Community Experience Distilled

Learning OpenStack High Availability

Build a resilient and scalable OpenStack cloud using advanced open source tools

Rishabh Sharma

[PACKT] open source
community experience distilled
PUBLISHING

Learning OpenStack High Availability

Build a resilient and scalable OpenStack cloud, using advanced open source tools

Rishabh Sharma

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Learning OpenStack High Availability

First published: November 2015

Production reference: 1251115

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

ISBN 978-1-78439-570-4

www.packtpub.com

Credits

Author
Rishabh Sharma

Reviewers
Michael A Cossenas
Vinoth Kumar Selvaraj

Commissioning Editor
Kunal Parikh

Acquisition Editor
Nadeem Bagban

Content Development Editor
Rashmi Suvarna

Technical Editor
Utkarsha S. Kadam

Copy Editor
Kausambhi Majumdar

Project Coordinator
Judie Jose

Proofreader
Safis Editing

Indexer
Tejal Daruwale Soni

Graphics
Abhinash Sahu

Production Coordinator
Aparna Bhagat

Cover Work
Aparna Bhagat

About the Author

Rishabh Sharma is currently working as a chief technology officer (CTO) at JOB Forward, Singapore. Prior to working for JOB Forward, he worked for Wipro Technologies, Bangalore, as a solution delivery analyst. He was involved in research projects of cloud computing, proof of concepts (PoC), infrastructure automation, big data solutions, and various giant customer projects related to cloud infrastructure and application migration.

In a short span of time, he has worked on various technologies and tools such as Java/J2EE, SAP(ABAP), AWS, OpenStack, DevOps, big data, and Hadoop. He has also authored many research papers in international journals and IEEE journals on a variety of issues related to cloud computing.

He has authored five technical books until now. He recently published two books with international publications:

Learning Chef (`https://www.packtpub.com/networking-and-servers/learning-chef`).

Cloud Computing: Fundamentals, Industry Approach and Trends (`http://www.wileyindia.com/cloud-computing-fundamentals-industry-approach-and-trends.html`).

He is also an open source enthusiast and writes for the Open Source For You (OSFY) magazine. You can get in touch with him at `ater.rishabh.sharma@gmail.com`.

I would like to give special gratitude to my spiritual guru for his guidance and blessings. I am very grateful to my family for its support and encouragement during this project. I would like to give my special thanks to friend Mr. Balachandar Raju for assisting me in this project.

I am very thankful for PACKT publishing to provide me this opportunity to present this book and for his valuable support and guidance during this endeavor. Reader's views, comments and suggestions are welcome.

About the Reviewers

Michael A Cossenas is a Linux/Network adminstrator from Athens, Greece.

He's been a network security specialist working for Digital Sima, a company specialized in LAN/WAN networking. He is now employed as a subcontractor for IBM Greece in the SO (Strategic Outsourcing) department. Here, he manages 50+ SUSE-based Linux servers for one of their customers.

His first experience with Linux was back in 1998, when he used Red Hat Linux 5.2. Since then, Michael has worked on various open source projects, including Zimbra, Distributed Replicated Block Device (DRBD), Kernel-based Virtual Machine (KVM), and Postfix.

He is also an openvpn forum moderator.

I would like to thank my family (my wife, Froso, my son, Antony, and my daughter, Kate) for supporting me in these difficult times in Greece.

Vinoth Kumar Selvaraj is working as an OpenStack engineer at Cloudenablers, a Cloud Technology start-up based in Chennai, India. At Cloudenablers, he takes care of setting up private and hybrid clouds for internal/external customers. He also leads the CloudLab initiatives, which involve the exploration and integration of various products and tools with the latest versions of OpenStack.

Also, he worked as a reviewer for the book *Openstack Cloud Security*.

In his spare time, Vinoth enjoys sharing his insights on technologies at `http://www.hellovinoth.com`.

I would like to thank my Amma, Appa, Anna, and my friends for their love and support.

My special thanks to Monisha—my confidante, Rathinasabapathy—my inspiration, Vinu Francis—my Guru, JayaPrakash—my mentor, and Beny Raja—my instructor.

---

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit `www.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. Get in touch with us at `service@packtpub.com` for more details.

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.

`https://www2.packtpub.com/books/subscription/packtlib`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and readPackt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

OpenStack is a set of software tools and packages used to build private and hybrid cloud computing platforms. It is one of the most popular and widely adopted open source software managed by OpenStack Foundation. Since its introduction in 2010, a huge number of reputed industries, including Red Hat, Intel, HP, IBM, AMD, Canonical, and many others, support OpenStack.

The most promising feature of OpenStack is that it provides a DIY(Do-It-Yourself) approach to cloud computing, and it can easily embrace the new development features. Therefore, companies such as Google and Facebook created their own data storage and cloud services using OpenStack.

High availability typically means achieving 99.99% availability, and basically this can be possible by removing all single point of failure (SPOF), which is also applicable for OpenStack. This book covers all the basic and advance approaches related to achieving high availability in OpenStack with detailed step-by-step explanations, hands-on exercises, and screenshots. These provide you with a real-time practical understanding of implementing high availability in OpenStack. Some customer case studies are also included:

`http://www.colocationamerica.com/blog/core-advantages-open-stack-for-iaas.htm`.

`http://www.tomsitpro.com/articles/openstack-costs-benefits,2-684-2.html`.

What this book covers

Chapter 1, *An Introduction to High Availability Concepts*, introduces the reader to a basic understanding of high availability in a production environment. The requirements and common design patterns will be explained in details; architectural choices to maximize availability have been critically discussed.

Chapter 2, *Database and Messaging Services*, explains how to obtain resilient and available OpenStack supporting services, such as database (MariaDB with Galera clustering) and RabbitMQ with replicated queues.

Chapter 3, *Load Balancing for Active/Active Services*, delves in to the the basic and advanced topics of network load balancing and explains with detailed examples, the configurations of HAProxy and keepalived.

Chapter4, *Clustering, Fencing, and Active/Passive Services*, describe the services in OpenStack that are still not fully stateless, and thus require classical clustering methods such as Pacemaker. This chapter will analyze in depth the construction of a cluster of services with their resource agents and dependencies.

