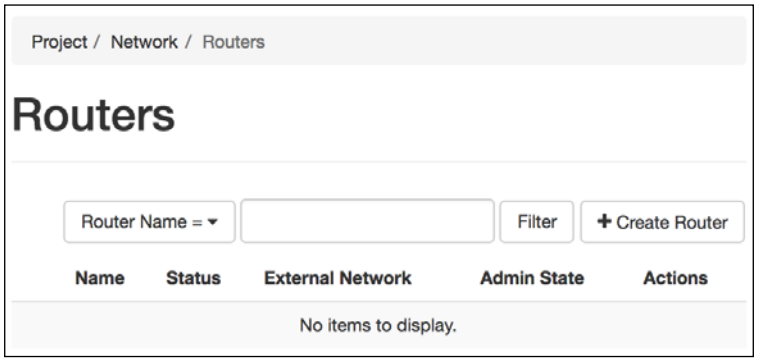
Once a private network is added, it probably needs to connect to the outside world. If you have a public network, simply create a router between your private network and the public network.

To manage routers, execute the following steps:

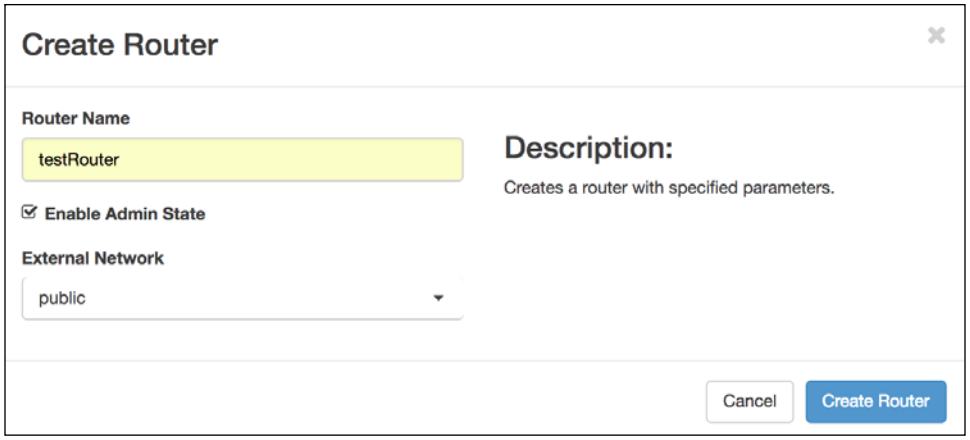
1. Select the **Routers** tab, as shown in the following screenshot:



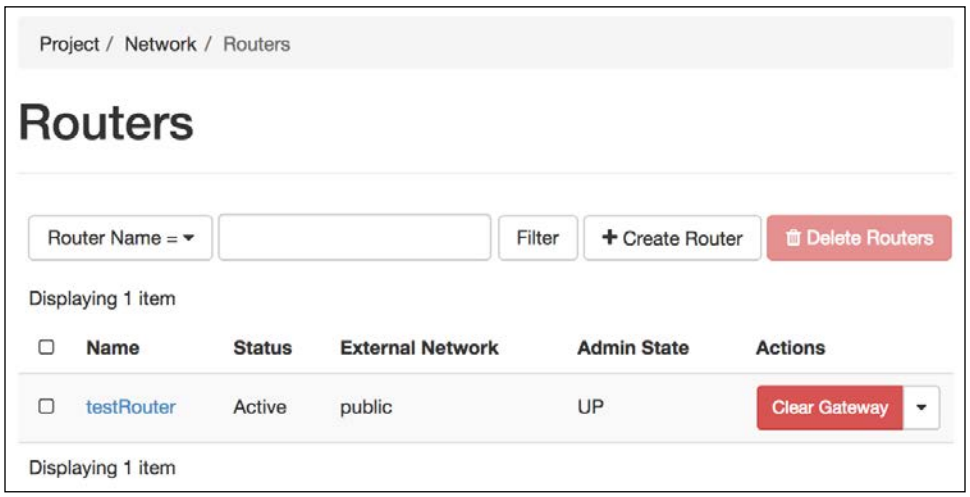
This will show you available routers for logged in users:



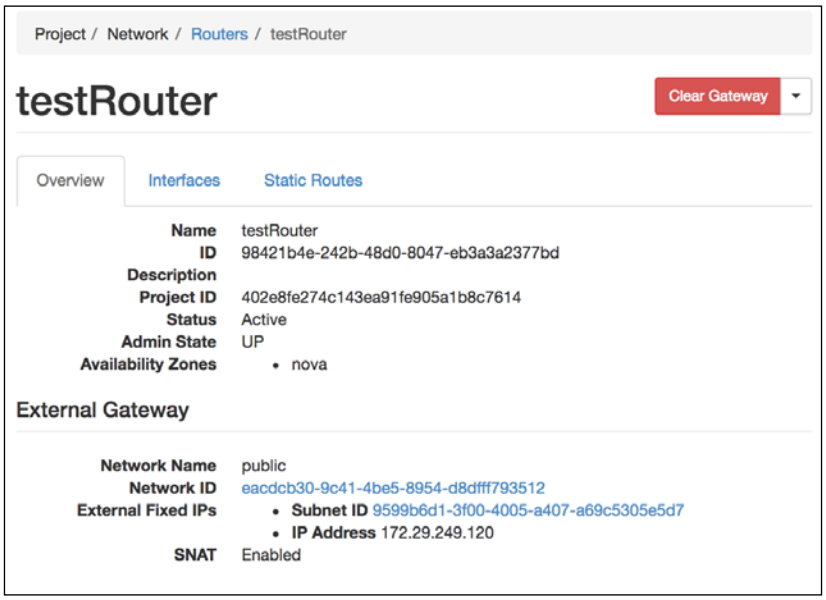
2. Click on the **Create Router** button to add a new router:



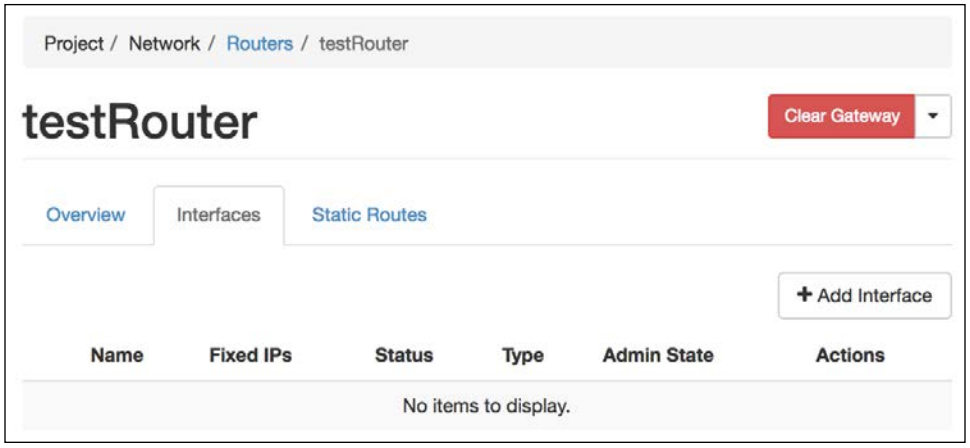
3. Enter a router name and select external network. In our case, we will select the **public** network. Click on the **Create Router** button. A list of available routers will be shown, including the newly created router:



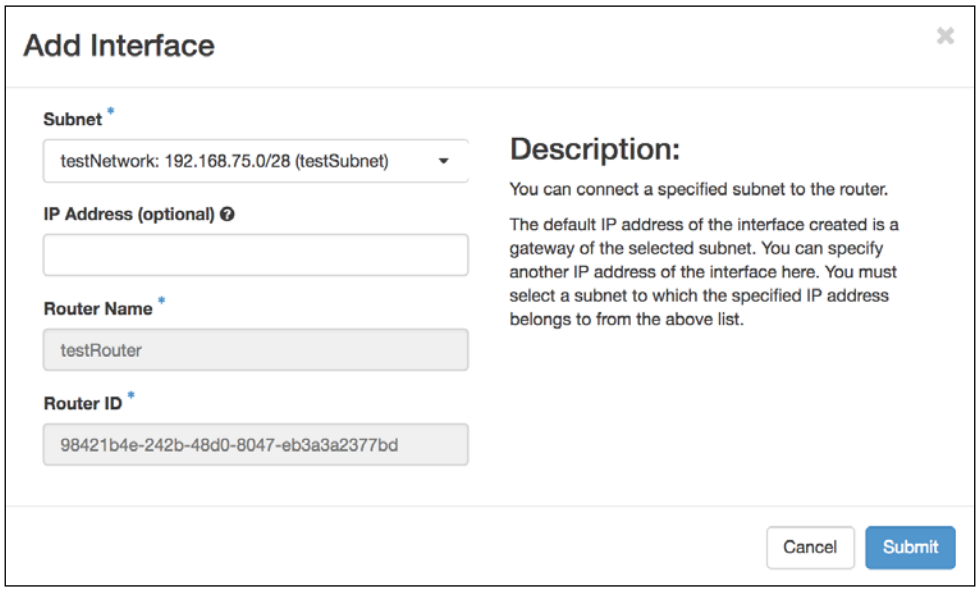
4. Click on the newly created router's name to view the details:



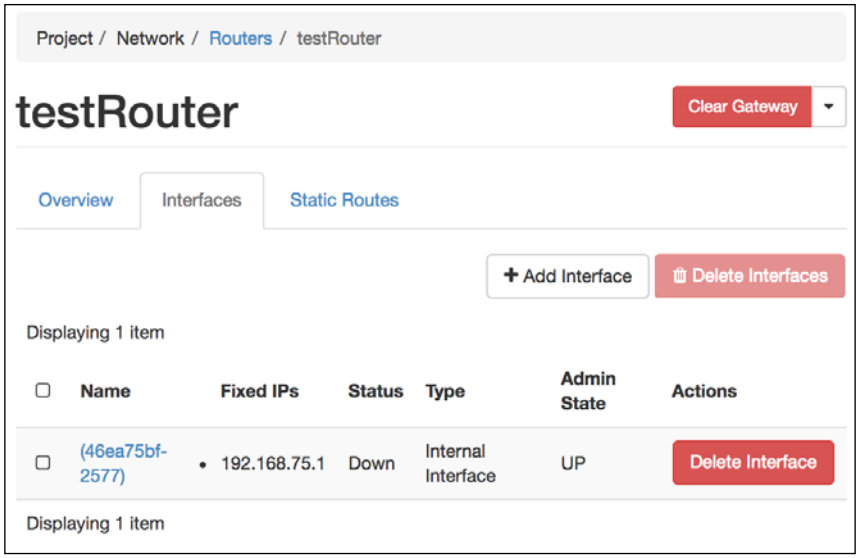
5. This router is still missing an interface. To add the private network as an available interface, select the **Interfaces** tab:



6. Click on the **Add Interface** button to add the missing interface:



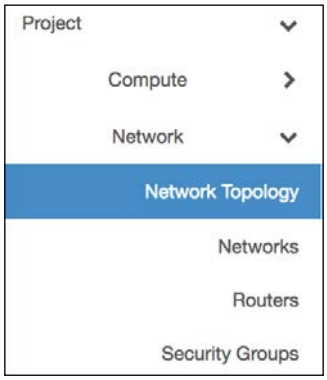
7. Select the subnet that needs to be connected to the public network via our router, and click on the **Submit** button. Now our router has an additional interface:



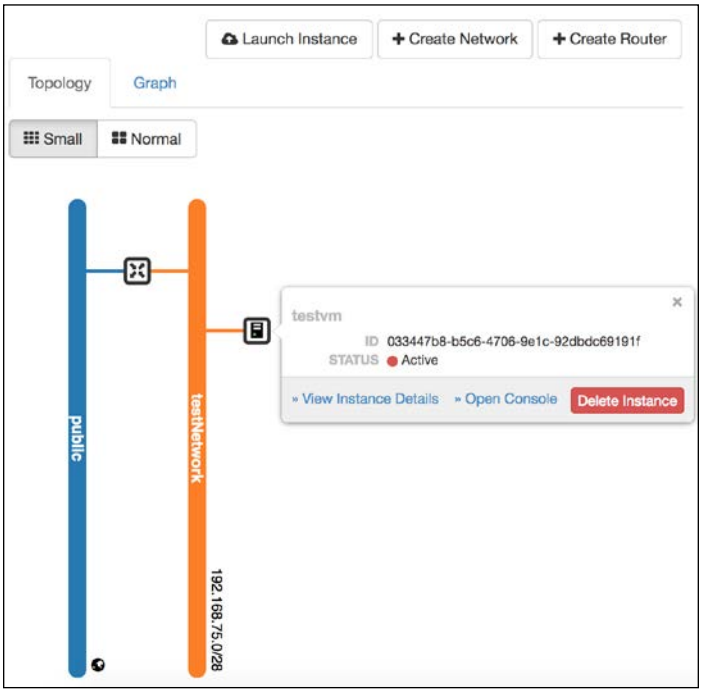
Viewing networks

The OpenStack Dashboard gives users and administrators the ability to view the topology of our environment. To view the network topology, carry out the following steps:

1. To manage networks within our OpenStack Dashboard, select the **Network Topology** tab, as shown in the following screenshot:



2. Clicking on the **Network Topology** tab gives a rich interface that gives an overview of our networks and instances attached to them as follows:

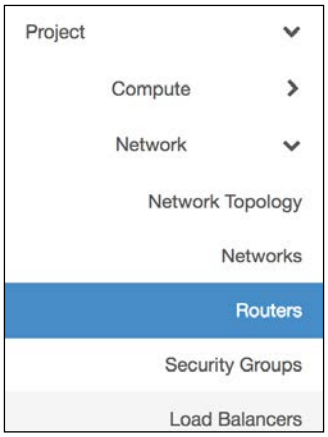


3. From this interface, we can click on various parts of this interface, such as the networks (which takes us to the manage network interface) and the instances (which takes us to the instances interface), as well as to create new networks, routers, and launch new instances.

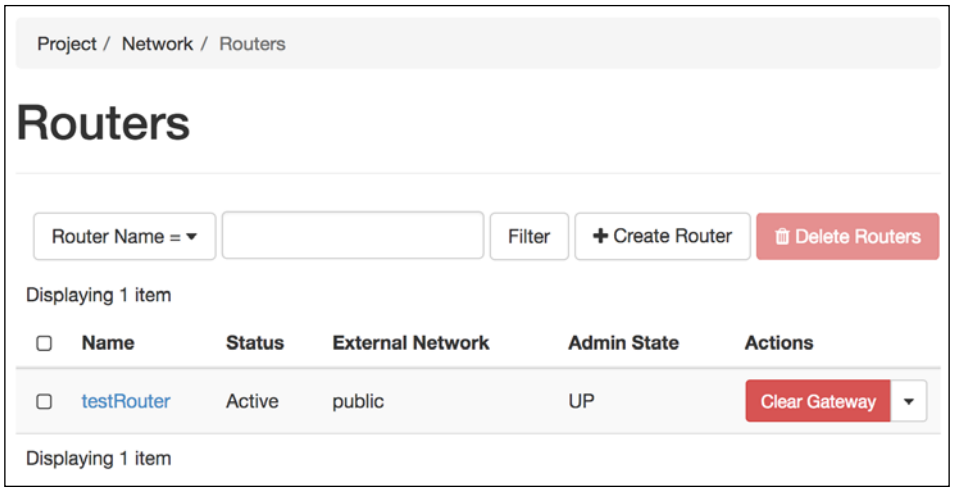
Deleting routers

To delete a private router for a logged in user, carry out the following steps:

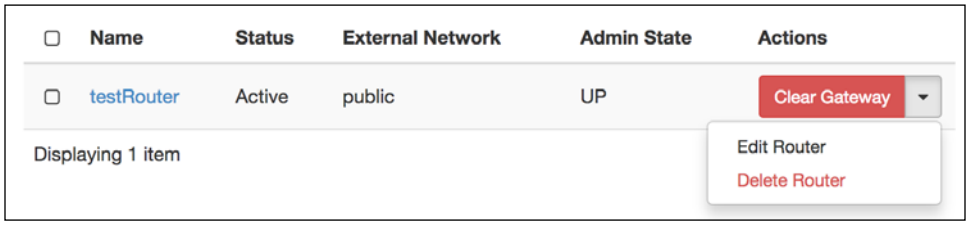
1. To manage routers within our OpenStack Dashboard, select the **Routers** tab, as shown in the following screenshot:



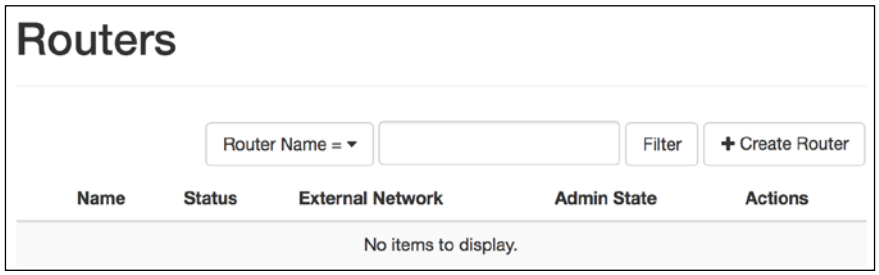
2. When this has been selected, we will see all logged in user's the available routers:



3. To delete a router, click on the drop-down action menu item next to the **Clear Gateway** button and click on the **Delete Router** option:



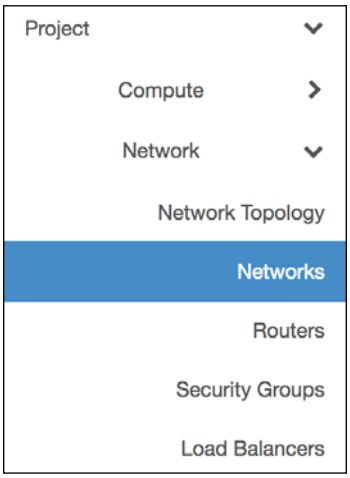
4. Once the router is deleted, you will be presented with a list of available remaining routers:



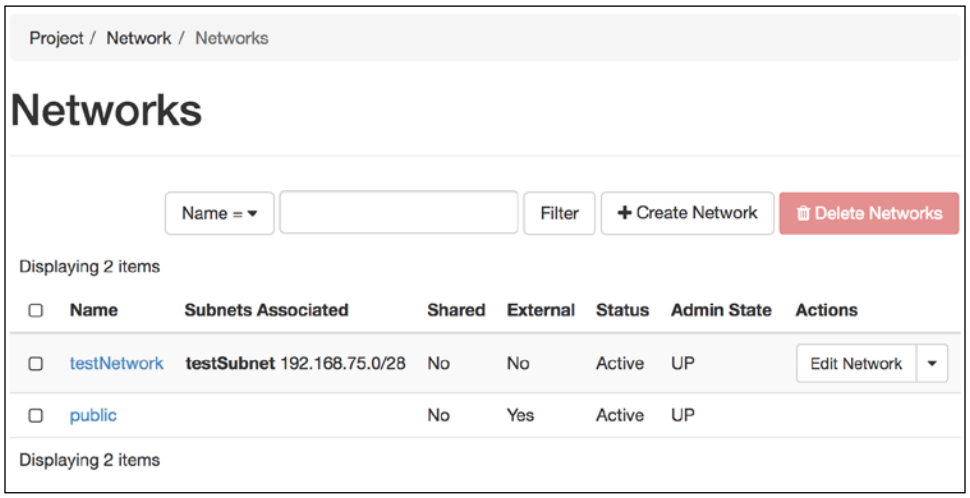
Deleting networks

To delete a private network for a logged in user, carry out the following steps:

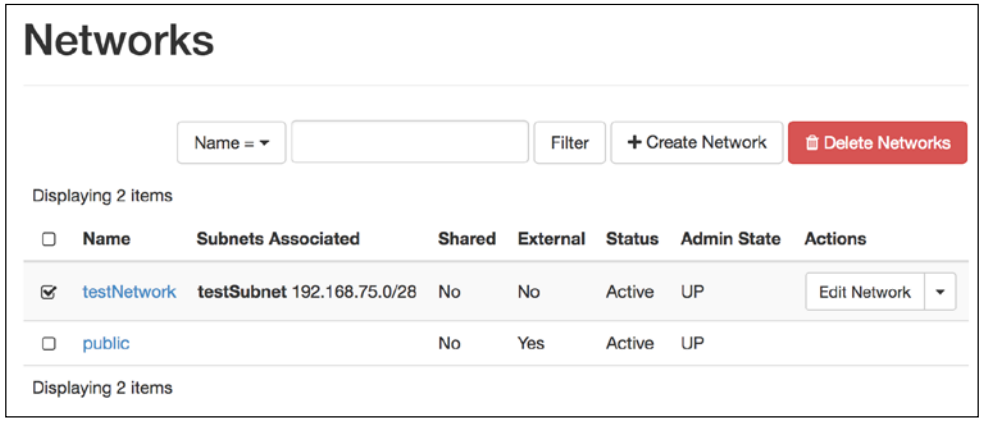
1. To manage networks within our OpenStack Dashboard, select the **Networks** tab, as shown in the following screenshot:



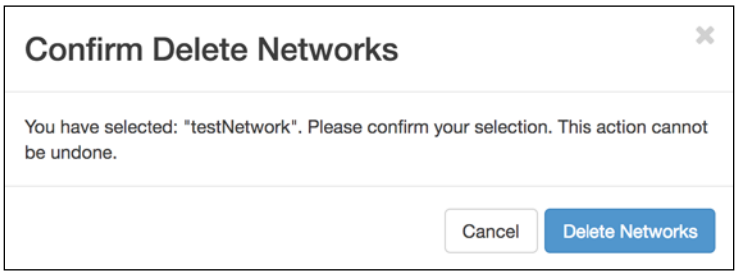
2. When this has been selected we will be presented with a list of networks that we can assign to our instances:



3. To delete a network, select the checkbox next to the name of the network we want to delete, then click on the **Delete Networks** button:



4. We will be presented with a dialog box asking us to confirm the deletion:



5. Clicking on the **Delete Networks** button will remove that network and return us to the list of available networks.

You can only remove a network that has no instances attached to it. You will be warned that this isn't allowed if there are instances still attached to that network.