Chapter 5, *Highly Available OpenStack Services*, deals with the scaling and resiliency of all the basic OpenStack services: compute, image, storage, object storage, and dashboard.

Chapter 6, *Distributed Networking*, delves into the details of Neutron Distributed Virtual Routers, multiple L3 agents in active/passive configuration, and third-party networking drivers that offer high availability options. The OpenStack Networking services are the most difficult to scale and render highly available.

Chapter 7, *Shared Storage*, describes how shared storage is a fundamental requirement for high availability and for a quick recovery from a failure (evacuating failed nodes with live migration). This chapter presents different options to provide shared storage to OpenStack and explains their configuration and setup to some extent.

Chapter 8, *Failure Scenario and Disaster Recovery*, focuses on how for any OpenStack operator who strives to bring the cloud they manage to higher availability, understanding the different ways that a failure can occur and how these can impact the performance and availability of the service is a fundamental skill. This chapter analyzes different failure scenarios and proposes solutions to provide a swift and effective recovery to a normal operational level.

Chapter 9, *The Principles of Design for Highly Available Applications*, explains how having a highly available cloud might not be enough, if the application running on top of it doesn't take advantage of the principles and concepts of a resilient design. This chapter mainly explains how correct application design can help improve reliability and uptime of end user services; particular focus has been dedicated to microservice architectures and distributed web applications.

Chapter 10, *Monitoring for High Availability*, covers how the control and maintenance of a cloud is of the utmost importance; visibility in operations and alerting on failures are the basis for correct functioning and quick recovery in case of unexpected outages or planned maintenance windows. The chapter introduces a few key concepts and tools to correctly measure and control the operations of an OpenStack cloud.

Chapter 11, *Use Cases and Real-World Examples*, covers a number of real-world examples of HA deployments. The relevant lessons for a design of similar resilient clouds have been distilled and are presented to the reader

What you need for this book

This book assumes that you are aware of the fundamental concepts of cloud computing and high performance computing.

A basic understanding of Linux administration and commands are beneficial here.

A hands-on experience of the installation of different software or packages on a Linux-based OS is essential to the installation of OpenStack's packages and to execute various commands.

The approach adopted in this book uses OpenStack 2014.1 (Icehouse) version, which requires at least a 700 MHz processor (Intel Celeron or better), 8GB RAM (system memory), and 120GB of hard-drive space. The preferred OS is Ubuntu 12.04 and higher.

Furthermore, you will need an Internet access to download the software packages that you do not already have.

Who this book is for

This book is for OpenStack administrator, cloud administrator, cloud engineer, or cloud developer with some real time understanding of cloud computing, OpenStack and familiarity with Linux command is essential to start with this book.

Conventions

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

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "We can include other contexts through the use of the `include` directive."

A block of code is set as follows:

```
primitive p_ip_mysqlocf:heartbeat:IPaddr2 \
paramsip="192.168.1.32" cidr_netmask="24" \
  op monitor interval="30s"
```