How it works...

The ability to view and edit Neutron networks is a convenient feature introduced in the one of the earlier releases of OpenStack. Managing Neutron networks and routers can be quite complicated, but having a visual aid, such as the one provided by the OpenStack Dashboard, makes this much easier.

As an administrator (a user with the admin role), you can create shared networks. The same process applies for the preceding recipes, but you are presented with an extra option to allow any created networks to be seen by all projects.

Using OpenStack Dashboard for security group management

Security groups are network rules that allow instances in one project to be kept separate from other instances in another. Managing security group rules for our OpenStack instances is done as simply as possible with OpenStack Dashboard.

Getting ready

Load a web browser, point it to our OpenStack Dashboard address at `http://192.168.100.117/`, and log in as a user in the `default` domain, such as the `developer` user, created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client*, with the `cookbook4` password. The URL for our dashboard is same as the public load balancer IP as discussed in *Chapter 1*, *Installing OpenStack with Ansible*. If you need to find out at what URL your Horizon is, use public IP from the OpenStack catalog list as described in *Chapter 3*, *Keystone – OpenStack Identity Service*.

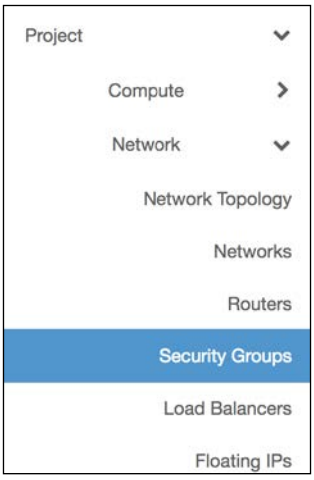
How to do it...

To administer security groups under OpenStack Dashboard, carry out the steps discussed in the following sections:

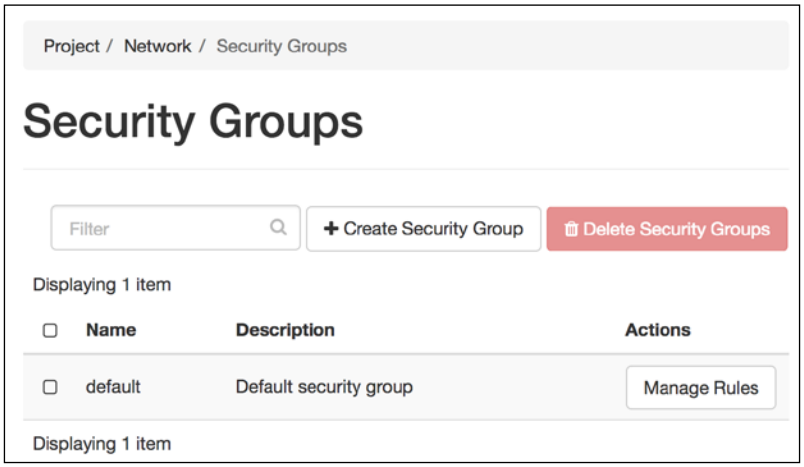
Creating a security group

To create a security group, perform the following steps:

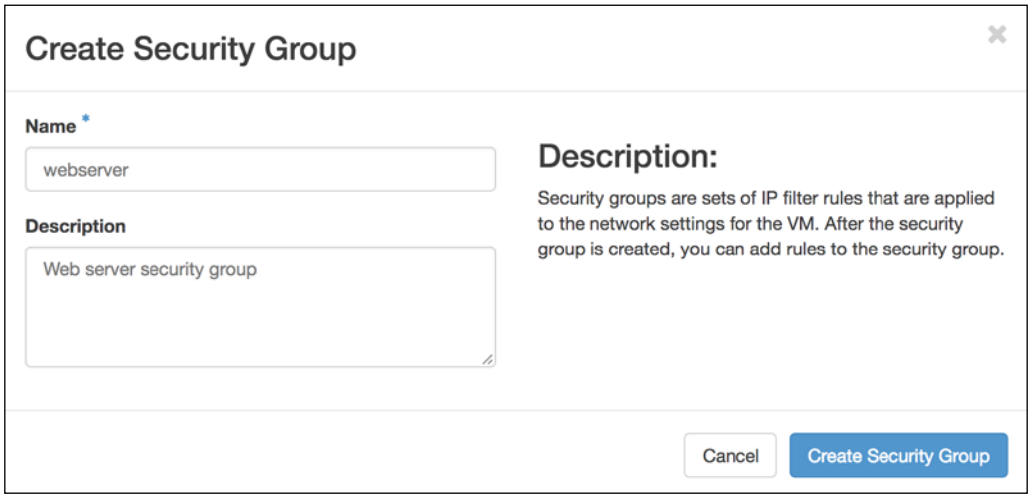
1. A new security group is added to our system using the **Security Groups** tab in the **Network** section, so click on it:



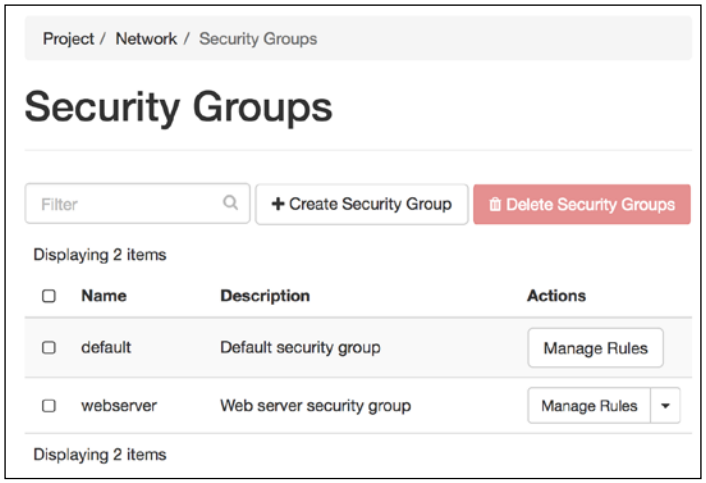
2. Next, we will see a screen allowing access to security settings and manage key pairs. In **Security Groups**, there will be a list of security groups that can be used when we launch our instances. To create a new security group, click on the **Create Security Group** button:



3. We are asked to name the security group and provide a description. The name cannot contain spaces:



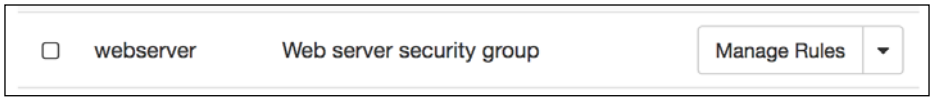
4. Once a new security group is created, the list of available security groups will appear on the screen. From here, we can add new network security rules to the new security group:



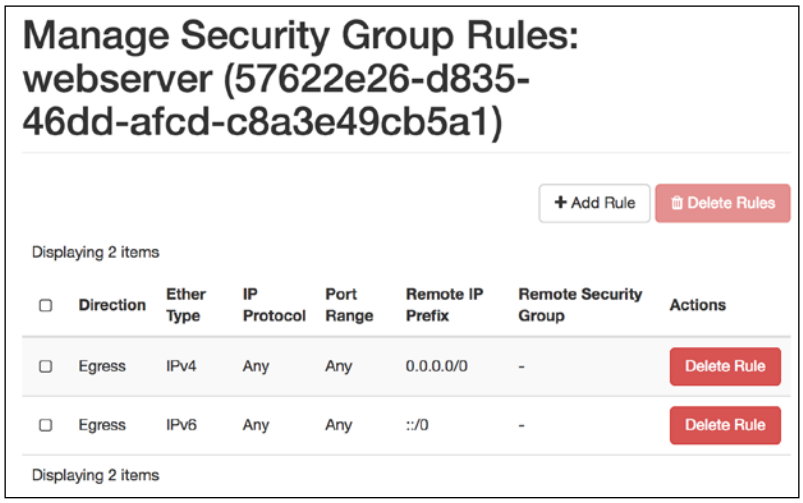
Editing security groups to add and remove rules

To add and remove rules, security groups can be edited by performing the following steps:

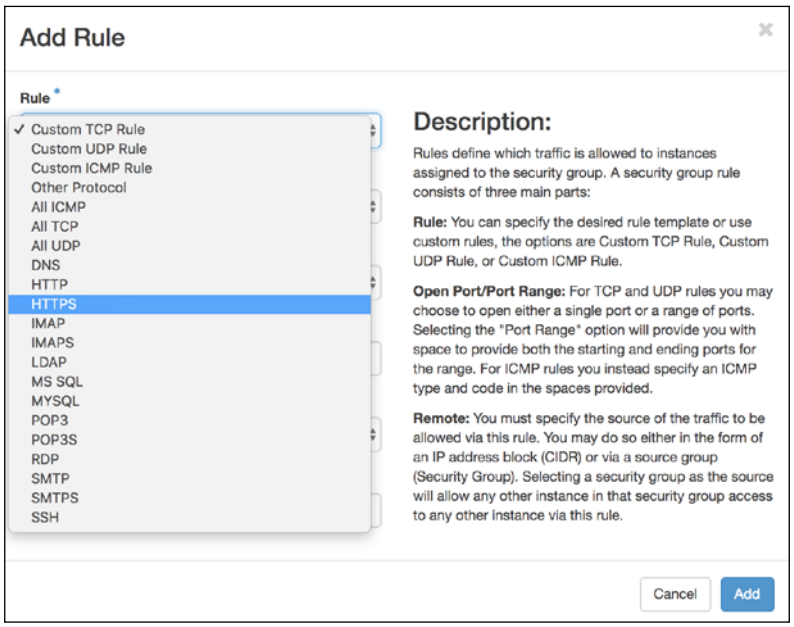
1. When we have created a new security group, or wish to modify the rules in an existing security group, we can click on the **Manage Rules** button for that particular security group:



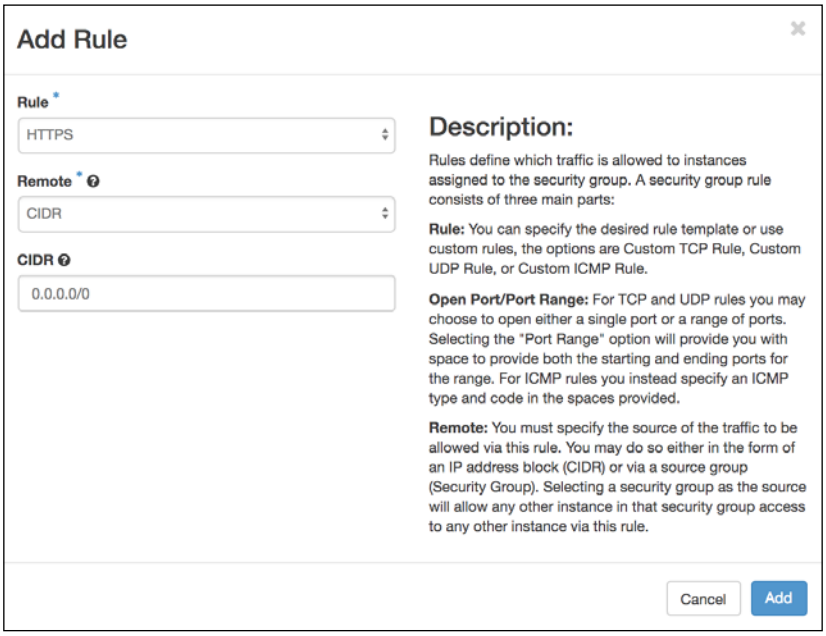
2. After we click the **Manage Rules** button, we will be taken to a screen that lists any existing rules, enabling us to add new rules to this group:



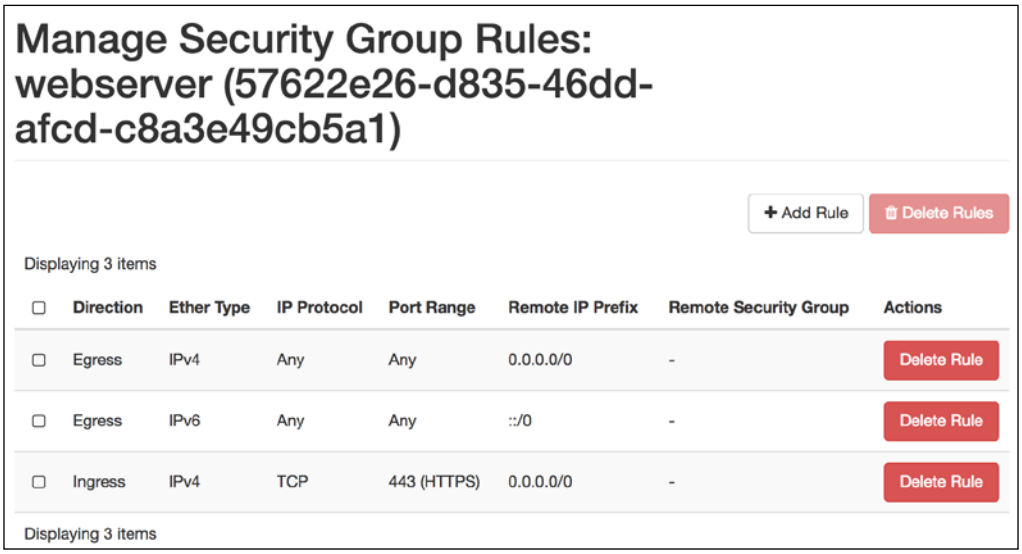
To add a rule to our new security group we click on the **Add Rule** button. This allows us to create rules based on the different protocol types: **ICMP**, **TCP**, and **UDP**. There is also a list of rule templates for commonly added services. As an example, we will add in a security group rule that allows **HTTPS** access from anywhere. To do this, we choose the following:



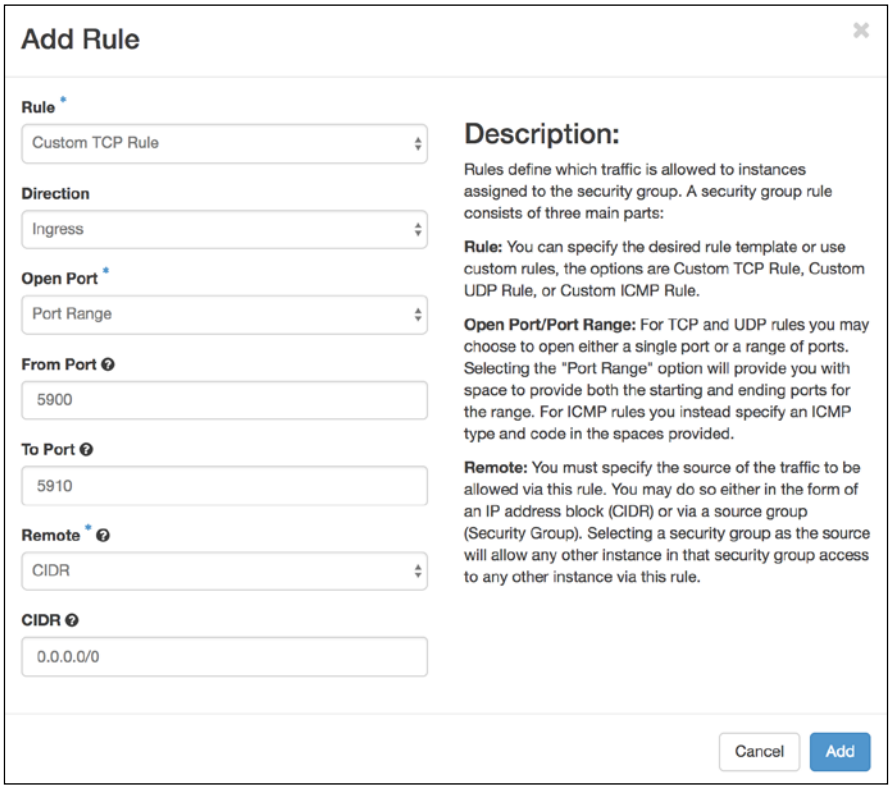
3. We select the **HTTPS** option from the drop-down menu. This returns us to the **Add Rule** menu, where we can specify the source of the network traffic:



4. When we click on the **Add** button, we are redirected to the list of rules now associated with our security group. Repeat the previous step until all the rules related to our security group have been configured:



5. We can also add custom security rules for services that do not have built-in rule templates. After we click on the **Add Rule** button, we will select the **Custom TCP Rule** option from the **Rule** drop-down list. Then, we will select the **Port Range** option from the **Open Port** drop-down list, which presents us with the **From Port** and **To Port** fields. We enter the port range, and click on the **Add** button:

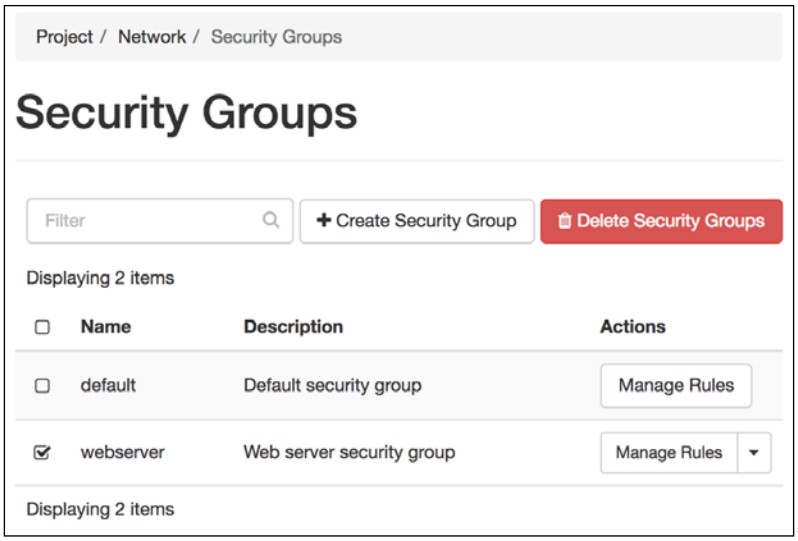


6. Note that we can remove rules from here too. Simply select the rule that we no longer require and click on the **Delete Rule** button. We will be asked to confirm this removal.

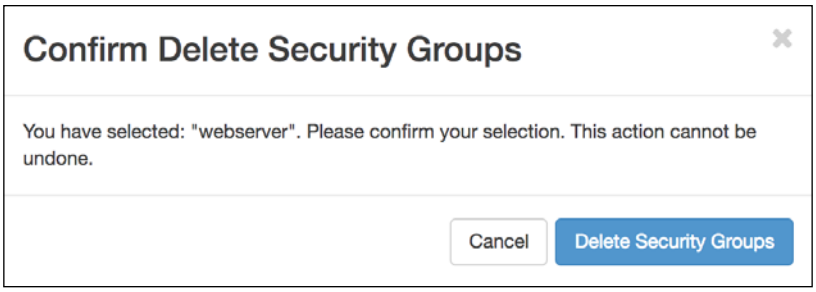
Deleting security groups

Security groups can be deleted by performing the following steps:

1. Security groups are deleted by selecting the security group that we want to remove and clicking on the **Delete Security Groups** button:



2. You will be asked to confirm this. Clicking on **Delete Security Groups** removes the security group and associated access rules:



You will not be able to remove a security group while an instance with that assigned security group is running.

How it works...

Security groups are important to our OpenStack environment, as they provide a consistent and secure approach for accessing our running instances. Allowing the users to create, delete, and amend security groups to use within their projects allows them to create secure environments. Rules within a security group are "deny by default," meaning that if there is no rule for that particular protocol, no traffic for that protocol can access the running instance with that assigned security group.

Security groups are associated with instances on creation; however, they also can be added later on to a running instance. For example, suppose an instance was launched with only the `default` security group. The `default` security group that we have set up only has TCP port `22` accessible and the ability to ping the instance. If we require access to TCP port `80`, we either have to add this rule to the `default` security group or add a new group to the running instance.

Modifications to security groups take effect immediately, and any instance assigned with that security group will have those new rules associated with it.

Using OpenStack Dashboard to launch instances

Launching instances is easily done using the OpenStack Dashboard. We will simply select our chosen image, choose the size of the instance, and then launch it.

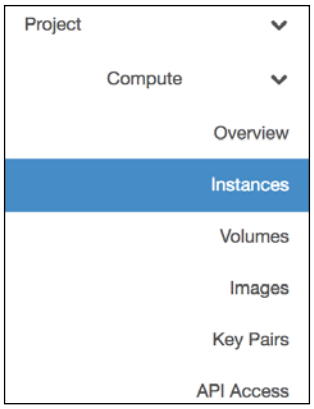
Getting ready

Load a web browser, point it to our OpenStack Dashboard address at `http://192.168.100.117/`, and log in as a user in the `default` domain, such as the `developer` user, created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client* with the `cookbook4` password. The URL for our dashboard is same as the public load balancer IP as discussed in *Chapter 1*, *Installing OpenStack with Ansible*. If you need to find out at what URL your Horizon is, use public IP from the OpenStack catalog list as described in *Chapter 3*, *Keystone – OpenStack Identity Service*.

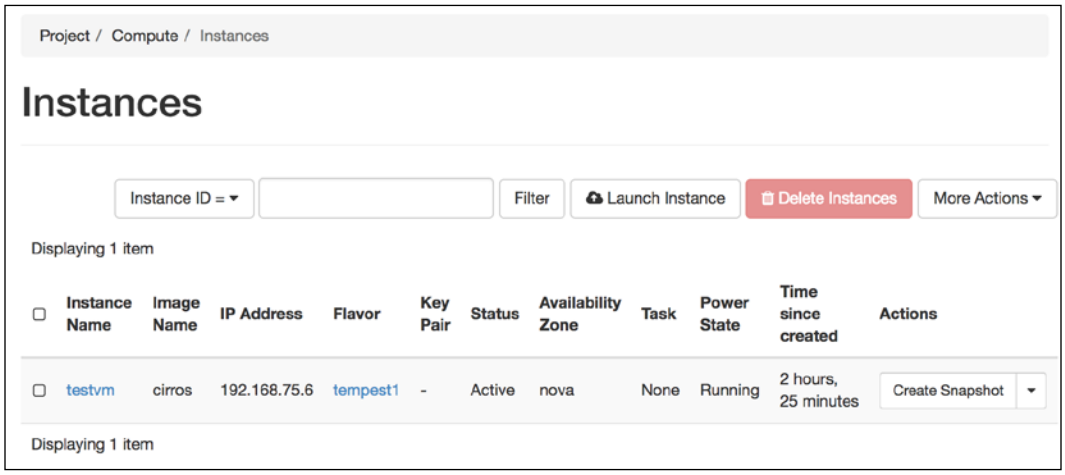
How to do it...

To launch an instance using the OpenStack Dashboard interface, carry out the following steps:

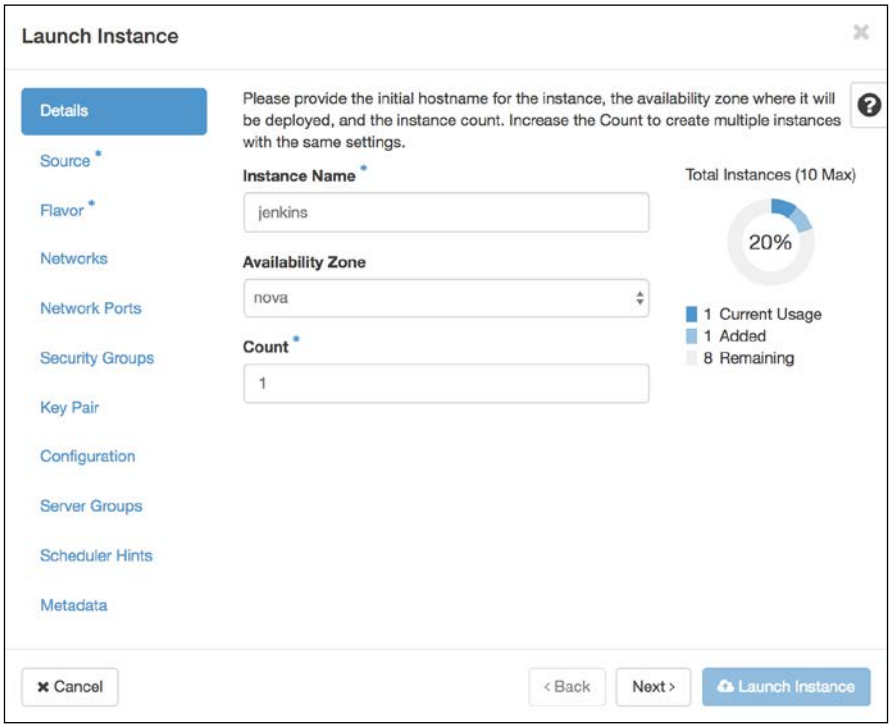
1. Navigate to the **Instances** tab in the **Compute** section:



2. Once there, you will see all the running instances:

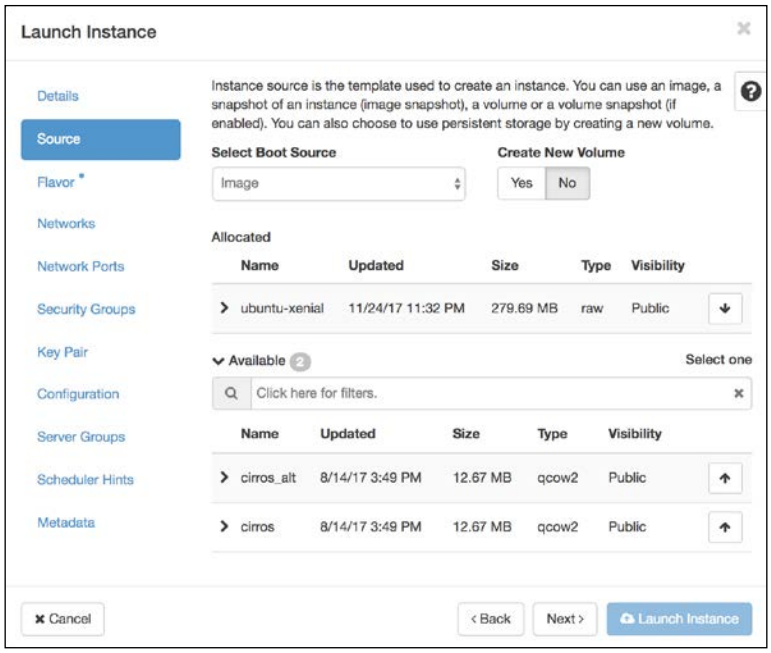


3. From the **Instances** screen, click on the **Launch Instance** button. This will present you with the first tab for instance creation:

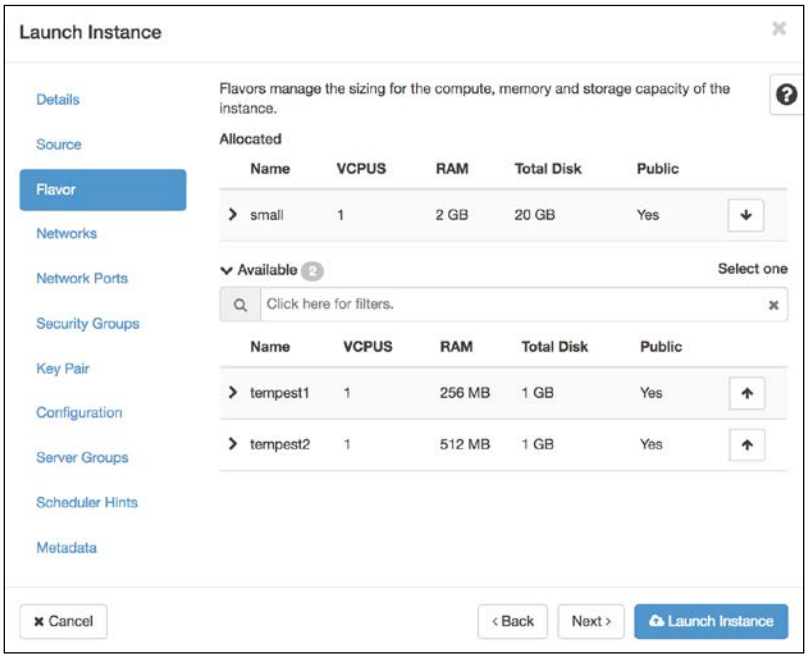


Enter instance name, select the availability zone and instance count. Then, click on the **Next** button.

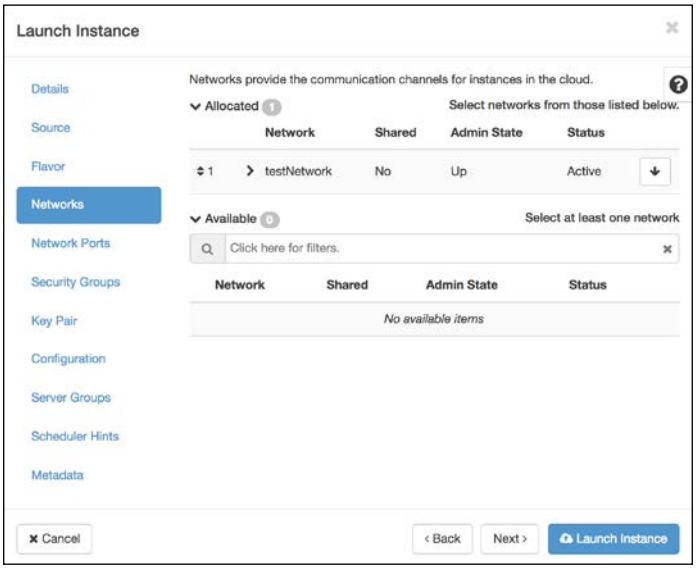
4. Select the source from which your instance should be launched. We are using the preloaded `ubuntu-xenial` image for our example. If you need a cinder volume, you can specify that on this tab as well. Click on the **Next** button after the instance source information is set:



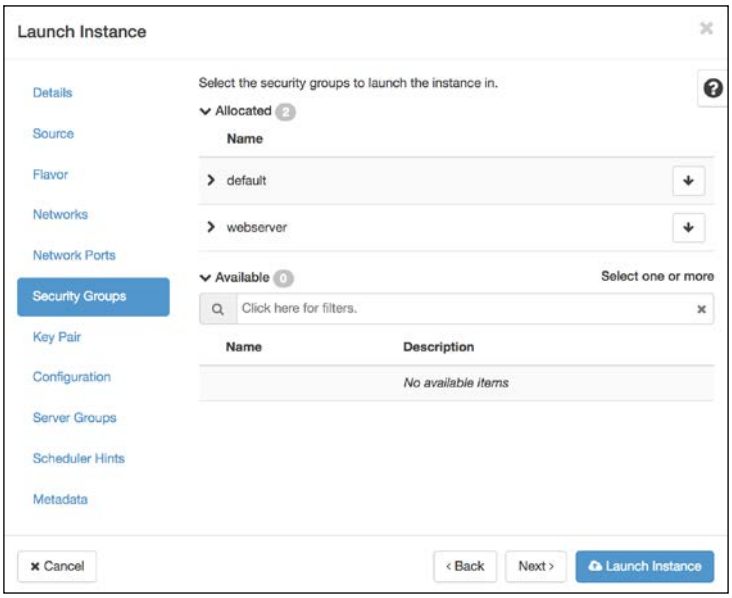
5. Select the flavor for your VM. Flavor determines the size of the VM. Click on the **Nex**t button to proceed:



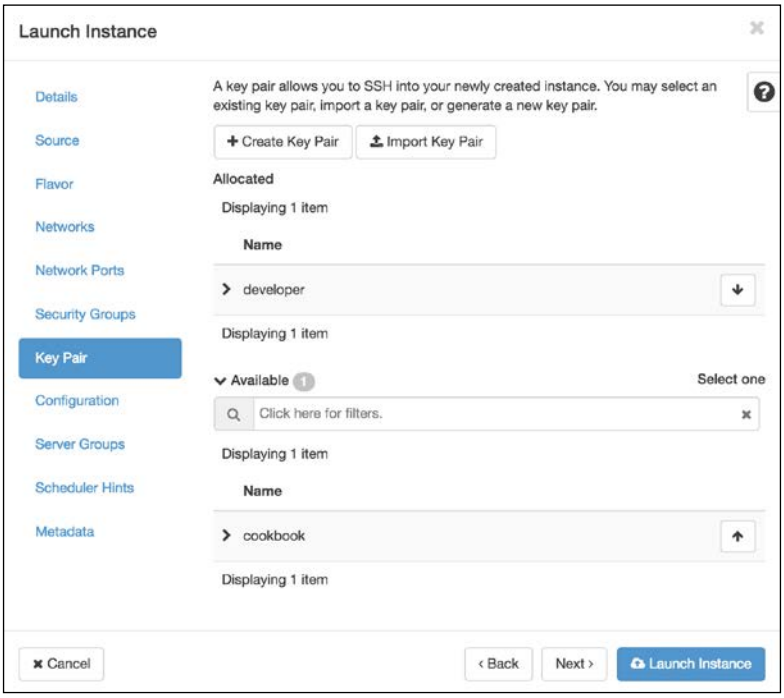
6. Select a network to which your instance should be attached during launch. More than one network may be specified. Once the networks are selected, click on the **Security Groups** tab on the left:



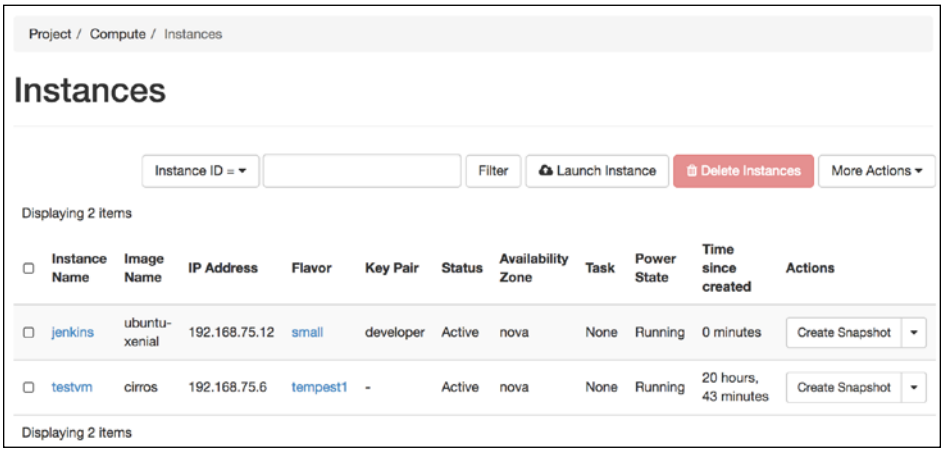
7. Select security groups that should be attached to your instance. Security groups determine network access to newly created VM. Click on the **Next** button to proceed:



8. On this screen, we have to select the key pair that will be used to access the instance after it has booted. After selecting the key pair, you may click on the **Launch Instance** button or check out the rest of the screens for additional information that might be added to the instance:



9. After the VM is launched, a list of existing instances will be displayed. Should there be an issue with the newly created VM, it will be displayed so in the **Status** column:



How it works...

Launching instances from Horizon—the OpenStack Dashboard—is done in several stages, all through the guided **Launch Instance** interface. Here are the required attributes to launch an instance:

- **Source**: This can be an image, snapshot, volume, or volume snapshot
- **Flavor** or **size**: This is for the compute, **memory**, and **storage** capacity of the instance
- **Networks**: At least one network interface must be selected for the instance to launch
- **Security groups**: These are not required to launch an instance; they are needed to control the instance's network access
- **Key pair**: This is also optional, but recommended for logging on to instances

The **Instances** tab shows the running instances under our development project.

You can also see an overview of what is running in our environment, by clicking on the **Overview** tab.

Using OpenStack Dashboard to delete instances

Terminating instances is very simple when using the OpenStack Dashboard.

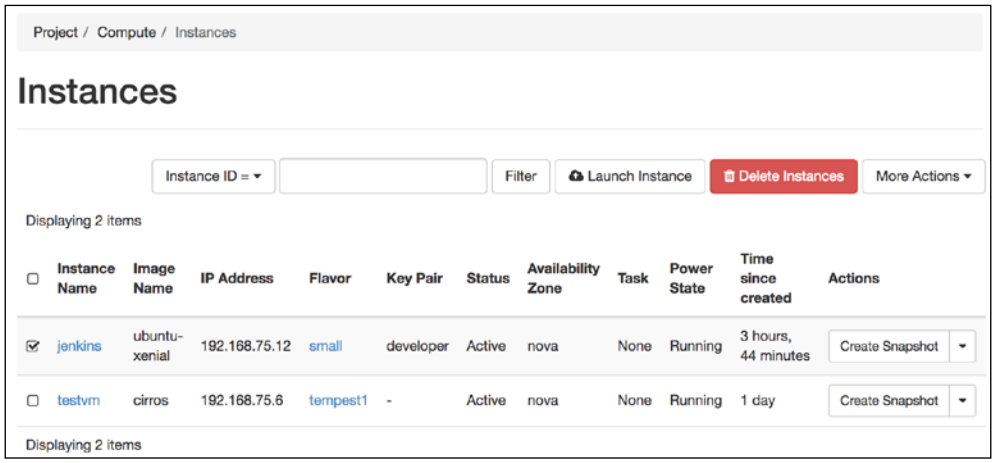
Getting ready

Load a web browser, point it to our OpenStack Dashboard address at `http://192.168.100.117/`, and log in as a user in the `default` domain, such as the `developer` user, created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client* with the `cookbook4` password. The URL for our dashboard is same as the public load balancer IP as discussed in *Chapter 1*, *Installing OpenStack with Ansible*. If you need to find out at what URL your Horizon is, use public IP from the OpenStack catalog list as described in *Chapter 3*, *Keystone – OpenStack Identity Service*.

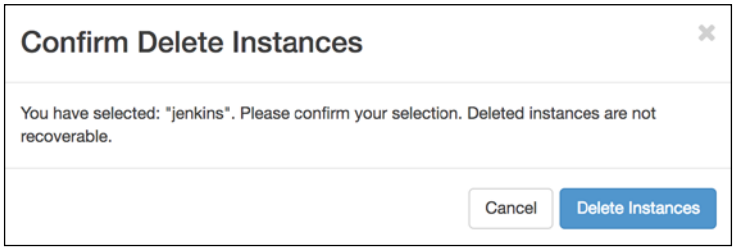
How to do it...

To deleting instances using OpenStack Dashboard, carry out the following steps:

1. Select the **Instances** tab and choose the instance to be terminated by selecting the checkbox next to the instance name (or names), and then click on the red **Delete Instances** button:



2. We will be presented with a confirmation screen. Click on the **Delete Instances** button to terminate the selected instance:



3. Now we will be presented with the **Instances** screen with a confirmation that the instance has been deleted successfully.

How it works...

Deleting instances using OpenStack Dashboard is easy. We will select our running instance and click on the **Delete Instances** button, which is highlighted when an instance is selected. After clicking on the **Delete Instances** button, we will be asked to confirm this action to minimize the risk of accidentally terminating an instance.

Using OpenStack Dashboard to add new projects

OpenStack Dashboard is a lot more than just an interface to our instances. It allows an administrator to configure environments, users, and projects. Adding new projects that users can be members of is achieved quite simply in OpenStack Dashboard.