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

```
Sudo rabbitmqctl cluster status
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "The default value of the **ENABLED** attribute value is changed to 1."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

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

Downloading the example code

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

Downloading the color images of this book

We also provide you with a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from `https://www.packtpub.com/sites/default/files/downloads/5704OS.pdf`.

Errata

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

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

Piracy

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

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

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

Questions

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

1
An Introduction to High Availability Concepts

Over the past couple of years, cloud computing has made a significant impact in transforming IT from a niche skill to a key element of enterprise production environments. From an **Infrastructure as a Service** (**IaaS**) point of view, cloud computing is much more advanced than mere virtualization; various industries and online businesses have started moving test, staging, and productions scenarios roles to IaaS and started replacing traditional dedicated resources with on-demand resource models.

OpenStack is one of the most popular and commonly used open source cloud computing platforms, and it is mainly used to deploy infrastructure as a service solution. Enabling high availability in OpenStack is a required skill for cloud administrators and cloud engineers. This chapter will introduce you to high availability concepts, a way of measuring and achieving high availability through architectural design in OpenStack.

In this chapter, we will cover the following topics:

- What does high availability mean?
- How to measure high availability
- Architecture design for high availability
- High availability in OpenStack

What does High Availability (HA) mean?

The basic understanding of high availability in the IT world is when any system continuously operates (100 percent operational) without any down time despite occurrences of failure in hardware, software, and application.

How to measure high availability

In order to measure a system's high availability, we usually check the total duration of the uptime of the system. For example if system availability is 99 percent, it means that the system is operational for 8672.4 hours throughout the year because the total hours in a year is 8760.

The following formula is used to calculate the total availability, and availability can be increased using high availability techniques to increase MTTF and decrease MTTR that we are going to discuss in this book with respect to OpenStack .Let's understand these term in more detail:

- **Mean time between failures** (**MTBF**): MTBF tells us the estimated time between two frequent failures within a process or a component, which can be repairable.
- **Mean time to failure** (**MTTF**): MTTF is the total estimated time of a system where repairing is not possible.
- **Mean time to repair or replace** (**MTTR**): MTTR is the average estimated time to repair a failed component or the total replacement time of a failed component.

The formula to calculate availability is this:

Availability = MTTF / (MTTF + MTTR)

The following table represents the **Service Level Agreement** (**SLA**) for high availability between a consumer and a provider:

The level of availability	Downtime/day	Downtime/month	Downtime/year
one 9s(90%)	144.00 minutes	72 hours	36.5 days
two 9s(99%)	14.40 minutes	7 hours	3.65 days
three 9s(99.9%)	86.40 seconds	43 minutes	8.77 hours
four 9s(99.99%)	8.64 seconds	4 minutes	52.60 minutes
five 9s(99.999%)	0.86 seconds	26 seconds	5.26 minutes

The following are some commonly used terms that are used to understand and measure high availability:

- **Single point of failure** (**SPOF**): It reflects a part or a particular component of a system that brings the entire system to a halt; when it fails, the whole system will stop working. The consideration for a possibly single point of failure identifies the critical components of a complex system that would cause a total system failure in case of any breakdown.
- **Recovery time objective** (**RTO**): It is a very important matrix to measure business continuity. RTO determines the tolerance of a business process when a system is not available. For high user traffic e-commerce websites and mission critical businesses, RTO should be zero or near to zero.
- **Recovery point objective** (**RPO**): RPO analysis is a very critical metric for any business. It is calculated as a total amount of data loss when a system is not available. It is measured in terms of time.
- **Service level agreement** (**SLA**): SLA is a mutual agreement between consumers and a service provider that defines detailed service offerings, delivery time, the **quality of service** (**QoS**), and the scope or constraints of the offered services.

Common content in the contract

Metrics related to performance assurance depend on the following components:

- RTO and RPO
- Uptime and downtime ratio
- System throughput
- Response time

How to achieve high availability

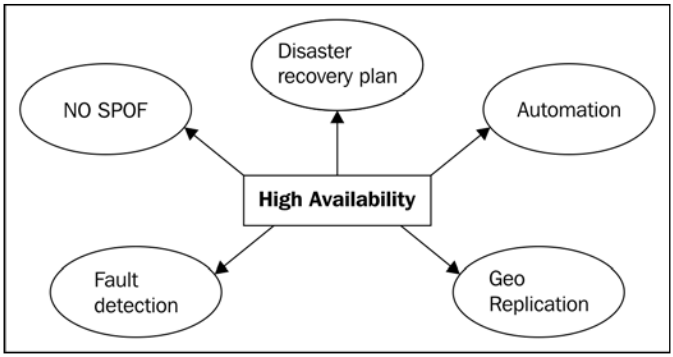
High availability in a production environment is achieved by removing single point of failures, enhancing replication, redundancy, and fail-over capability of the system, and quickly detecting failures as they occur. High availability can be achieved at many different levels including the infrastructure level, data center level, geographic redundancy level, and even application level. The basic understanding of high availability in the infrastructure level includes the following things:

- Multiple web servers
- Multiple database servers

- Multiple load balancers
- Leveraging different storage systems

Here, multiple web servers means multiple web nodes, multiple load balancers means active/passive load balancers, multiple database servers means replicated DB servers, and leveraging different storage system means the maximum utilization of all possible storage solutions with a backup plan that provides redundancy at each layer of the configuration.

For advanced HA configurations, we also need to plan about automatic failover and geo-replication, in case of disaster recovery, and also need to design our application for high availability. In short, we can say that there are various advanced techniques to achieve high availability, which we will discuss in detail in the following chapters, but the prime objective of each technique is to obtain the following five principals of high availability as described in this figure:



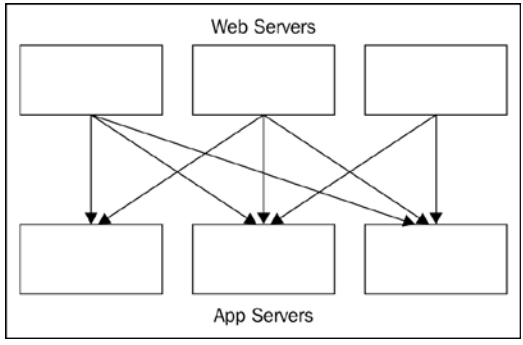
High availability principles

Architecture design for high availability

The following are some common design patterns used to design a high availability architecture. Let's discuss this in detail:

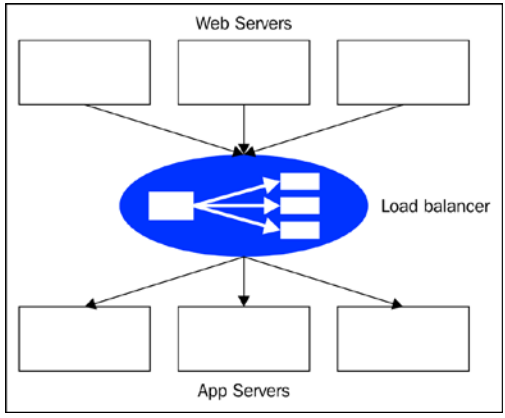
- **Design for failure consideration**: This must be the first and foremost concern of a cloud architect. Whenever any company or organization decides to move to a cloud infrastructure solution, it has to plan for failure. If the failure conditions are planned properly, then this will be of little or no consequence to the HA services or the resources, and the system will always be available.

- **Decouple your components**: All the components in a cloud infrastructure should be decoupled and isolated from each other; for example, whenever we have set upload balancing applications, traditionally web servers have been tightly coupled with the application server, but this is not the best practice. Isolation and decoupling are necessary between components. The following figure shows that each web server and application server is highly coupled:



High coupling

Web servers and application servers can be loosely coupled and isolated by putting a load balancer between them, as shown in the following figure. Any web or application server can be easily scaled up or scaled down without any dependency.



Decoupled applications

- **Build security in every layer**: While designing a cloud infrastructure, building security in each layer is recommended. As security is the shared responsibility of the consumer and the cloud provider, the following are some steps that a consumer must take to ensure security:
    - Enforce the principle of least privileges when developing applications
    - Encrypt data during transitions state
    - Try to use a multiway and multifactor authentication
- **Design parallel**: While designing a cloud infrastructure, a parallel architecture that is fast and efficient should be implemented. Two possible approaches in parallel architecture could be as follows:
    - A server works on a job sequentially for 4 h
    - Four servers work on a job in parallel for 4 h

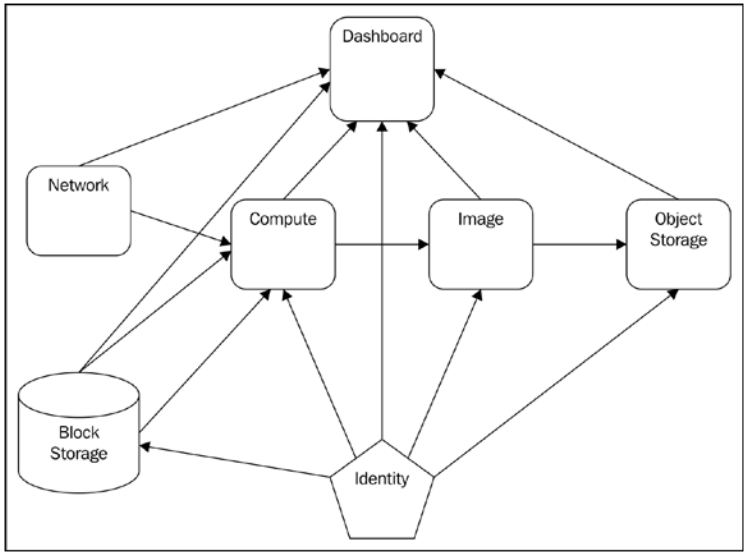
  There is no difference cost-wise, but the second approach is likely to be four times faster; therefore, a parallel design is highly recommended.
- **Automate and test everything**: When a production environment is enabled to handle cloud service failure, you should test your automated processes for the occurrence of hardware, software, and application failure. This kind of automation is provided by the cloud infrastructure service provider, which allows you to implement failover automated processes across instances, availability zones, regions, and other clouds. You should automate the data backups as well because in case of any outage or disaster, your data must be immediately ready.

  Any disaster recovery plan can only be successful if it is tested properly to make sure it works. It can be tested by disabling your various cloud servers and associated services and transferring high loads to your production servers so that you can test the actual potential of your existing infrastructure. Significantly, cloud infrastructure solutions provide easily configurable disaster recovery and backup services nowadays. Despite highly popular services, organizations cannot effectively enable high availability services in the cloud until they implement a proper architecture and use the right management tools.

High availability in OpenStack

Many organizations go with the OpenStack cloud nowadays to provide an infrastructure as a service environment since OpenStack has excellent hybrid cloud features and can leverage public cloud scale out feature through APIs. Therefore, OpenStack is quite capable of handling the workload of mission critical applications.

OpenStack provides an easy-to-control dashboard to manage networking, storage, and computing resources and facilitates users to have a provision for computing resources. The following figure illustrates the architecture of OpenStack:



An OpenStack component

Organizations that run their production applications over OpenStack, generally try to achieve five 9s (99.999) of availability. Hence, the failure of any running controller node or service should not create any kind of disruption on any application running on the resources provided or managed by OpenStack.

In short, achieving high availability in OpenStack means removing all single points of failure, implementing redundancy, and replicating component failover capability and automation. Also, workload should be scaled out or scaled in according to real-time workload. We can achieve high availability in OpenStack in the following ways:

- Enabling multimaster database replication and building an efficient and reliable messaging cluster that forms the basis of any highly available OpenStack deployment
- Setting up open source load balancing software such as HAProxy and keepalived and load balancing of HTTP REST API's, MySQL, and AMQP clusters
- Enabling some classical clustering methods such as a pacemaker with two or more nodes to control active/passive OpenStack services
- Enabling OpenStack services in stateless mode to offer a scalable cloud framework with scaling and resiliency of all the basic OpenStack services: compute, image, storage, object storage, and dashboard
- Leveraging third-party networking drivers that offer high availability options
- Enabling distributed networking with Neutron Distributed Virtual Routers and configuring multiple L3 agents in active/passive configuration and third-party networking drivers that offer high availability options
- Leveraging software-defined storage such as GlusterFS, Ceph, traditional enterprise storage such as NFS, iSCSI for quick recovery after a failure, and backup services
- Enabling automatic failover and geo-replication using swift and effective recovery technique such as network partitioning split brain

Summary

In this chapter, we learned the fundamentals of high availability and what a highly available design is meant to achieve in a production environment.

We have also learned about SPOF, RTO, RPO MTTR, MTTF, MTTF, the concept of SLA, and the general architectural design of a highly available system with an overview of OpenStack high availability requirements and the various ways of achieving high availability in OpenStack.

In the next chapter, we will take a deep dive into database replication and how to build an efficient and reliable messaging cluster to enable high availability in OpenStack.

2
Database and Messaging Services

In order to store all the information of OpenStack services, such as the state of a running instance, the availability of a computing node, and other instance-associated information, one central database service is used. As we learned in the first chapter, we should avoid **Single Point of Failure** (**SPOF**) to achieve high availability. Since DB is the central operational component of the OpenStack, it must be clustered to avoid SPOF in OpenStack.

Here, our objective is to achieve resiliency in case of database failures and enable OpenStack components to communicate with the database nodes. We'll also build an efficient reliable messaging RabbitMQ cluster.

In this chapter, we will cover the following topics:

- Installing MariaDB with Galera clustering
- Installing a high availability RabbitMQ cluster

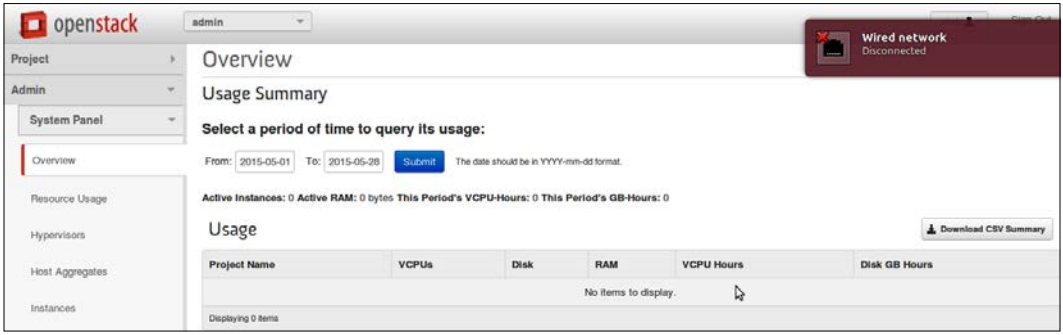
Installing MariaDB with Galera clustering

We are going to build a high availability database service. For this experiment, we will use MariaDB, which is basically a fork of MySQL and has a version control built for Galera that is available in an apt repository for Ubuntu 12.04. After this, we will use Galera clustering for bidirectional replication.

The following is the step-by-step procedure to install and configure MariaDB:

1. Create two OpenStack controller servers and assign to each of them an IP address (192.168.122.1 and 192.168.122.2)—install all the OpenStack services. After a successful installation of OpenStack on the server nodes, launch the horizon dashboard as follows.

2. The following image depicts what you see after you log in with the default user name and password as `admin` and OpenStack respectively.



3. After a successful login via the previous dashboard, the OpenStack cloud services will be offered and monitored through the following console. This includes all the activities of the cloud administrator for the provision and de-provision of services offered by OpenStack.

4. Then, add the MariaDB repo and install the MariaDB and Galera packages on the control nodes by carrying out the following instructions:

    1. Open a terminal in the controller node.
    2. Type the following command for the installation as shown in the following screenshot:

        sudo apt-get install python-software-properties