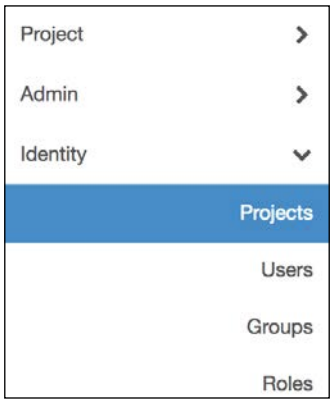
Getting ready

Load a web browser, point it to our OpenStack Dashboard address at `http://192.168.100.117/`, and log in as a user in the `default` domain, such as the `developer` user, created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client* with the `cookbook4` password. The URL for our dashboard is same as the public load balancer IP as discussed in *Chapter 1*, *Installing OpenStack with Ansible*. If you need to find out at what URL your Horizon is, use public IP from the OpenStack catalog list as described in *Chapter 3*, *Keystone – OpenStack Identity Service*.

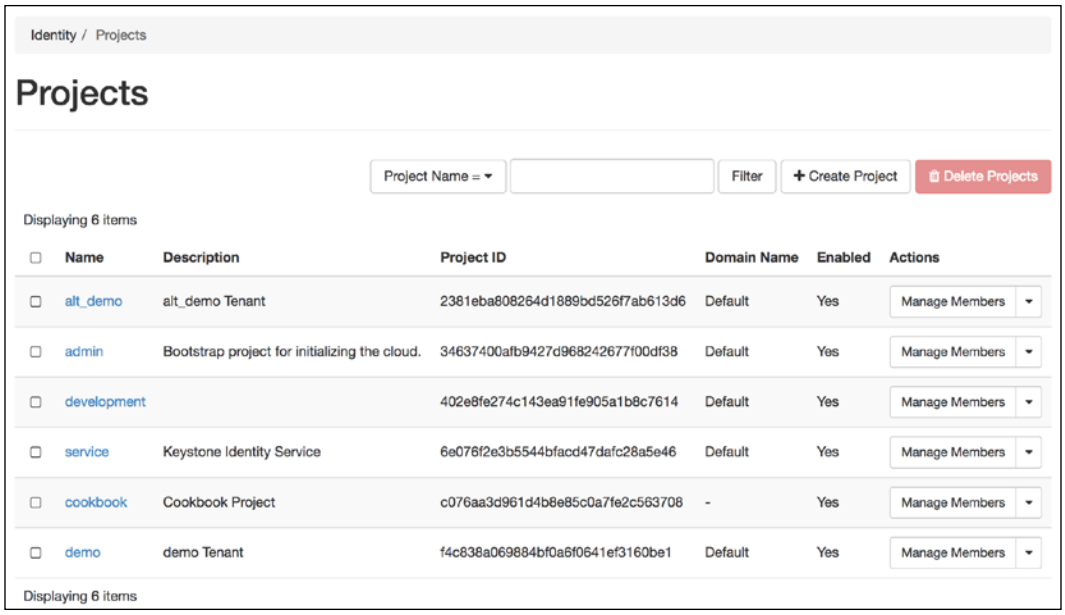
How to do it...

To add a new project to our OpenStack environment, carry out the following steps:

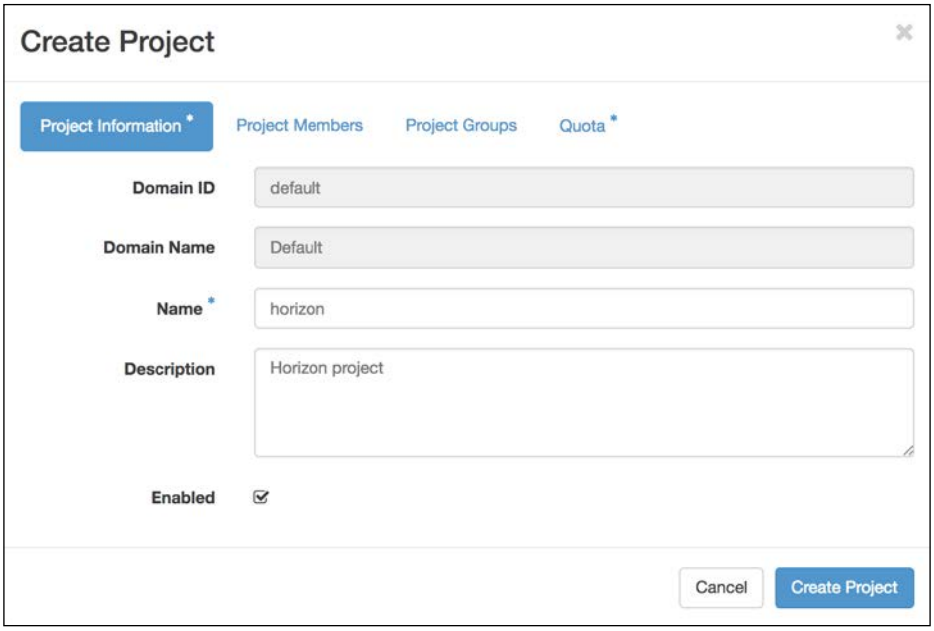
1. After we login as a user with admin privileges, we get more menu options under the **Identity** tab. One of them is a **Projects** option, as shown in the following screenshot:



2. To manage projects, we click on the **Projects** option listed under the **Identity** tab. This will list the available projects in our environment, as shown in the following screenshot:



3. To create a new project, click on the **Create Project** button.
4. Next, we will be presented with a form that asks for the name of the project and a description. Enter `horizon` as our project name, and enter a description:



5. We will enable the project by selecting the **Enabled** checkbox, and then click on the **Create Project** button.
6. We will be presented with the list of projects that are now available and a message saying that the `horizon` project was created successfully.

How it works...

OpenStack Dashboard is a feature-rich interface that complements the command-line options available to you when managing our OpenStack environment. This means that we can simply create a project to which users can belong using OpenStack Dashboard. When creating a new project, we need to be logged in as a user with admin privileges to get access to the full project management interface.

Using OpenStack Dashboard for user management

The OpenStack Dashboard gives us the ability to administer users through the web interface. This allows an administrator to easily create and edit users within an OpenStack environment. To manage users, you must log in using an account that is a member of the admin role.

Getting ready

Load a web browser, point it to our OpenStack Dashboard address at `http://192.168.100.117/`, and log in as a user in the `default` domain, such as the `developer` user, created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client* with the `cookbook4` password. The URL for our dashboard is same as the public load balancer IP as discussed in *Chapter 1*, *Installing OpenStack with Ansible*. If you need to find out at what URL your Horizon is, use public IP from the OpenStack catalog list as described in *Chapter 3*, *Keystone – OpenStack Identity Service*.

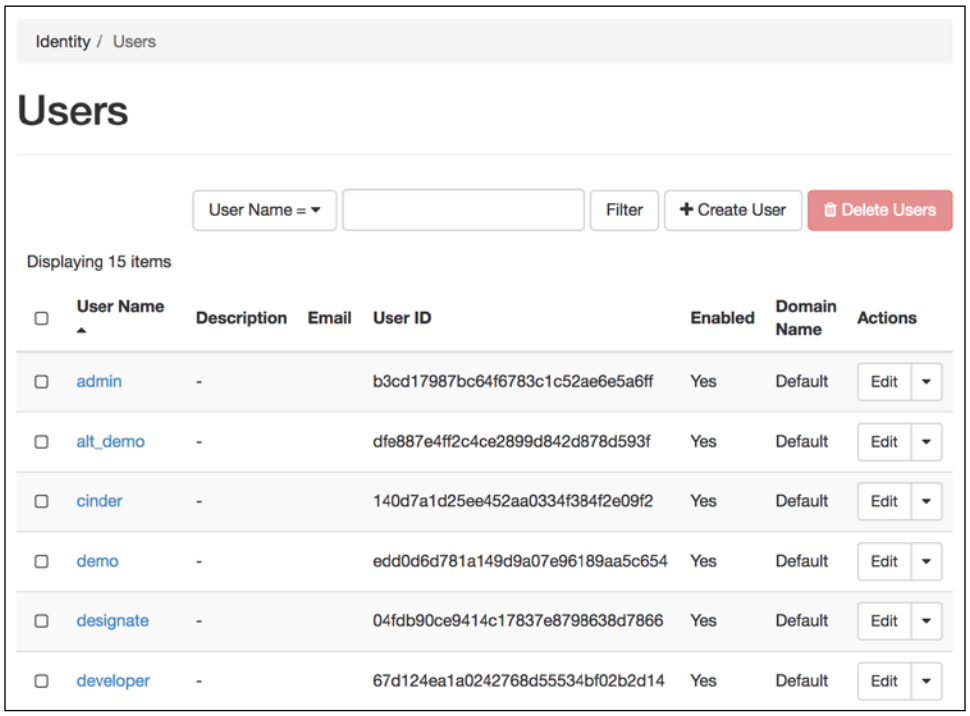
How to do it...

User management under OpenStack Dashboard is achieved by carrying out the steps discussed in the following sections.

Adding users

To add users, perform the following steps:

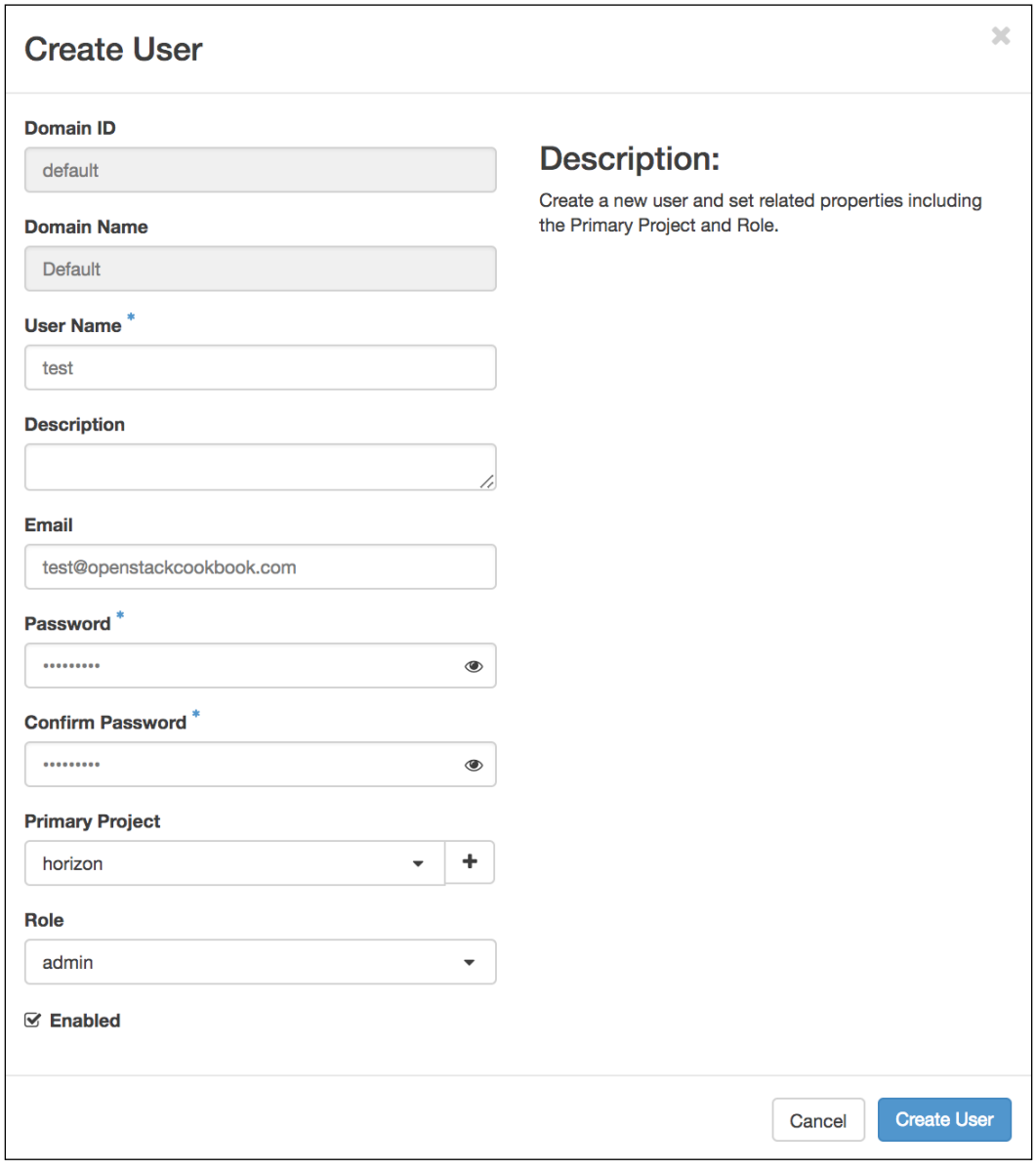
1. In the **Identity** panel, click on the **Users** option to get a list of users configured on the system:



   This screenshot truncates the available user list for the sake of clarity and space.

2. To create a new user, click on the **Create User** button.

3. We will be presented with a form that asks for username details. Enter the username, email, and the password for that user. In the example shown in the following screenshot, we create a user named `test`, set a password, and assign that user to the `horizon` project with the role of **admin**. Click on the **Create User** button to create a new user with all the details specified:



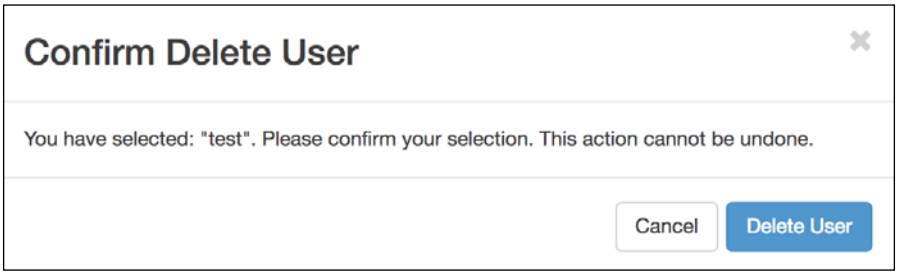
4. We will be returned to the screen listing the users of our OpenStack environment with a message stating that our user creation was successful.

Deleting users

To delete users, perform the following steps:

1. In the **Identity** panel, click on the **Users** option to get a list of users on the system.
2. We will be presented with a list of users in our OpenStack environment. To delete a user, click on the **Edit** button, which will present a drop-down list with the **Delete User** option:

3. Clicking on the **Delete User** option will bring up a confirmation dialog box. Confirm by clicking on the **Delete User** button to remove that user from the system:

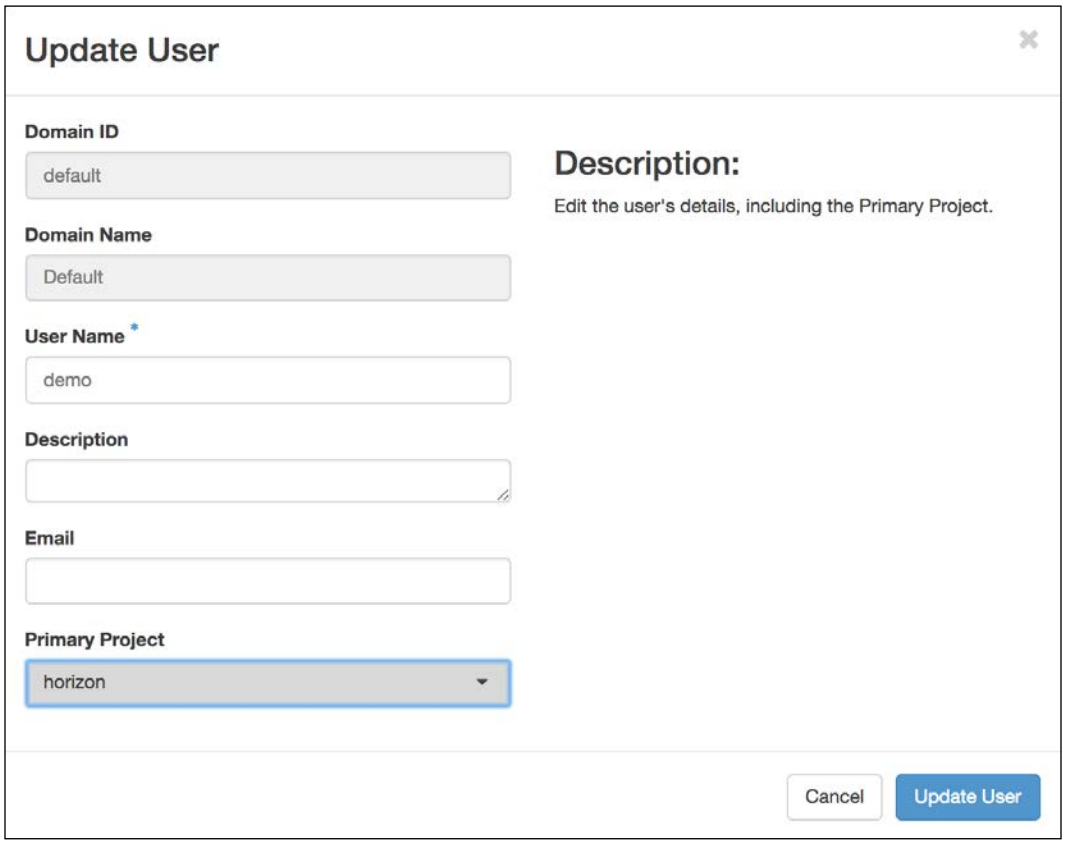


Updating user details and passwords

To update user details and passwords, perform the following steps:

1. In the **Identity** panel, click on the **Users** option to bring up a list of users on the system.
2. To change a user's password, email address, or primary project, click on the **Edit** button for that user.

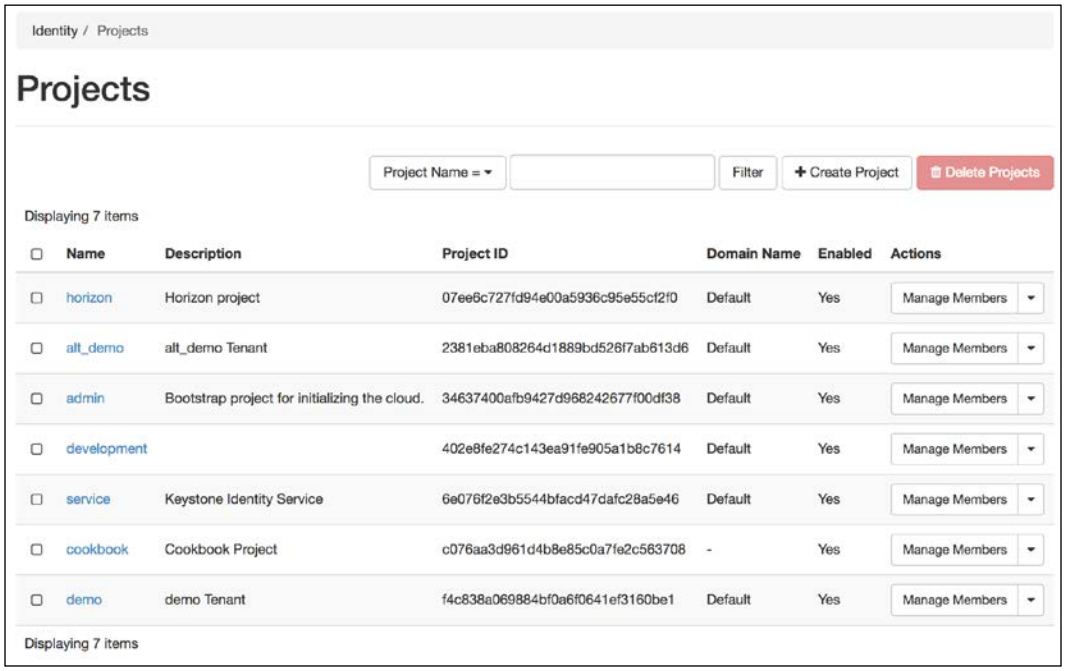
3. This brings up a dialog box asking for the relevant information. When the information has been set as we want it to be, click on the **Update User** button:



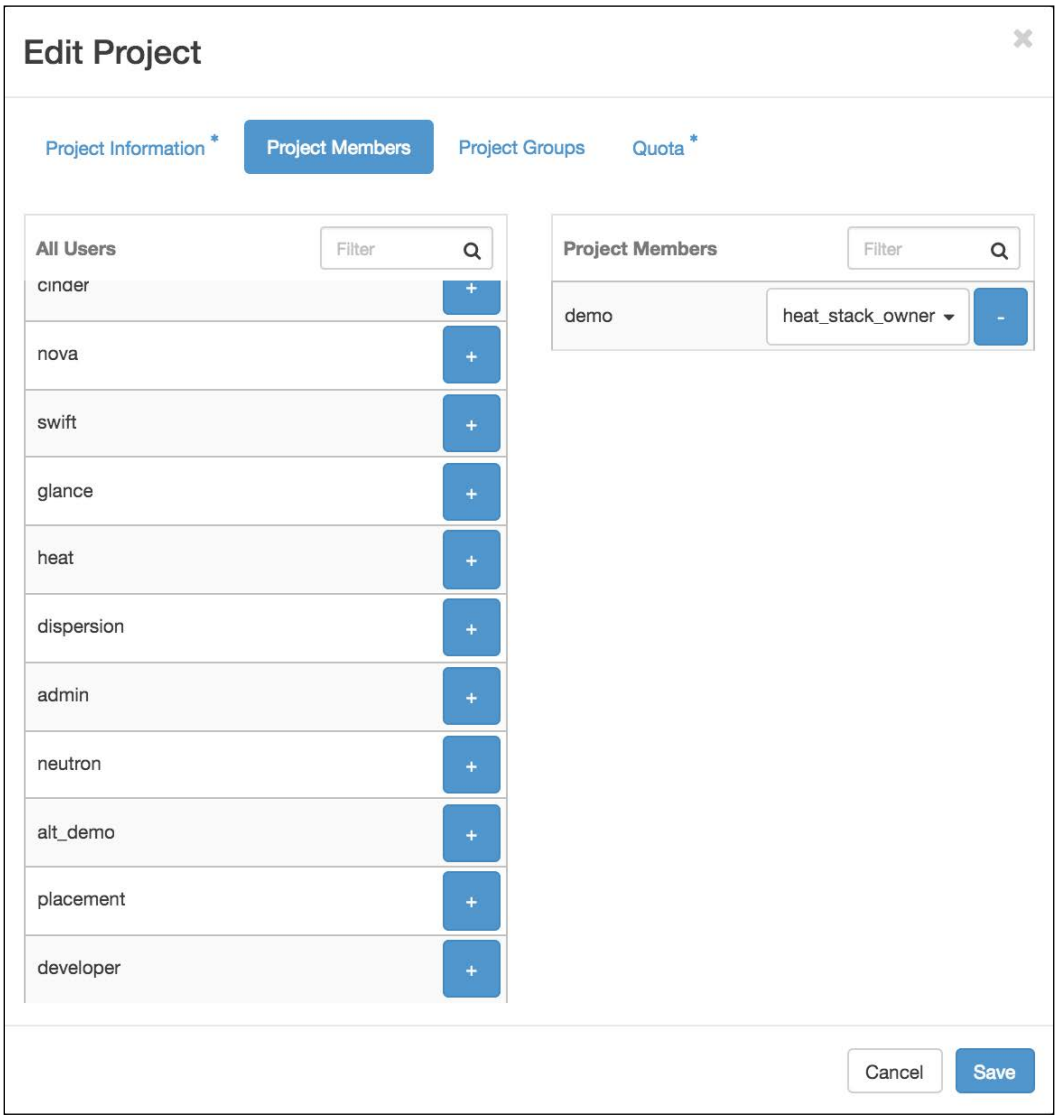
Adding users to projects

To add users to projects, perform the following steps:

1. In the **Identity** panel, click on the **Projects** option to bring up a list of projects on the system:

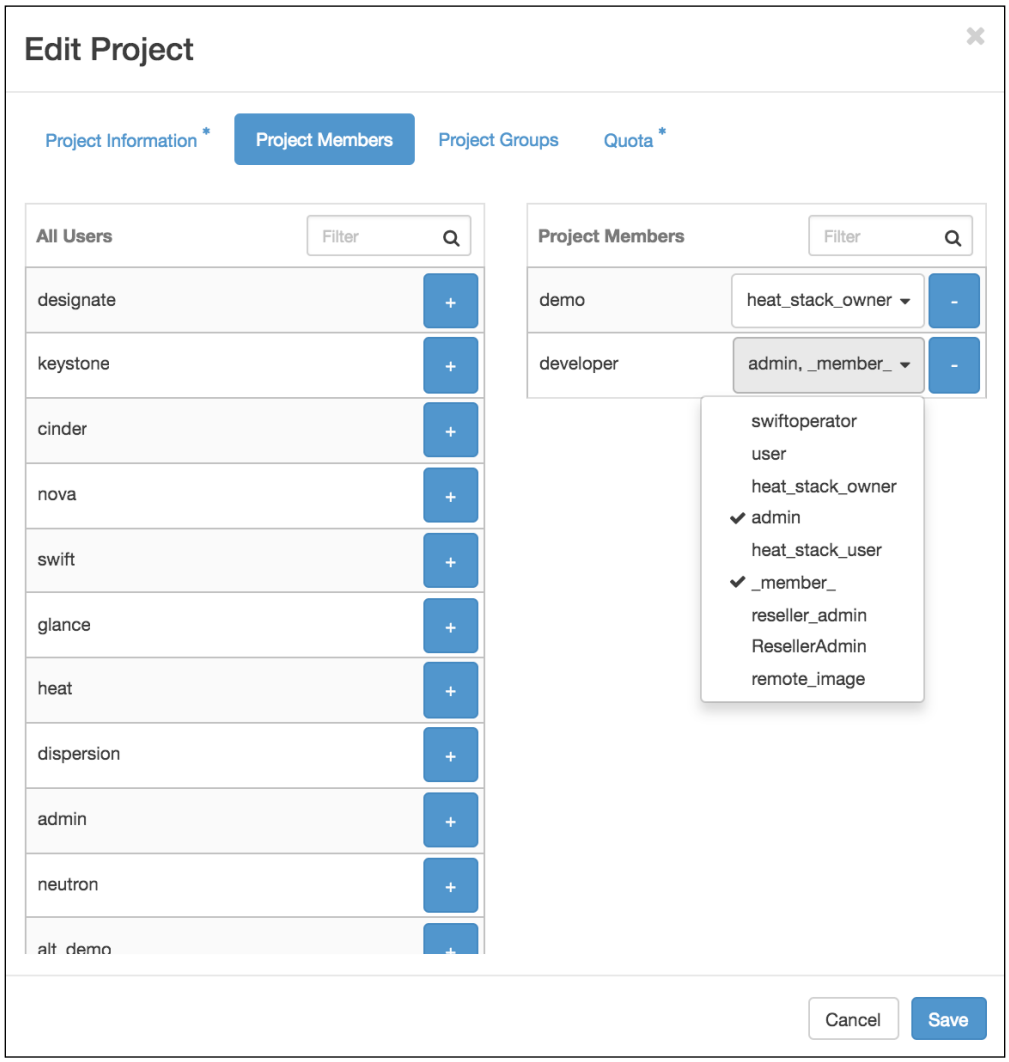


2. Click on the **Manage Members** option to bring up a list of users associated with a project as well as a list of users, which we can add to that project:



3. To add a new user to the list, simply click on the **+** (plus sign) button next to that user.

4. To change the role of the user within that project, select the dropdown next to the username and select a new role (or roles):



In our example, we have added the `developer` user to the `demo` project and have given the `user` `admin` and `_member_` roles.

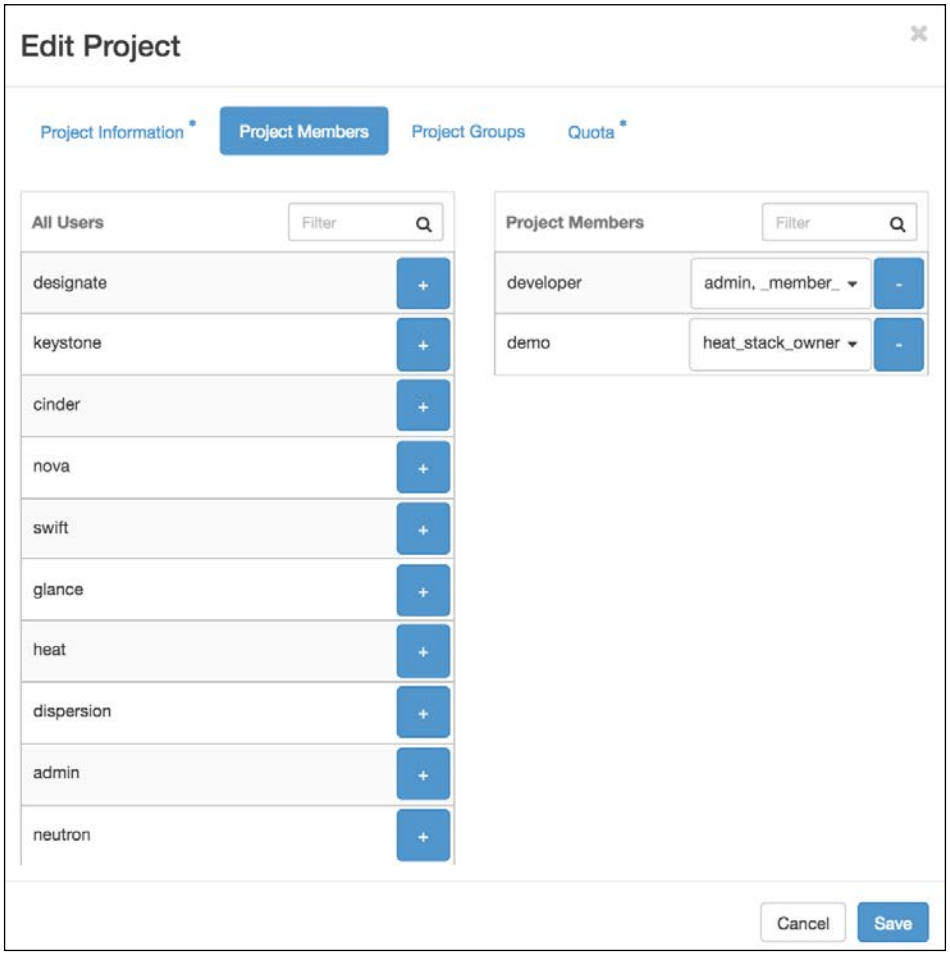
5. After clicking on the **Save** button at the bottom of the dialog box, we will see a message saying that our project has been updated. This user can now launch instances in different projects when they log on.

Removing users from projects

To remove users from projects, perform the following steps:

1. In the **Identity** panel, click on the **Projects** option to bring up a list of projects on the system.
2. To remove a user from a project, for example, `demo`; click on the **Manage Members** button:

3. By clicking on the **Manage Members** button, you will get a modal window with a list of all users as well as project members, which we can remove from that project:



4. To remove a user from this project, click on the **-** (minus sign) button next to that particular user under project members.
5. After clicking on the **Save** button at the bottom of the dialog box, we will see a message saying that our project has been updated.

How it works...

The OpenStack Dashboard makes it easy to create users, modify their membership within projects, update passwords, and remove them from the system altogether.

Using OpenStack Dashboard with LBaaS

The OpenStack Dashboard has the ability to view, create and edit load balancers, add **Virtual IPs** (**VIPs**), and add nodes to be behind a load balancer. The dashboard also provides an interface for creating the HA Proxy server load balance service for our instances. We do this first by creating load balancing pool and then adding running instances to those pools.

In this section, we will use two instances running Apache. We will create a HTTP load balancing pool, create a VIP, and configure instances to be part of the pool. The result will be an ability to use the HTTP load balancer pool address to send traffic to two instances running Apache.

Getting ready

Load a web browser, point it to our OpenStack Dashboard address at `http://192.168.100.117/`, and log in as a user in the `default` domain, such as the `developer` user, created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client* with the `cookbook4` password. The URL for our dashboard is same as the public load balancer IP as discussed in *Chapter 1*, *Installing OpenStack with Ansible*. If you need to find out at what URL your Horizon is, use public IP from the OpenStack catalog list as described in *Chapter 3*, *Keystone – OpenStack Identity Service*.

How to do it...

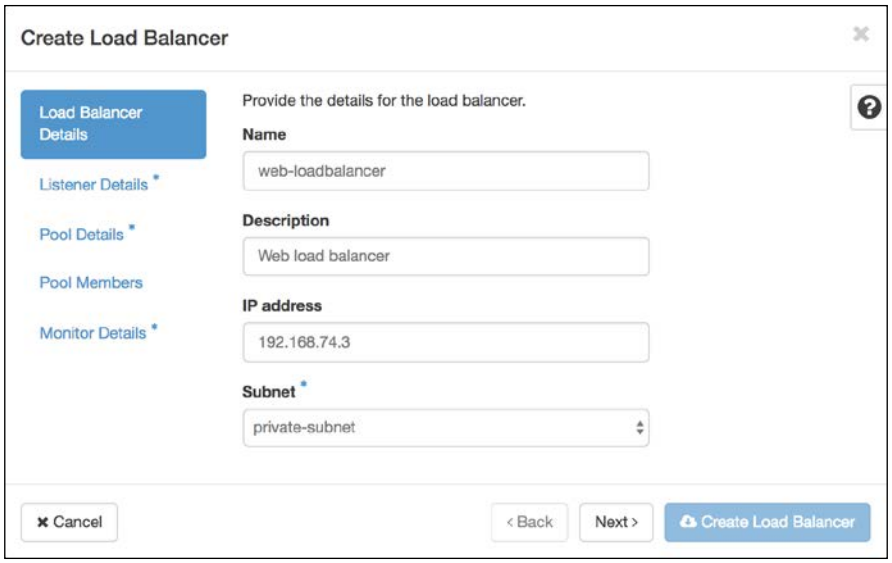
This recipe will walk you through creating and deleting load balancers using the OpenStack Dashboard:

Creating load balancer

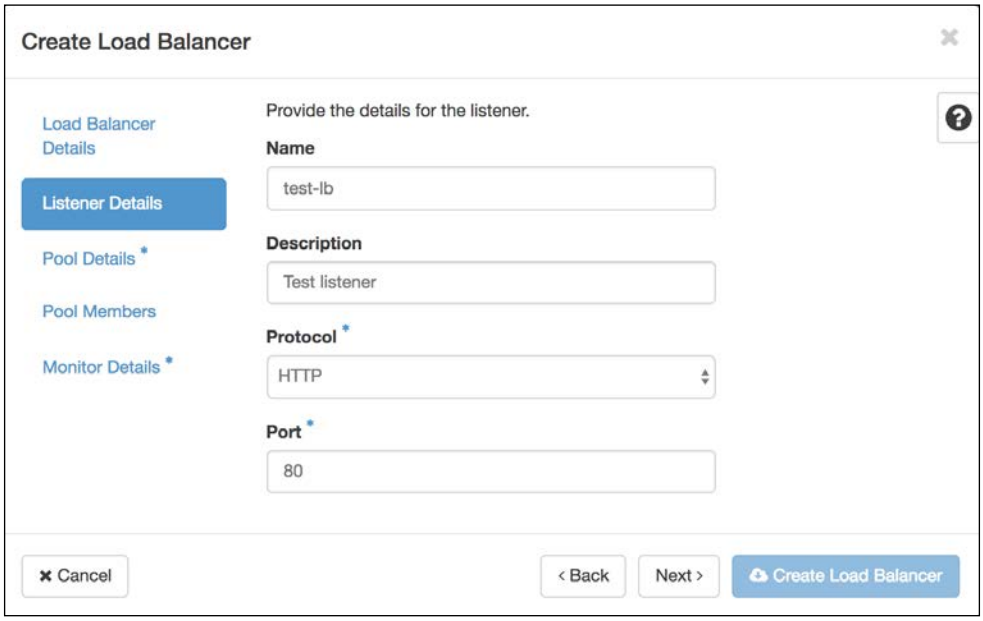
To create a load balancer pool for a logged in user, carry out the following steps:

1. To manage load balancers within our OpenStack Dashboard, select the **Load Balancers** tab, as shown in the following screenshot:

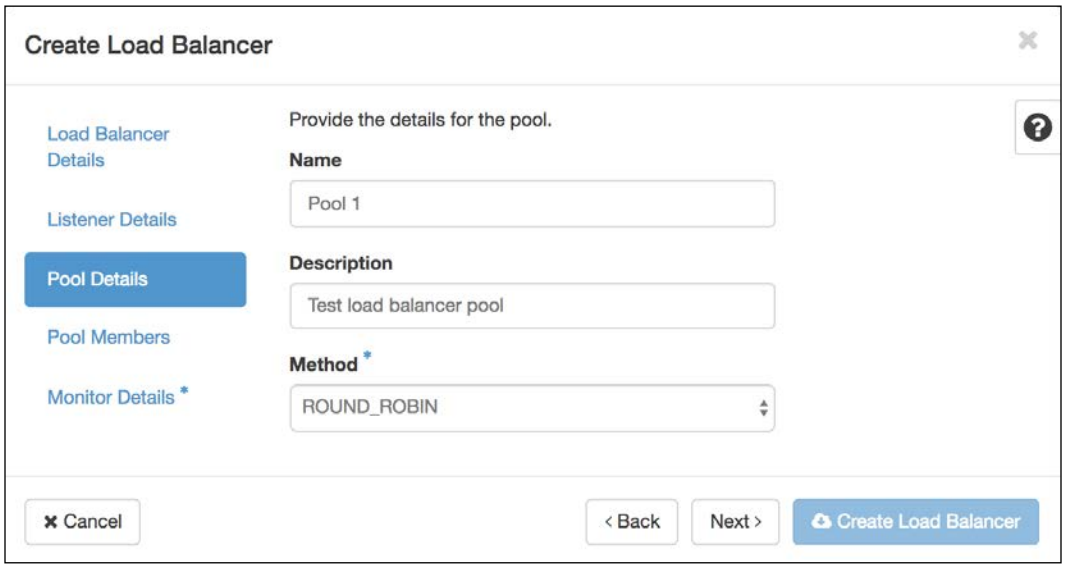
2. This will show available load balancer pools. Since we currently do not have any created, click on the **Create Load Balancer** button in the top right corner to add a load balancer.
3. After clicking on the **Create Load Balancer** button, we will be presented with a modal window. Fill out the details to add a new load balancer:



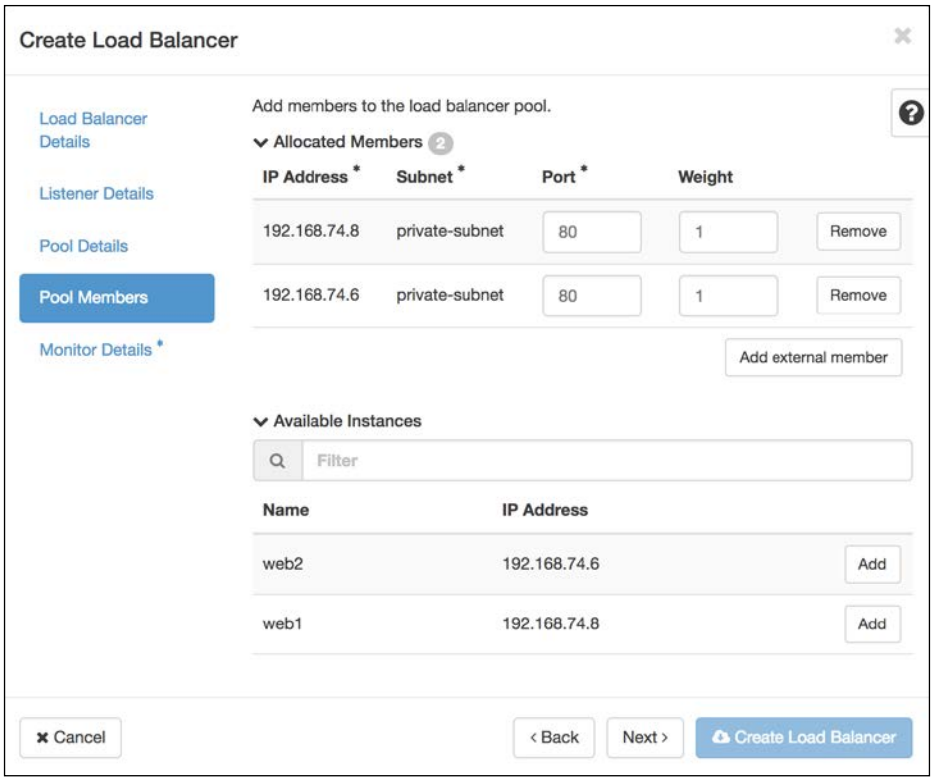
4. The IP address is not required; if one is not provided, the system will select the next available IP on the subnet selected.
5. Select a subnet for the pool by clicking on the drop-down menu. All of our instances are attached to the private network, so we select **private-subnet**.
6. After clicking on the **Next** button, provide details for the load balancer's listener:



7. The `HTTP` and `TCP` protocols are available to us, so we will use `HTTP`. `TERMINATED_HTTPS` is available only with a configured key-manager service.
8. Select a port on which the frontend listens. Multiple listeners can be configured on one load balancer, but each must have a unique port.
9. After clicking on the **Next** button, provide pool details. We will use the `ROUND_ROBIN` load balancing method. Other algorithms include `LEAST_CONNECTIONS` and `SOURCE_IP`:



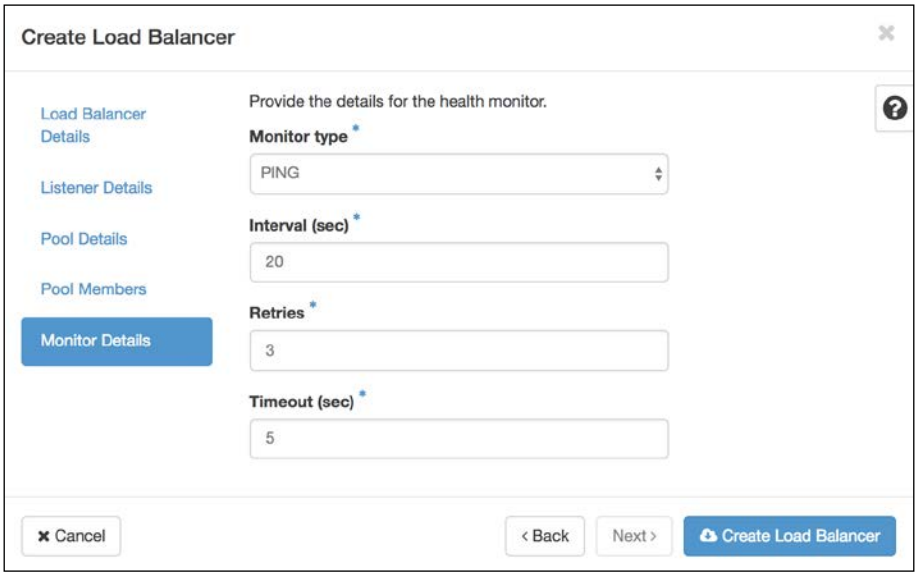
10. Click on the **Next** button to proceed to the **Pool Members** screen. Add members to the pool by clicking on the **Add** button next to the available instances:



If the instance is not listed under **Available Instances**, click on the **Add external member** button.