```
openstack@openstack-VirtualBox:~/devstack$ sudo apt-get install python-software-properties
Reading package lists... Done
Building dependency tree
Reading state information... Done
python-software-properties is already the newest version.
The following packages were automatically installed and are no longer required:
  gir1.2-ubuntuoneui-3.0 firefox-globalmenu libubuntuoneui-3.0-1 thunderbird-globalmenu
Use 'apt-get autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 32 not upgraded.
```

    3. Type the following command to get the following screen:

        sudo apt-key adv --recv-keys --keyserver hkp://keyserver.ubuntu.com:80 0xcbcb082a1bb943db

```
openstack@openstack-VirtualBox:~/devstack$ apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 0xcbcb082a1bb943db
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --secret-keyring /tmp/tmp.OItRorSdKZ --keyring /etc/apt/trusted.gpg --primary-ke
yring /etc/apt/trusted.gpg --recv-keys --keyserver keyserver.ubuntu.com 0xcbcb082a1bb943db
gpg: requesting key 1BB943DB from hkp server keyserver.ubuntu.com
gpg: no writable keyring found: eof
gpg: error reading '[stream]': general error
gpg: Total number processed: 0
openstack@openstack-VirtualBox:~/devstack$ sudo apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 0xcbcb082a1bb943db
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --secret-keyring /tmp/tmp.Gu9lFOA0Q3 --trustdb-name /etc/apt/trustdb.gpg --keyri
ng /etc/apt/trusted.gpg --primary-keyring /etc/apt/trusted.gpg --recv-keys --keyserver keyserver.ubuntu.com 0xcbcb082a1bb943db
gpg: requesting key 1BB943DB from hkp server keyserver.ubuntu.com
gpg: key 1BB943DB: public key "MariaDB Package Signing Key <package-signing-key@mariadb.org>" imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:               imported: 1
```

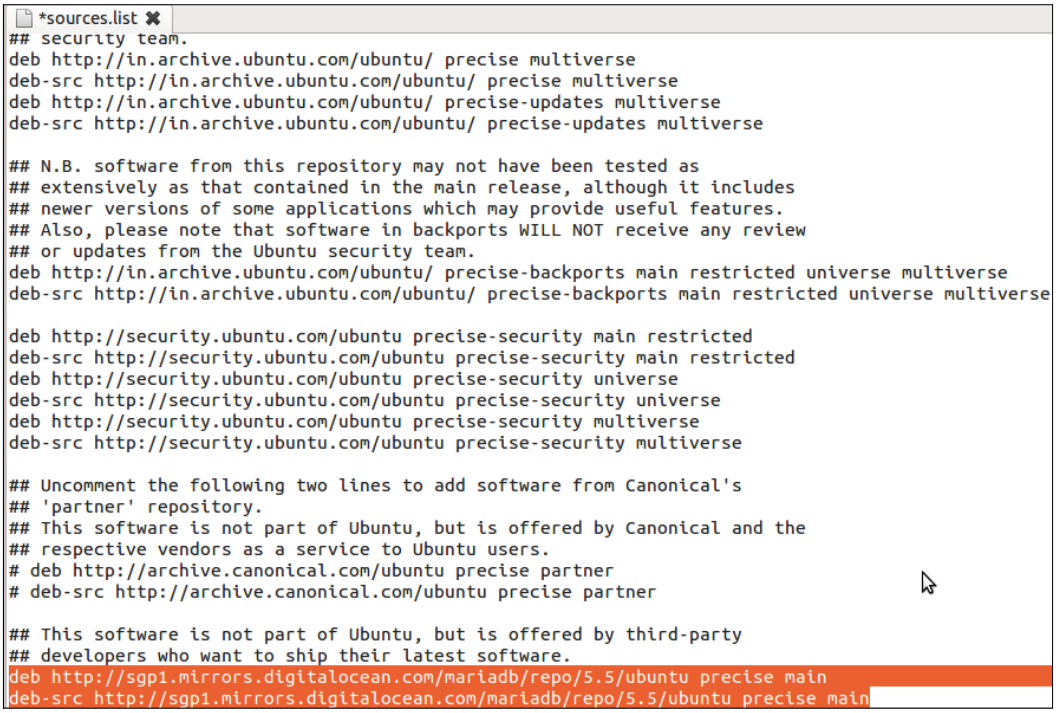
    4. The following `debian` packages are added to the Ubuntu debian package to update the system to support MariaDB. So, add the following lines at the bottom of your `/etc/apt/sources.list` file:

        deb http://sgp1.mirrors.digitalocean.com/mariadb/repo/5.5/ubuntu precise main

        deb-src http://sgp1.mirrors.digitalocean.com/mariadb/repo/5.5/ubuntu precise main