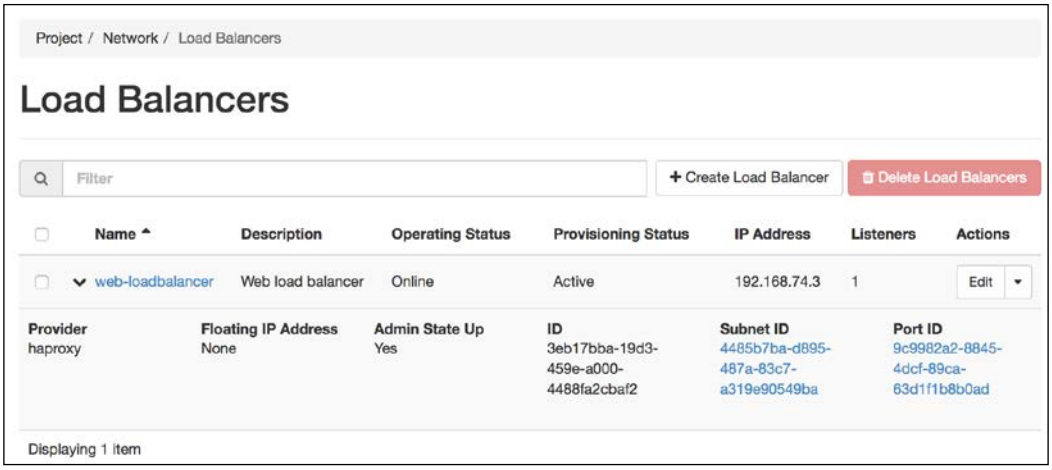
11. After clicking on the **Next** button, we will be presented with the **Monitor Details** screen. Here we configure our health checks for pool members. We will use `PING` for monitor type. Other options available are `HTTP` and `TCP`.

12. Set health check interval time in seconds, followed by the number of retries and the time after which the health check will timeout:



Each pool can have only one health monitor associated with it.

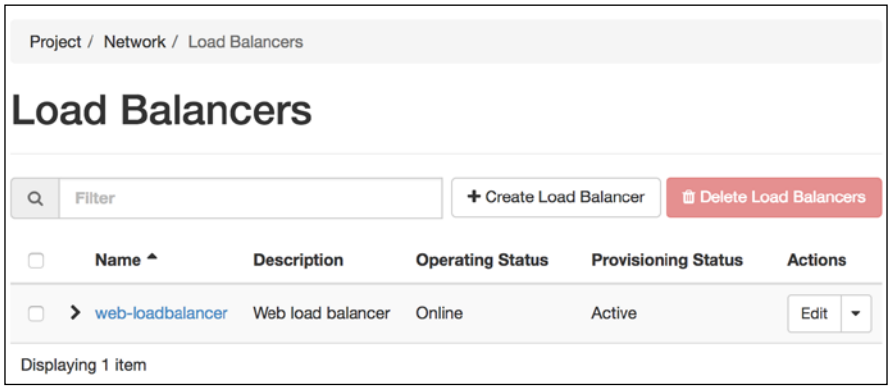
13. Click on the **Create Load Balancer** button to create the load balancer with all the details entered. You will see a newly created load balancer in a list of all available load balancers.



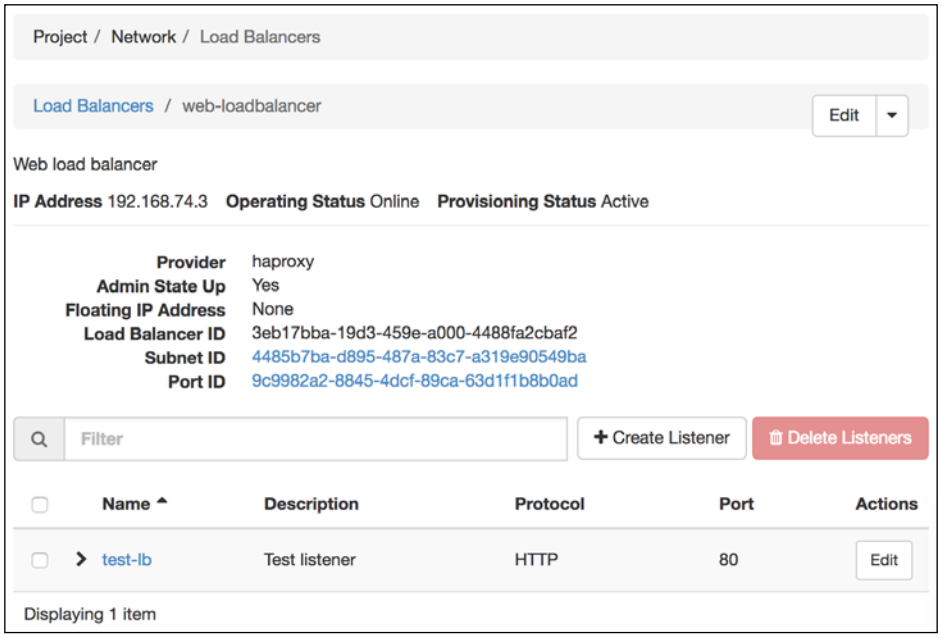
Deleting load balancer

To delete the load balancer, we will first delete attached health monitors, listeners and pools. In our example, we will show how to delete one of each, but if you have multiple pools, you will need to remove all pools, listeners, and health monitors before you are able to delete a load balancer.

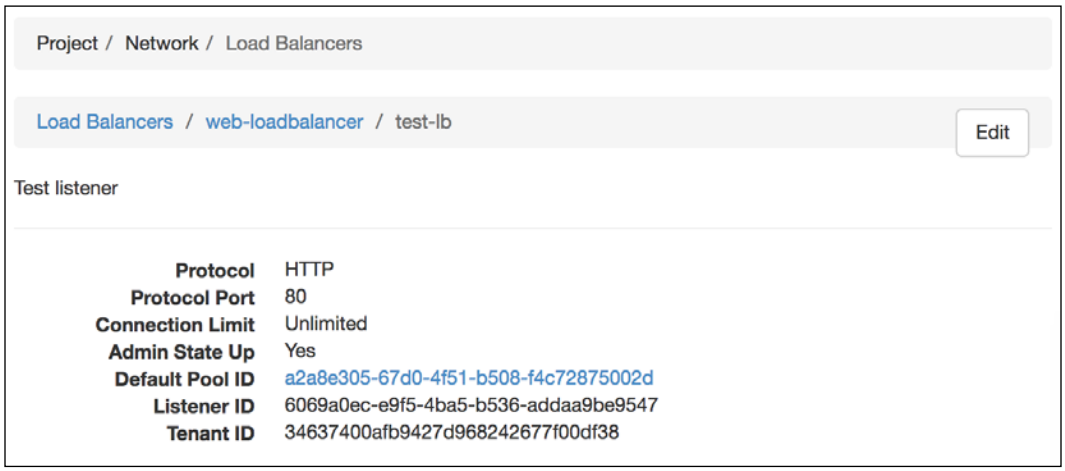
1. Start on a **Load Balancers** page and click on the load balancer name you wish to delete. In our example, the name is `web-loadbalancer`:



2. From load balancer details page, click on the listener name. In our example, the listener is named `test-lb`:

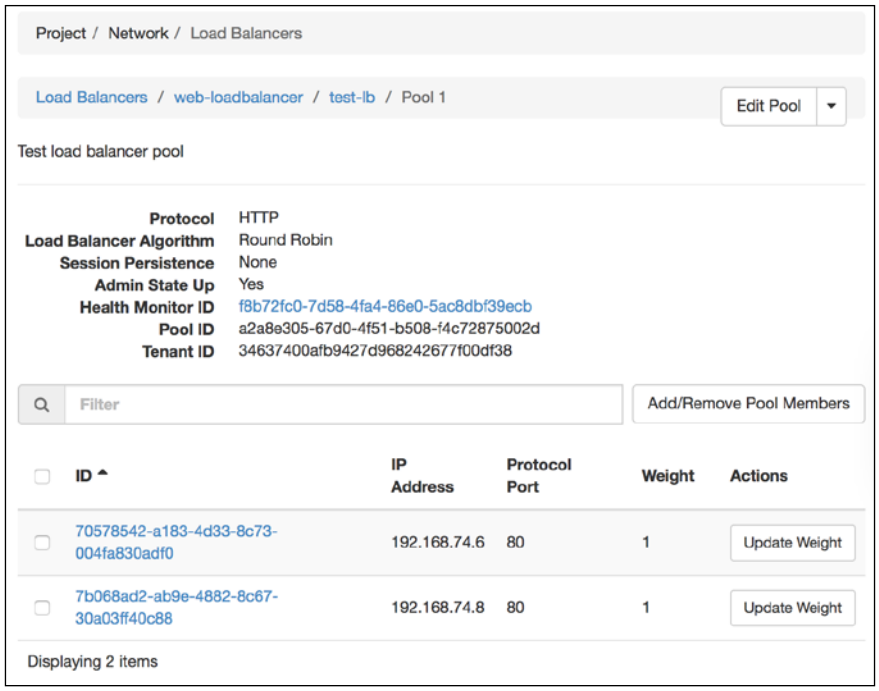


3. On the listener details page, click on **Default Pool ID**:

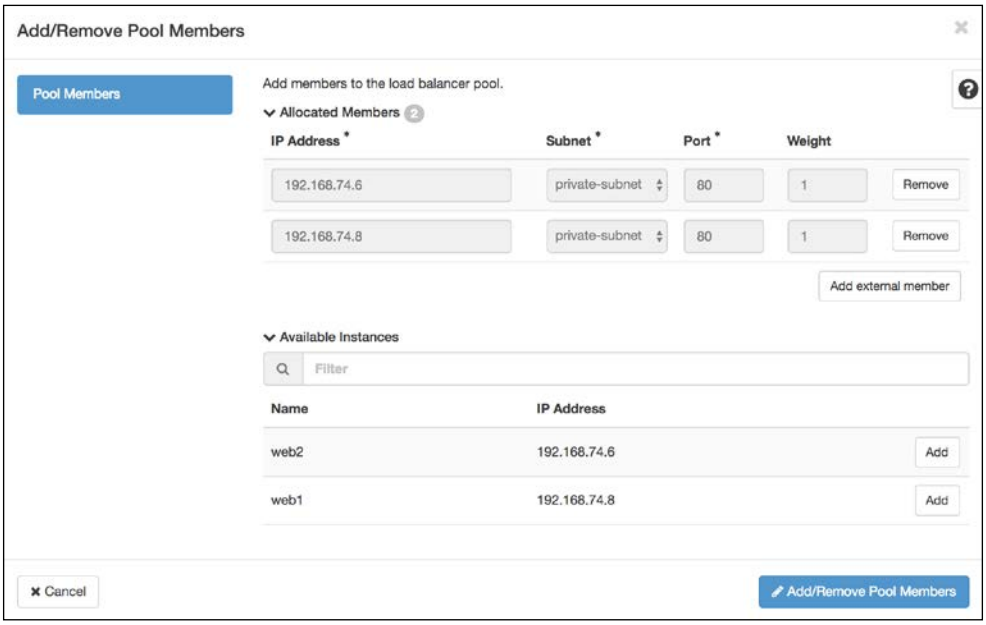


The default pool ID is right under the admin state, and it is the first of the three IDs shown:

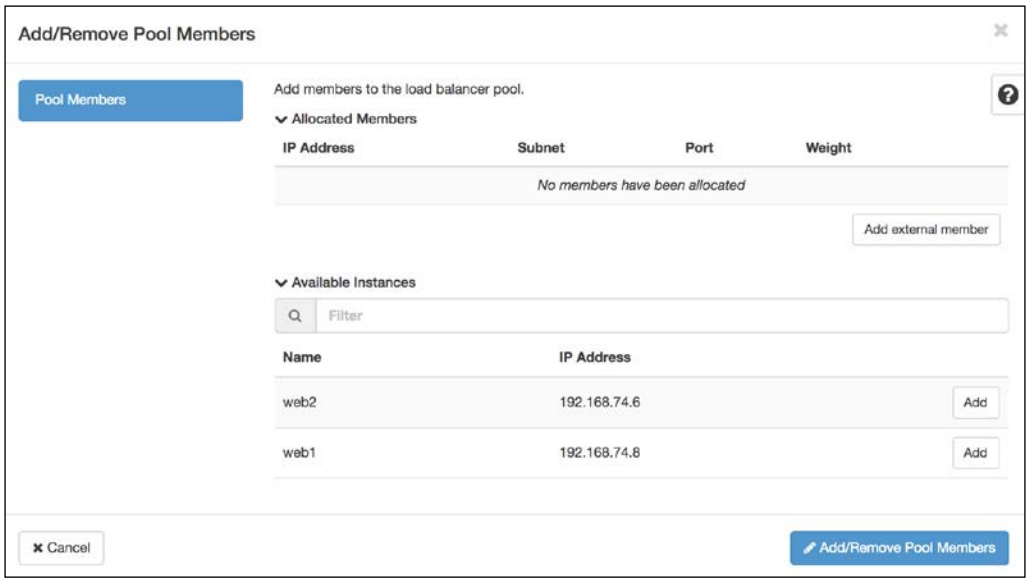
4. Clicking on the pool ID brings us to the pool details page. Remove all pool members by clicking on the **Add/Remove Pool Members** button:



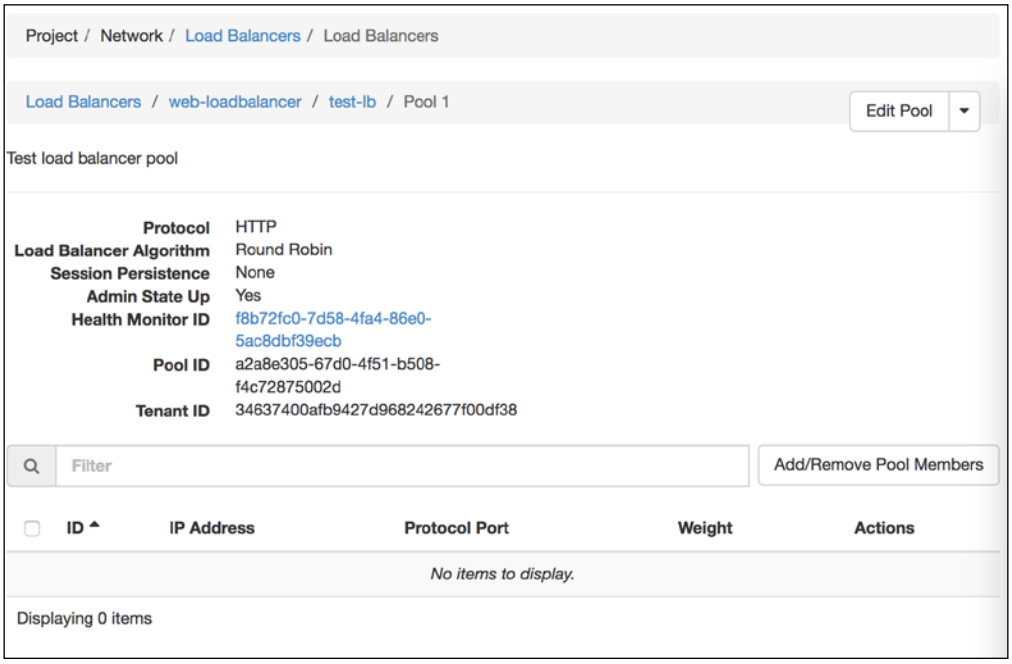
5. This brings up a new screen for removing pool members. Remove all pool members from the pool by clicking on the **Remove** button next to each member:



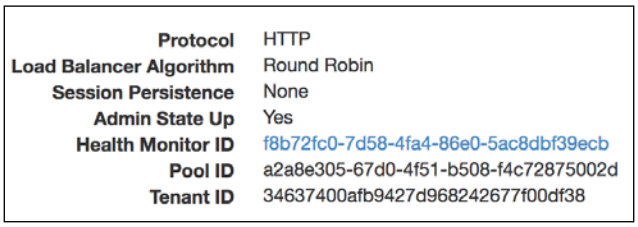
6. Click on the **Add/Remove Pool Members** button to save the changes:



7. After deleting pool members, we will get back to the pool details page. We need to delete all health monitors in order to be able to delete the pool:

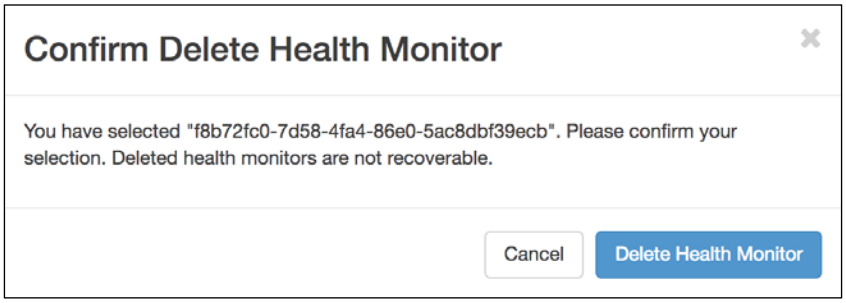


8. Click on the **Health Monitor ID** to access the health monitor details:

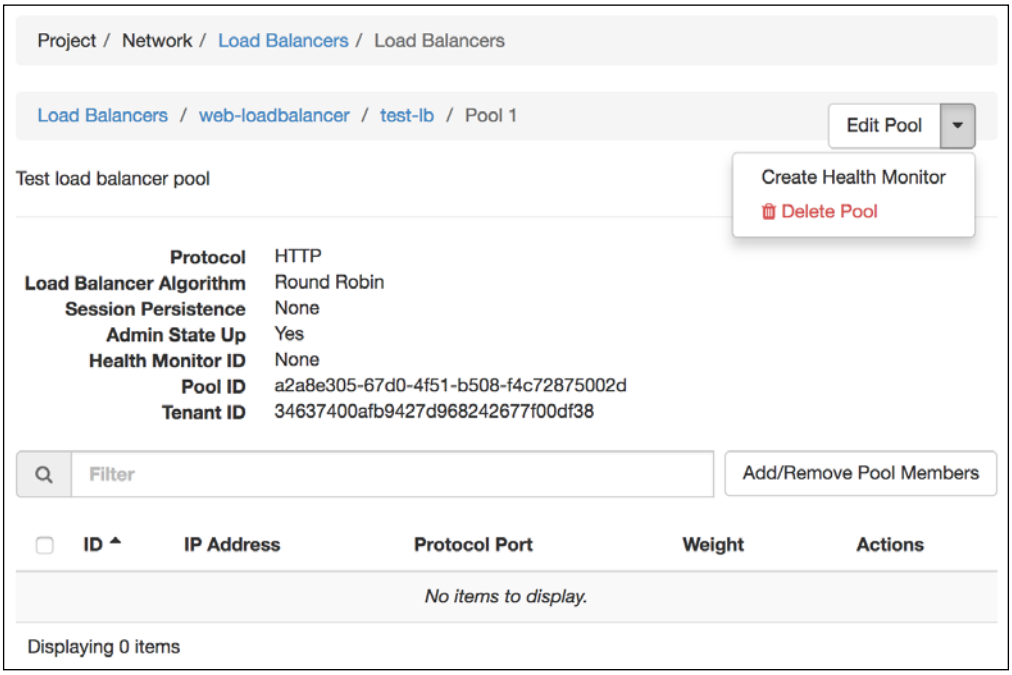


9. From the health monitor details page, click on the drop-down arrow next to the **Edit** button to reveal the **Delete Health Monitor** option. Click on the **Delete Health Monitor** option to delete the health monitor:

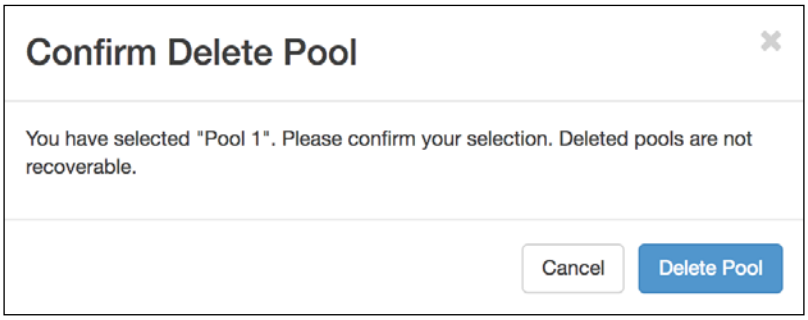
10. This will give you a confirmation screen. Click on the **Delete Health Monitor** button to confirm:



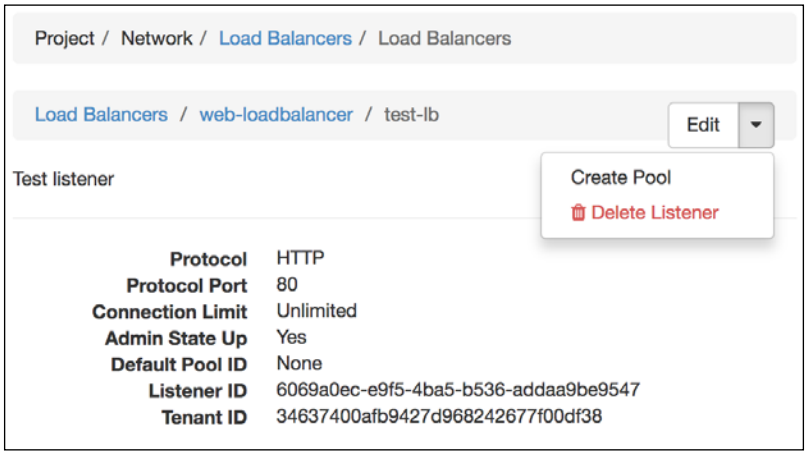
11. After deleting the health monitor, we get back to the pool screen. Click on the action arrow next to the **Edit Pool** button to reveal the **Delete Pool** option. Click on the **Delete Pool** option:



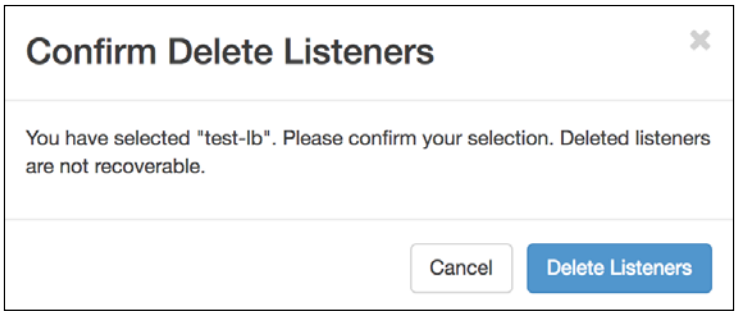
12. You will be presented with a confirmation screen. Click on the **Delete Pool** button to confirm pool deletion:



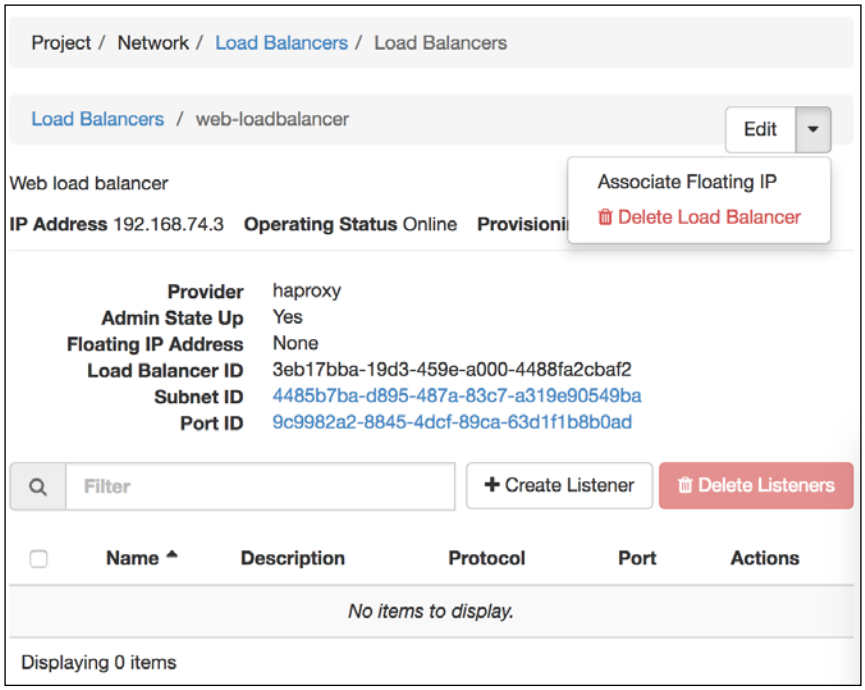
13. After deleting the pool, we are back on the listener details page. Click on the action arrow next to the **Edit** button to reveal the **Delete Listener** option. Click on the **Delete Listener** option to delete the listener:



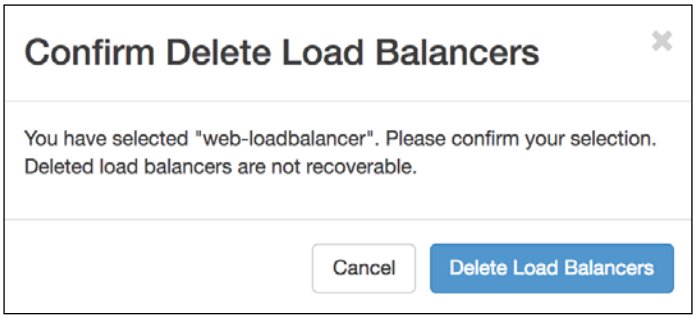
14. You will be presented with the confirmation screen; click on the **Delete Listeners** button to confirm listener deletion:



15. After deleting the listener, we will be back on the load balancer details page. Click on the action arrow next to the **Edit** button to reveal the **Delete Load Balancer** option. Click on the **Delete Load Balancer** option to delete the load balancer:



16. You will be presented with the confirmation screen. Click on the **Delete Load Balancers** button to delete the load balancer:



How it works...

We created a load balancer pool with a health monitor and added two instances to it. To do this, we executed the following steps:

1. Creating a load balancer from the load balancer page.
2. Selecting subnet to which all the nodes are attached when creating listener.
3. Adding members to the listener.
4. Adding a health monitor.

All the load balancer details can be edited after creation. Deleting a load balancer requires deletion of all the assets that were created: pool, listener, and health monitor.

Using OpenStack Dashboard with OpenStack Orchestration

Heat is the OpenStack Orchestration engine that enables users to quickly spin up whole environments using templates. Heat templates, known as **HOT** (**Heat Orchestration Templates**) are **YAML** (**Yet Another Markup Language**) based files. The files describe the resources being used, and the type and size of the instances, and the network an instance will be attached to, among other pieces of information required to run that environment.

In *Chapter 9*, *OpenStack Orchestration Using Heat and Ansible*, we showed you how to use the Heat command-line client. In this section, we will show how to use an existing Heat template file in OpenStack Dashboard to spin up two web servers running Apache, connected to a third instance running HA Proxy.

Getting ready

Load a web browser, point it to our OpenStack Dashboard address at `http://192.168.100.117/`, and log in as a user in the `default` domain, such as the `developer` user, created in the *Common OpenStack identity tasks* recipe in *Chapter 2*, *The OpenStack Client*, with the `cookbook4` password. The URL for our dashboard is same as the public load balancer IP as discussed in *Chapter 1*, *Installing OpenStack with Ansible*. If you need to find out at what URL your Horizon is, use public IP from the OpenStack catalog list as described in *Chapter 3*, *Keystone – OpenStack Identity Service*.

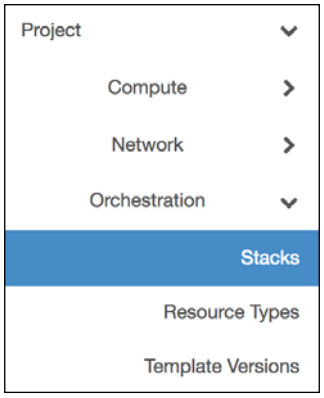
How to do it...

This recipe will walk you through the launching and deleting of orchestration stacks using the OpenStack Dashboard.

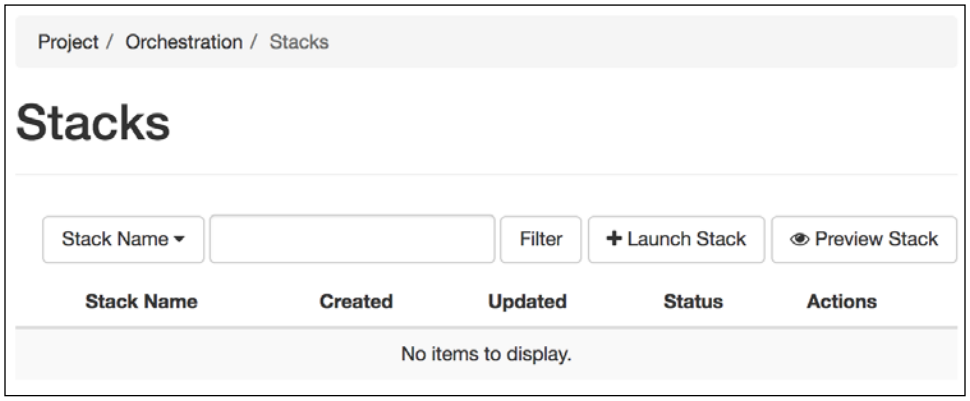
Launching stacks

To launch a Heat stack for a logged in user, carry out the following steps:

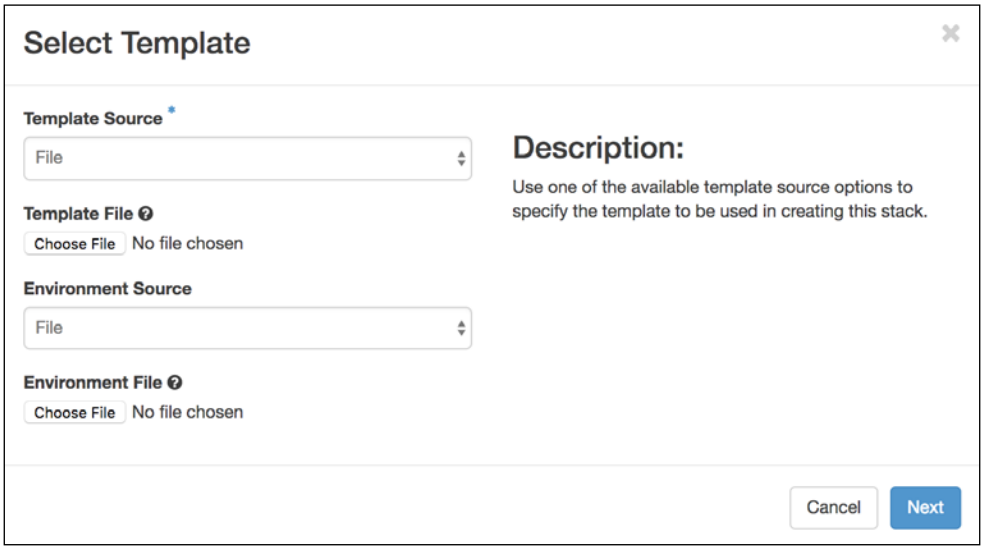
1. To view available Heat stacks within our OpenStack Dashboard, select the **Stacks** tab from the **Orchestration** menu, as shown in the following screenshot:



2. After clicking on the **Stacks** tab, you will see all running stacks in your environment. In our case, our list is empty, as shown here:



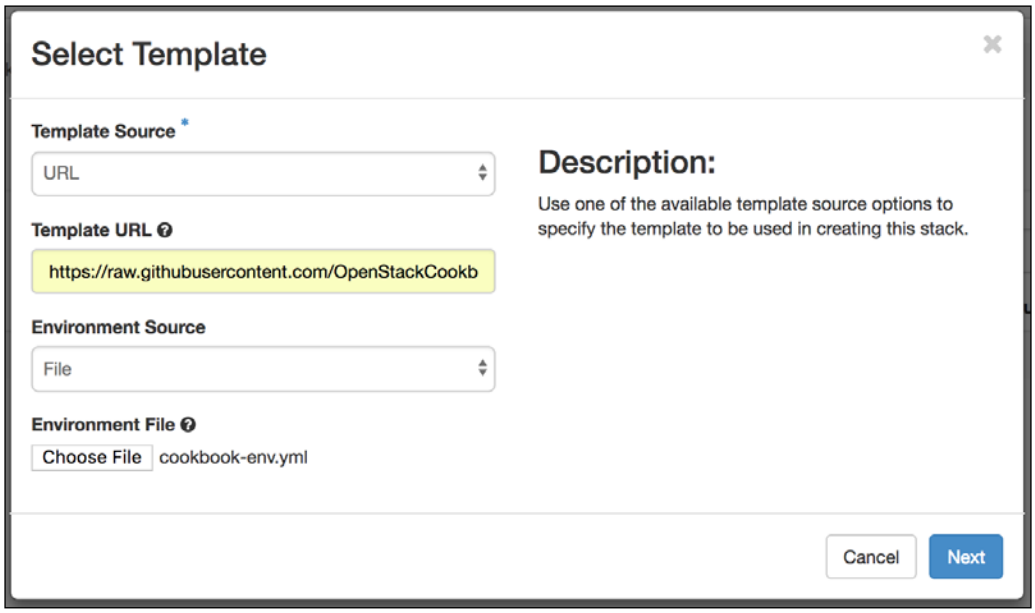
3. Click on the **Launch Stack** button to create a new stack. You will see the following window:



4. There are several ways to specify what **Template Source** to use in a stack: **File**, **Direct Input**, or **URL**. Choose which option is the most convenient for you. For our example, you can either use the URL directly, or upload a file. A template file can be downloaded from here: `https://raw.githubusercontent.com/OpenStackCookbook/OpenStackCookbook/master/cookbook.yaml`. We will use the **URL** option and use the link directly in the template.
5. Just like the template source file can be uploaded, we can also upload the **Environment Source** file. In this case, we do not have to use environment source, but it makes the process convenient. The environment file stores the values we would have to enter into the browser manually, but instead loads the values for us on the **Launch Stack** screen, as shown in step 8. In our example, we are using an environment file that can be downloaded from here: `https://raw.githubusercontent.com/OpenStackCookbook/OpenStackCookbook/master/cookbook-env.yaml`. Update the `public_net_id`, `private_net_id`, and `private_subnet_id` fields to match your environment.

If you are not sure where to find network information, refer to the *Using OpenStack Dashboard to manage Neutron networks and routers* recipe.

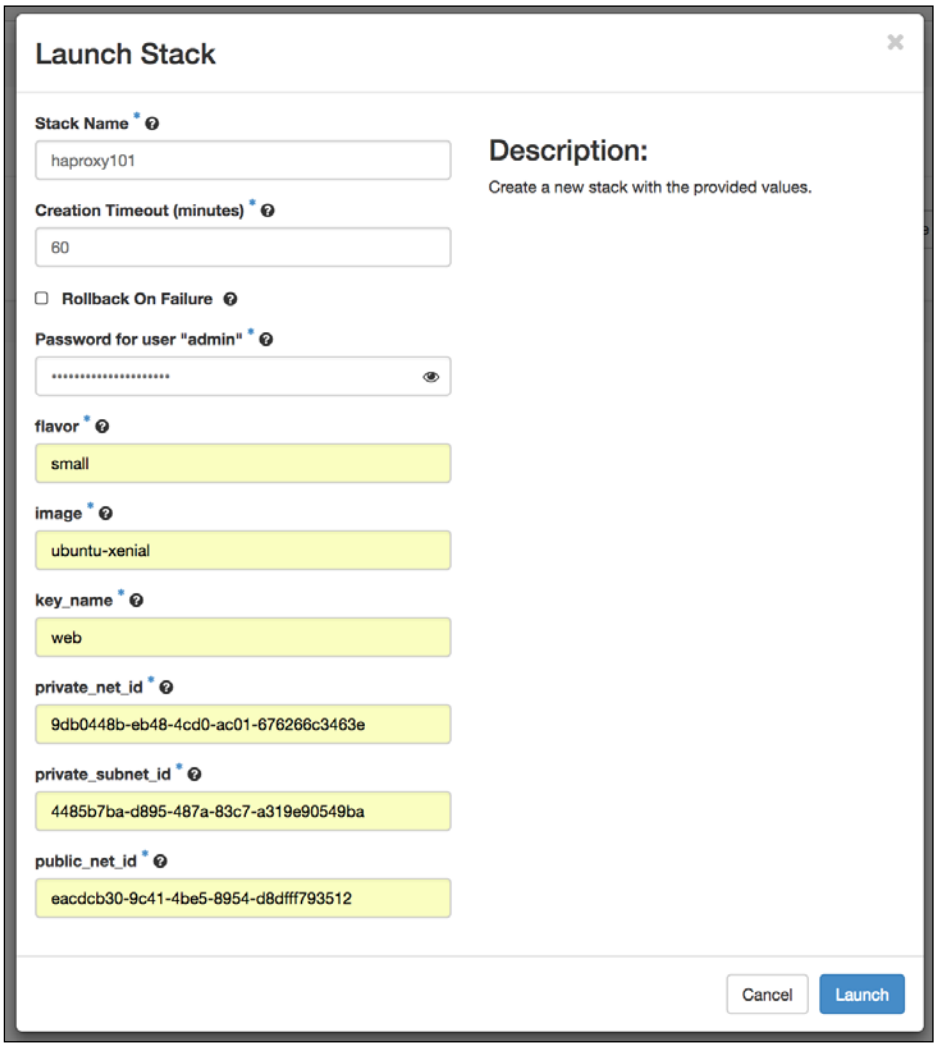
6. After selecting the **Template Source** and **Environment Source** files, click on **Next**:



7. Our sample environment file contains the following:

```
parameters:
  key_name: web
  image: ubuntu-xenial
  flavor: small
  public_net_id: eacdcb30-9c41-4be5-8954-d8dfff793512
  private_net_id: 9db0448b-eb48-4cd0-ac01-676266c3463e
  private_subnet_id: 4485b7ba-d895-487a-83c7-a319e90549ba
```

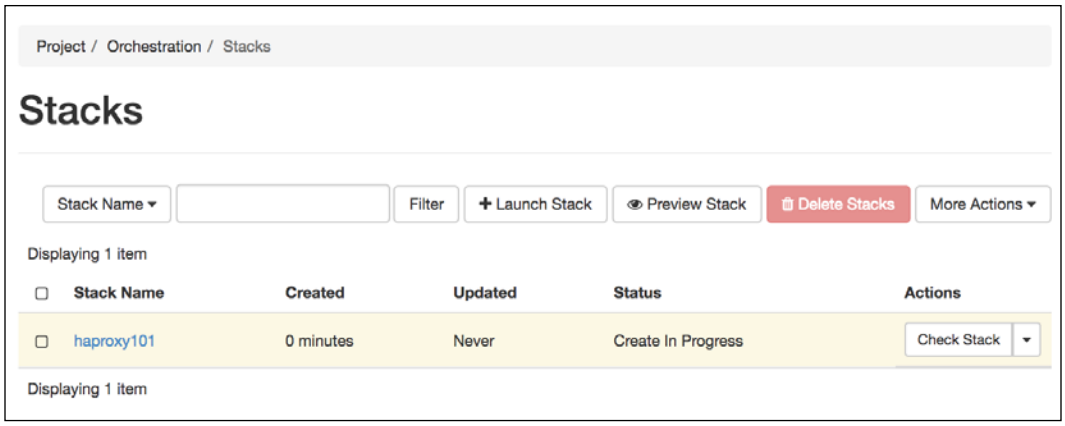
8. Clicking on **Next** will give you a **Launch Stack** window with all the inputs:



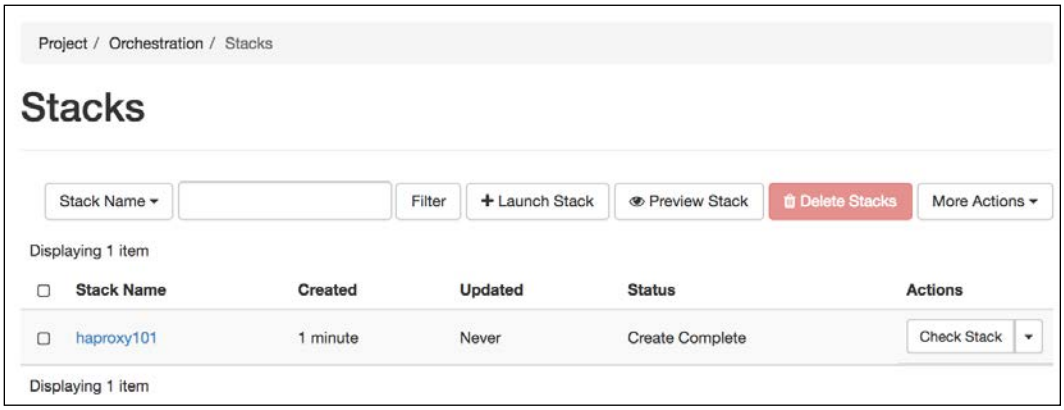
9. Note that most of the inputs in our template are now populated. If you did not specify an environment source file in the previous step, you will need to enter the `key_name`, `image`, `flavor`, `public_net_id`, `private_net_id`, and `private_subnet_id` fields.