```
openstack@openstack-VirtualBox:~$ cd /etc/apt/
openstack@openstack-VirtualBox:/etc/apt$ ls
apt.conf.d      sources.list   sources.list.d     trustdb.gpg  trusted.gpg~
preferences.d   sources.list~  sources.list.save  trusted.gpg  trusted.gpg.d
openstack@openstack-VirtualBox:/etc/apt$ sudo gedit sources.list
```

After opening the preceding `sources.list` file with any one of the editors such as `vi` and `nano`, add the previous two lines of the new debians at the bottom of a file as shown in the following screenshot:



5. Now type the following command:

   ```
   sudo apt-get update
   ```

   The preceding update command works with Ubuntu's packages, an upgrade of the existing software packages, an update of the package list index, and even an upgrade of the entire Ubuntu system. So by running this command, the system will be updated.

Downloading the example code

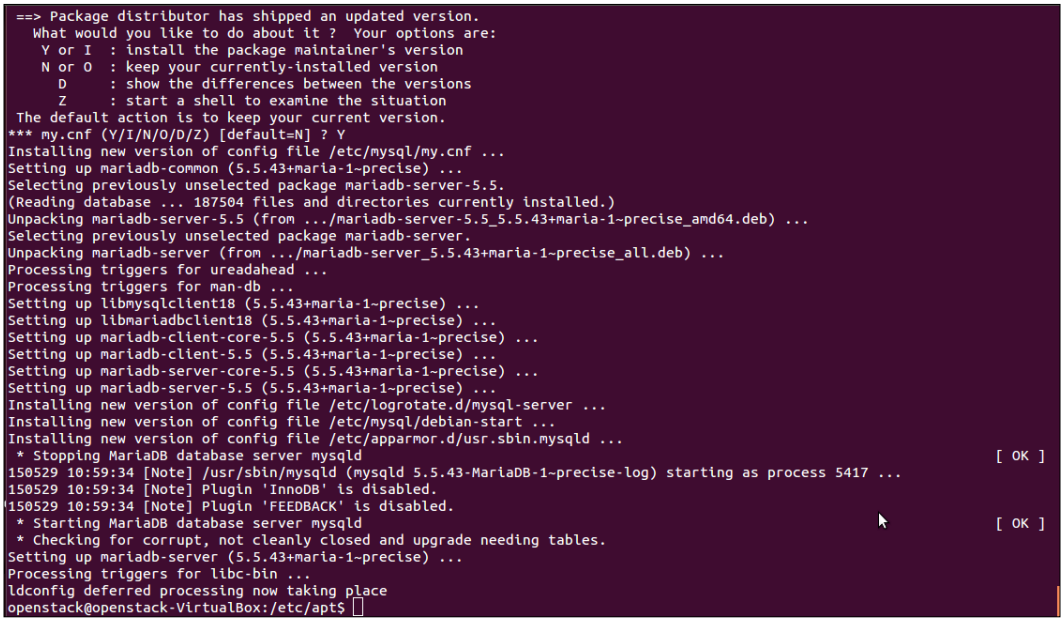
You can download the example code files from your account at `http://www.packtpub.com` for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

6. Next, we will install MariaDB by typing the following command:

```
sudo apt-get install mariadb-server
```

```
ureadahead will be reprofiled on next reboot
(Reading database ... 187360 files and directories currently installed.)
Preparing to replace mysql-common 5.5.37-0ubuntu0.12.04.1 (using .../mysql-commo
n_5.5.43+maria-1~precise_all.deb) ...
Unpacking replacement mysql-common ...
Selecting previously unselected package mariadb-common.
Unpacking mariadb-common (from .../mariadb-common_5.5.43+maria-1~precise_all.deb
) ...
Preparing to replace libmysqlclient18 5.5.37-0ubuntu0.12.04.1 (using .../libmysq
lclient18_5.5.43+maria-1~precise_amd64.deb) ...
Unpacking replacement libmysqlclient18 ...
Selecting previously unselected package libmariadbclient18.
Unpacking libmariadbclient18 (from .../libmariadbclient18_5.5.43+maria-1~precise
_amd64.deb) ...
Selecting previously unselected package mariadb-client-core-5.5.
Unpacking mariadb-client-core-5.5 (from .../mariadb-client-core-5.5_5.5.43+maria
-1~precise_amd64.deb) ...
Selecting previously unselected package mariadb-client-5.5.
Unpacking mariadb-client-5.5 (from .../mariadb-client-5.5_5.5.43+maria-1~precise
_amd64.deb) ...
Selecting previously unselected package mariadb-server-core-5.5.
Unpacking mariadb-server-core-5.5 (from .../mariadb-server-core-5.5_5.5.43+maria
-1~precise_amd64.deb) ...
Processing triggers for man-db ...
Setting up mysql-common (5.5.43+maria-1~precise) ...

Configuration file `/etc/mysql/my.cnf'
 ==> Modified (by you or by a script) since installation.
 ==> Package distributor has shipped an updated version.
   What would you like to do about it ?  Your options are:
    Y or I  : install the package maintainer's version
    N or O  : keep your currently-installed version
      D     : show the differences between the versions
      Z     : start a shell to examine the situation
 The default action is to keep your current version.
*** my.cnf (Y/I/N/O/D/Z) [default=N] ? Y
```

7. We will see the following information after a successful installation:



8. After a successful installation of MariaDB, the Galera configuration file needs to be changed according to our cluster setup as explained in the remaining steps.
9. Find a file called `cluster.conf` under `/etc/mysql/conf.d/cluster.cnf`.

10. The (`cluster.cnf`) must be updated with the attributes along with their values as shown in the following screenshot using the following command:

    `sudo nano /etc/mysql/conf.d/cluster.cnf`

```
[mysqld]
query_cache_size=0
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
query_cache_type=0
bind-address=0.0.0.0

# Galera Provider Configuration
wsrep_provider=/usr/lib/galera/libgalera_smm.so
#wsrep_provider_options="gcache.size=32G"

# Galera Cluster Configuration
wsrep_cluster_name="test_cluster"
wsrep_cluster_address="gcomm://192.168.122.1,192.168.122.2"

# Galera Synchronization Congifuration
wsrep_sst_method=rsync
#wsrep_sst_auth=user:pass

# Galera Node Configuration
wsrep_node_address="192.168.122.1"
wsrep_node_name="controllerNode1"
```

11. Use `wsrep_cluster_address` to match the address of our node called `controllerNode1` (192.168.122.1) as shown in the previous screenshot.

12. Finally, we need to edit the MySQL configuration to remove the bind-address statement as shown in following figure. The configuration file is called `my.conf` and resides under `/etc/mysql`. We need to add a comment with a #(hash symbol) to the bind-address attribute in the `my.conf` file; however, before that, open the file using the following command:

    ```
    sudo nano  /etc/mysql/my.conf
    ```

    This command gives us the following output:

```
lc_messages_dir = /usr/share/mysql
lc_messages     = en_US
skip-external-locking
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
#bind-address           = 127.0.0.1
#
# * Fine Tuning
#
max_connections         = 100
connect_timeout         = 5
wait_timeout            = 600
max_allowed_packet      = 16M
thread_cache_size       = 128
sort_buffer_size        = 4M
bulk_insert_buffer_size = 16M
tmp_table_size          = 32M
max_heap_table_size     = 32M
#
# * MyISAM
#
# This replaces the startup script and checks MyISAM tables if needed
# the first time they are touched. On error, make copy and try a repair.
myisam_recover          = BACKUP
key_buffer_size         = 128M
#open-files-limit       = 2000
table_open_cache        = 400
myisam_sort_buffer_size = 512M
concurrent_insert       = 2
read_buffer_size        = 2M
read_rnd_buffer_size    = 1M
```

13. Complete all the previously mentioned steps on both the nodes, and then we should stop the `mysql` service on both the nodes with the following command:

    ```
    service mysql stop
    ```

Installation of high availability RabbitMQ cluster

There are many forms of high availability, replication, and resilience in the face of various different types of failure. A RabbitMQ cluster can be made to work in an active and passive setup such that persistent messages that have been written to a disk on the active node can be recovered by the passive node, should the active node fail. Clustering enables a high availability of queues and increases the throughput. If a node fails, queues that were on the failed node are lost. With the high availability setup described in the following steps, a different node can recover the durable queues and the persistent messages within them when a node fails.

For this experiment, we will build two new Ubuntu 12.04 LTS servers with the following names and IP addresses:

- `Controller_1` (`192.168.56.101`)
- `Controller_2` (`192.168.56.102`)

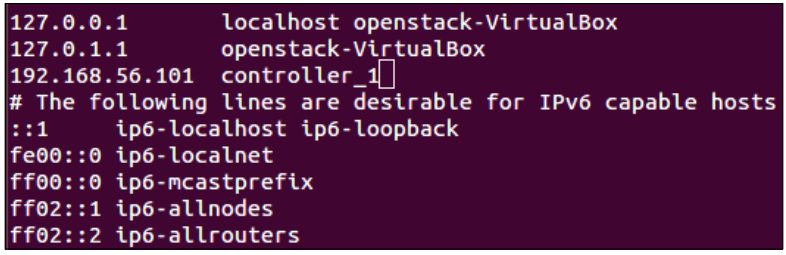
We will want the two nodes to be able to resolve each other by name. Then add each node to the other node's `/etc/hosts` file as shown in the following section.

Configuring the nodes to know each other

The following change is for the `Controller_1` (`/etc/hosts`):

1. Add the name and IP address of `controller_2` using the following command:

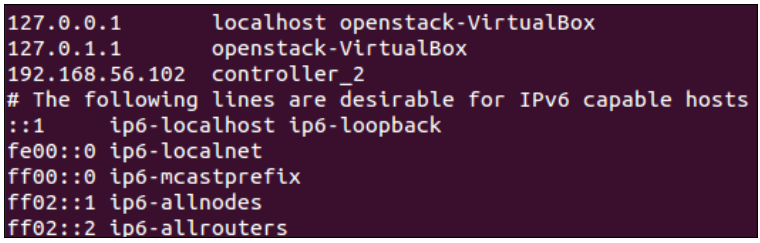
   `sudo nano /etc/hosts`



2. Then, save the preceding changes according to the editors (vi, nano, or some other editor) that you are using.
3. Next, make the changes to `Controller_2` (`/etc/hosts`).

4. Add the name and IP address of `controller_2` using the following command:

   ```
   sudo  nano /etc/hosts
   ```



5. Then, save the preceding changes. In the nano editor, use *Ctrl* + *x* to exit from the nano editor and type `yes` or `y` to save the file.

Installing RabbitMQ on the two nodes

The following procedure must be followed for both the nodes:

1. Update the system with the `sudo apt-get update` command.

   The update command works with Ubuntu's packages, an upgrade of the existing software packages, an update of the package list index, and even an upgrade of the entire Ubuntu system.

   Install the RabbitMQ message queue service on the node using the following command. The `ntp` package is used for the synchronization of services on multiple machines:

   ```
   sudo apt-get install ntp rabbitmq-server
   ```

2. After the installation has completed, we should stop the RabbitMQ services on both the nodes with the following command:

   ```
   sudo service rabbitmq-server stop
   ```

```
openstack@openstack-VirtualBox:~$ sudo service rabbitmq-server start
Starting rabbitmq-server: SUCCESS
rabbitmq-server.
openstack@openstack-VirtualBox:~$ sudo service rabbitmq-server stop
Stopping rabbitmq-server: rabbitmq-server.
```

Constructing a RabbitMQ broker

We are building a cluster of the RabbitMQ nodes to construct a RabbitMQ broker, a logical grouping of several Erlang nodes. Therefore, we need to copy the `erlang` cookie from `controller_1` to `controller_2`:

1. We may need to enable root `ssh` logon on `controller_2` for this step, otherwise we might need to copy the file to our `home` directory on `controller_2` and then move it to the correct location. To enable `root` logon on `controller_2`, simply type `passwd root` and enter a new password for the root when prompted.

```
openstack@openstack-VirtualBox:~$ passwd root
passwd: You may not view or modify password information for root.
openstack@openstack-VirtualBox:~$ sudo passwd root
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

2. Then going back to `controller_1`, we can copy the cookie to `controller_2`: with the following command:

   `scp /var/lib/rabbitmq/.erlang.cookie root@192.168.56.101:/var/lib/rabbitmq/.erlang.cookie`

```
openstack@openstack-VirtualBox:~$ scp /var/lib/rabbitmq/.erlang.cookie root@192.168.56.101:/var/lib/rabbitmq/.erlang.cookie
root@192.168.56.101's password:
```

Restarting the RabbitMQ services on the nodes

To restart the RabbitMQ services on both the nodes, type the `sudo` service rabbitmq-server `restart` command and run it on your terminal. These commands are input on the default settings by restarting the RabbitMQ services.

```
openstack@openstack-VirtualBox:~$ sudo service rabbitmq-server restart
Restarting rabbitmq-server: RabbitMQ is not running
SUCCESS
rabbitmq-server.
```

Formation of cluster

Run the following commands on `controller_2`:

```
rabbitmqctl stop_app
rabbitmqctl join_cluster rabbit@controller_1
rabbitmqctl start_app
```

```
openstack@openstack-VirtualBox:~$ sudo rabbitmqctl stop_app
Stopping node 'rabbit@openstack-VirtualBox' ...
...done.
openstack@openstack-VirtualBox:~$ sudo rabbitmqctl start_app
Starting node 'rabbit@openstack-VirtualBox' ...
...done.
```

Check the status of a cluster

Run the following command to know the status of the cluster from `controller_1`:

```
Sudo rabbitmqctl cluster status
```

```
openstack@openstack-VirtualBox:~$ sudo rabbitmqctl cluster_status
Cluster status of node 'rabbit@openstack-VirtualBox' ...
[{nodes,[{disc,['rabbit@openstack-VirtualBox']}]},
 {running_nodes,['rabbit@openstack-VirtualBox']}]
...done.
```

Summary

In this chapter, we had a practical example of the MariaDB installation with Galera clustering. We also learned about the detailed installation and setup procedure of a high availability RabbitMQ server.