These fields are specific to each template used. Your templates may have different fields.

10. Enter stack name and user password for your user. If you are logged in as admin or demo, the password is `openstack`.
11. Click on **Launch** to start stack creation. If all inputs were correct, you should see your stack being created, similar to the following example:



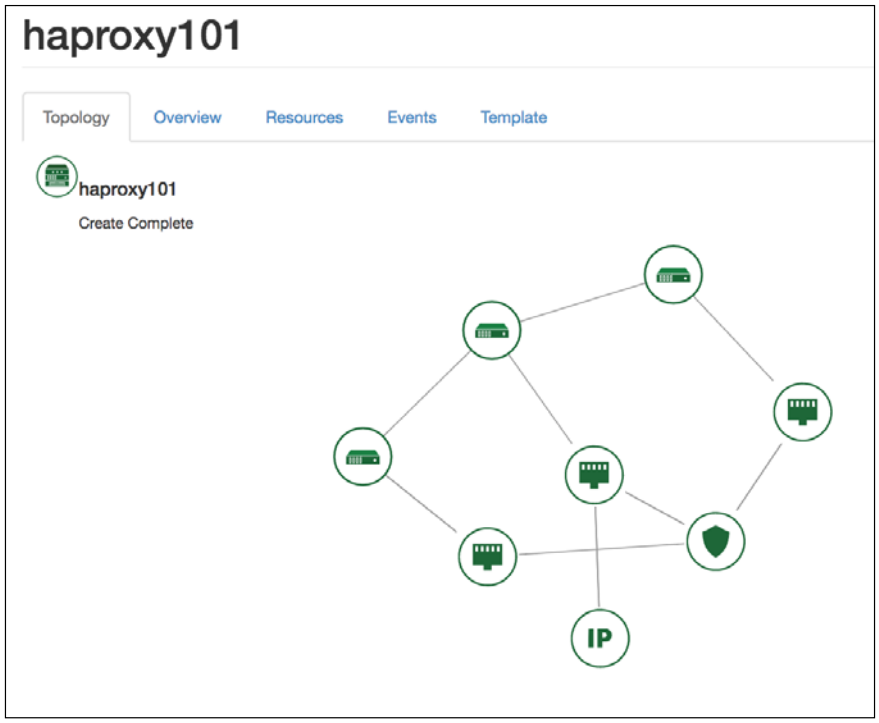
12. After the stack finishes creating and if there were no errors during creation, you will see your stack's status updated to **Create Complete**:



Viewing stack details

After launching a stack, there is a lot of information associated with each, including inputs, outputs, and in case of errors, information of why stack creation failed.

1. To view the details of the stack, click on the stack name from the stacks list. The first available view is **Topology**:



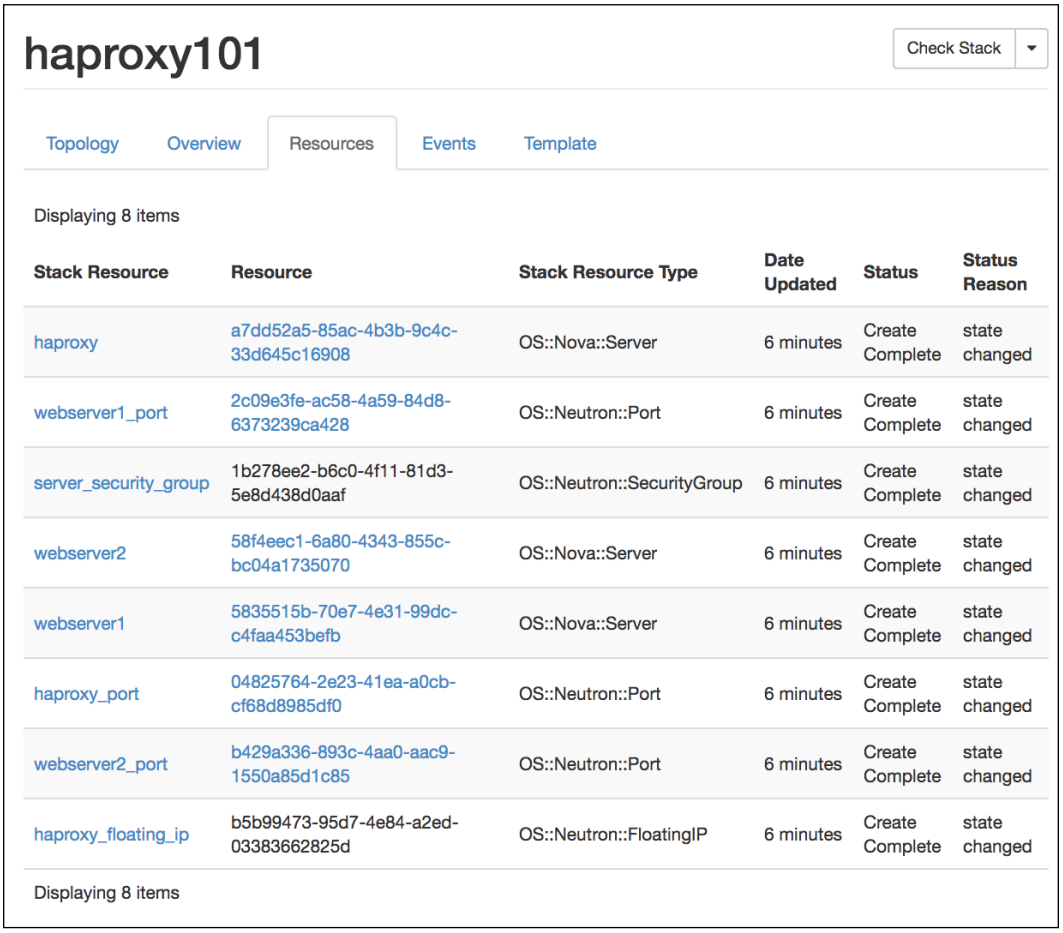
Explore the topology by clicking on the nodes. If the graph does not fully fit or you would like a different perspective, you can drag the graph around the window.

2. The next tab is **Overview**, it will provide all of the information that was used in creating the stack:

Stack information available in the **Overview** tab is as follows:

- Info
- Status
- Outputs
- Stack parameters
- Launch parameters

3. The **Resources** tab will show all the HEAT resources that were created during stack launch:



haproxy101

Check Stack

Topology | Overview | Resources | Events | Template

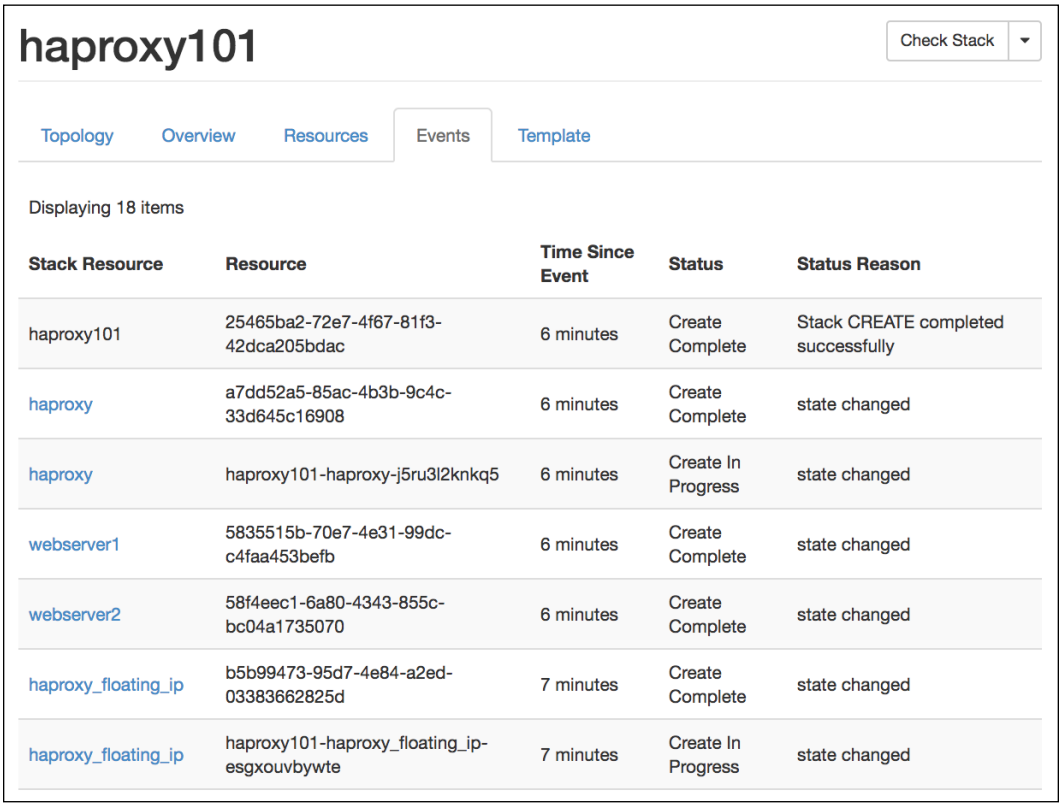
Displaying 8 items

Stack Resource	Resource	Stack Resource Type	Date Updated	Status	Status Reason
haproxy	a7dd52a5-85ac-4b3b-9c4c-33d645c16908	OS::Nova::Server	6 minutes	Create Complete	state changed
webserver1_port	2c09e3fe-ac58-4a59-84d8-6373239ca428	OS::Neutron::Port	6 minutes	Create Complete	state changed
server_security_group	1b278ee2-b6c0-4f11-81d3-5e8d438d0aaf	OS::Neutron::SecurityGroup	6 minutes	Create Complete	state changed
webserver2	58f4eec1-6a80-4343-855c-bc04a1735070	OS::Nova::Server	6 minutes	Create Complete	state changed
webserver1	5835515b-70e7-4e31-99dc-c4faa453befb	OS::Nova::Server	6 minutes	Create Complete	state changed
haproxy_port	04825764-2e23-41ea-a0cb-cf68d8985df0	OS::Neutron::Port	6 minutes	Create Complete	state changed
webserver2_port	b429a336-893c-4aa0-aac9-1550a85d1c85	OS::Neutron::Port	6 minutes	Create Complete	state changed
haproxy_floating_ip	b5b99473-95d7-4e84-a2ed-03383662825d	OS::Neutron::FloatingIP	6 minutes	Create Complete	state changed

Displaying 8 items

If there were any errors during stack launch, check this page to see which component's creation failed.

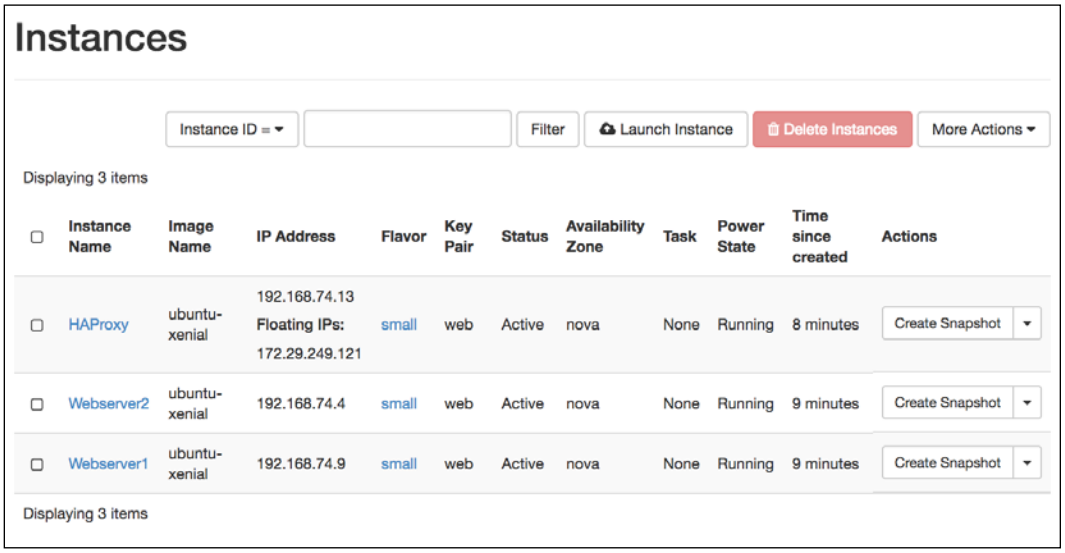
4. The **Events** tab shows all the events that occurred when the stack was created. This page can also be very helpful in troubleshooting Heat templates.



Stack Resource	Resource	Time Since Event	Status	Status Reason
haproxy101	25465ba2-72e7-4f67-81f3-42dca205bdac	6 minutes	Create Complete	Stack CREATE completed successfully
haproxy	a7dd52a5-85ac-4b3b-9c4c-33d645c16908	6 minutes	Create Complete	state changed
haproxy	haproxy101-haproxy-j5ru3l2knkq5	6 minutes	Create In Progress	state changed
webserver1	5835515b-70e7-4e31-99dc-c4faa453befb	6 minutes	Create Complete	state changed
webserver2	58f4eec1-6a80-4343-855c-bc04a1735070	6 minutes	Create Complete	state changed
haproxy_floating_ip	b5b99473-95d7-4e84-a2ed-03383662825d	7 minutes	Create Complete	state changed
haproxy_floating_ip	haproxy101-haproxy_floating_ip-esgxouvbywte	7 minutes	Create In Progress	state changed

5. The **Template** tab will show the template that was used to create the stack.

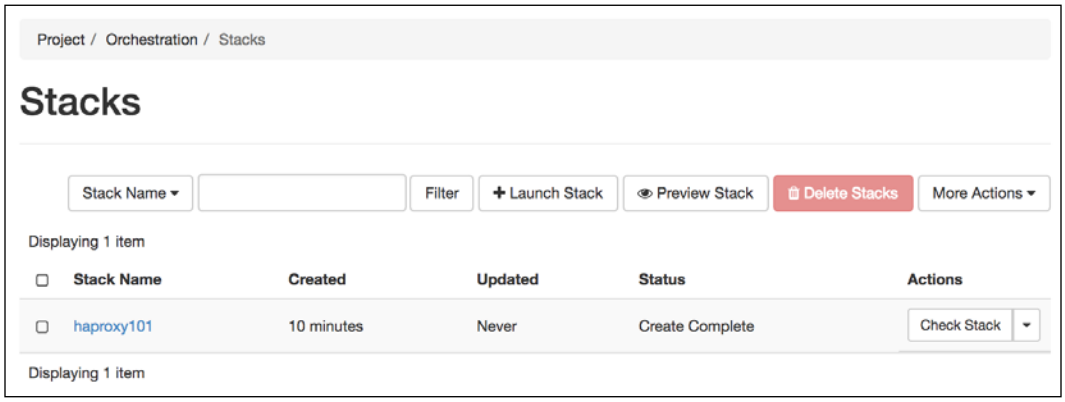
6. While your Heat stack is running, you can also see how many instances it created in the **Instance** option of the **Compute** tab. This is what our instances look like on the **Instances** page:



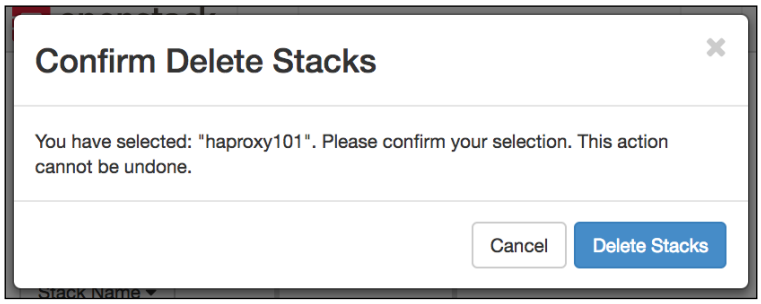
Deleting stacks

Stack deletion is simple, however note that, when deleting, it will delete all resources that were created during stack launch.

1. To delete a stack, first view available stacks on the Stacks page:



2. Select the stack to delete and click on the **Delete Stacks** button to delete a stack. You will be asked to confirm the deletion:



3. After confirming deletion, all resources associated with the stack will be deleted.

How it works...

We used the OpenStack Dashboard to launch, view, and delete Orchestration stacks. We first needed to download a sample HAProxy **Heat Orchestration Template** (**HOT**) from GitHub. Since we were using an environment file, we also had to modify appropriate inputs. Your own templates may have different inputs.

After launching our HAProxy stack, we explored its topology, resources, and events. Resources created during stack launch will also be reflected in the rest of your environment. If you are launching new instances, all of them will also be available on the **Instances** page. Delete and modify resources created during the stack launch only through the Orchestration section in OpenStack dashboard or on command line. Deleting stacks through the dashboard will delete all associated resources.

Another Book You May Enjoy

If you enjoyed this book, you may be interested in another book by Packt:

OpenStack for Architects

Michael Solberg, Ben Silverman

ISBN: 978-1-78439-510-0

- Familiarize yourself with the components of OpenStack
- Build an increasingly complex OpenStack lab deployment
- Write compelling documentation for the architecture teams within your organization
- Apply Agile configuration management techniques to deploy OpenStack
- Integrate OpenStack with your organization's identity management, provisioning, and billing systems
- Configure a robust virtual environment for users to interact with
- Use enterprise security guidelines for your OpenStack deployment
- Create a product roadmap that delivers functionality quickly to the users of your platform

Leave a review – let other readers know what you think

Please share your thoughts on this book with others by leaving a review on the site that you bought it from. If you purchased the book from Amazon, please leave us an honest review on this book's Amazon page. This is vital so that other potential readers can see and use your unbiased opinion to make purchasing decisions, we can understand what our customers think about our products, and our authors can see your feedback on the title that they have worked with Packt to create. It will only take a few minutes of your time, but is valuable to other potential customers, our authors, and Packt. Thank you!

Index

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y