In the next chapter, we will deep dive into network load balancing with detailed examples on configurations for HAproxy and keepalived and will learn about load balancing of HTTP REST API's, MySQL, and AMQP clusters.

3
Load Balancing for Active/Active Services

To avoid a single point of failure in OpenStack, we set up a two-node load balancer configuration with HAProxy, and keepalived. HAProxy provides a very consistent load balancing and high availability solution for applications, which are based on TCP and HTTP.

The two nodes monitor each other using keepalived. This is a piece of software that can be used to achieve high availability by assigning a virtual IP to two or more nodes and monitoring these nodes, and then failing over when one of the nodes goes down. The keepalived can do more than this, such as load balancing and monitoring. However, in this chapter, we'll focus on a very simple setup, that is IP failover.

In this chapter, we will cover the following main topic:

- The installation of HAProxy and keepalived

The installation of HAProxy and keepalived

Before installing HAProxy and keepalived, we should know the prerequisites for this experimental setup.

The requirement for an experimental setup

In this experiment, you will setup two node clusters with Ubuntu 12.04 LTS installed on both of these two nodes. These two nodes are created with the following names, `controller_1` and `controller_2`, and they are assigned the following IP addresses, `192.168.56.101` and `192.168.56.102` respectively.

Then a third IP address, `192.168.1.32`, is allocated to be used as a virtual IP address (VIP).

The following is the step-by-step procedure to install HAProxy and keepalived:

1. As an initial step of this installation process, we need to make the kernel aware that we intend to bind additional IP addresses that won't be defined in the interfaces file.
2. To do this, we edit `/etc/sysctl.conf` using any one of the editors mentioned in the previous chapter.

```
openstack@openstack-VirtualBox:~$ sudo gedit /etc/sysctl.conf
```

Then add the following line:

```
net.ipv4.ip_nonlocal_bind=1
```

```
*sysctl.conf
33 #net.ipv6.conf.all.forwarding=1
34
35
36 ###################################################################
37 # Additional settings - these settings can improve the network
38 # security of the host and prevent against some network attacks
39 # including spoofing attacks and man in the middle attacks through
40 # redirection. Some network environments, however, require that these
41 # settings are disabled so review and enable them as needed.
42 #
43 # Do not accept ICMP redirects (prevent MITM attacks)
44 #net.ipv4.conf.all.accept_redirects = 0
45 #net.ipv6.conf.all.accept_redirects = 0
46 # _or_
47 # Accept ICMP redirects only for gateways listed in our default
48 # gateway list (enabled by default)
49 # net.ipv4.conf.all.secure_redirects = 1
50 #
51 # Do not send ICMP redirects (we are not a router)
52 #net.ipv4.conf.all.send_redirects = 0
53 #
54 # Do not accept IP source route packets (we are not a router)
55 #net.ipv4.conf.all.accept_source_route = 0
56 #net.ipv6.conf.all.accept_source_route = 0
57 #
58 # Log Martian Packets
59 #net.ipv4.conf.all.log_martians = 1
60 #
61 net.ipv4.ip_nonlocal_bind=1
```

3. To reflect the change made in the file, we'll run the following command without rebooting the machine:

```
sudo sysctl -p
```

```
openstack@openstack-VirtualBox:~$ sudo sysctl -p
net.ipv4.ip_nonlocal_bind = 1
```

4. Next, you will install the HAProxy and keepalived software using the following command:

```
sudo apt-get update && apt-get install keepalived haproxy –y
```

```
openstack@openstack-VirtualBox:~$ sudo apt-get install keepalived haproxy -y
```

5. After a successful installation of the previous software, you will get the following screenshot:

```
openstack@openstack-VirtualBox:~$ sudo apt-get install keepalived haproxy -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
haproxy is already the newest version.
The following packages were automatically installed and are no longer required:
  gir1.2-ubuntuoneui-3.0 firefox-globalmenu libubuntuoneui-3.0-1 thunderbird-globalmenu
Use 'apt-get autoremove' to remove them.
Suggested packages:
  heartbeat ldirectord
The following NEW packages will be installed:
  ipvsadm keepalived
0 upgraded, 2 newly installed, 0 to remove and 307 not upgraded.
Need to get 170 kB of archives.
After this operation, 514 kB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu/ precise/main ipvsadm amd64 1:1.25.clean-1ubuntu5 [46.5 kB]
Get:2 http://in.archive.ubuntu.com/ubuntu/ precise-updates/main keepalived amd64 1:1.2.2-3ubuntu1.1 [124 kB]
Fetched 170 kB in 3s (48.2 kB/s)
Preconfiguring packages ...
Selecting previously unselected package ipvsadm.
(Reading database ... 187581 files and directories currently installed.)
Unpacking ipvsadm (from .../ipvsadm_1%3a1.25.clean-1ubuntu5_amd64.deb) ...
Selecting previously unselected package keepalived.
Unpacking keepalived (from .../keepalived_1%3a1.2.2-3ubuntu1.1_amd64.deb) ...
Processing triggers for ureadahead ...
ureadahead will be reprofiled on next reboot
Processing triggers for man-db ...
Setting up ipvsadm (1:1.25.clean-1ubuntu5) ...
update-rc.d: warning: ipvsadm start runlevel arguments (2 3 4 5) do not match LSB Default-Start values (2 3 5)
 * ipvsadm is not configured to run. Please run dpkg-reconfigure ipvsadm
Setting up keepalived (1:1.2.2-3ubuntu1.1) ...
```

Next, we define the `keepalived` configuration. We start by creating `/etc/keepalived/keepalived.conf` on both `controller_1` and `controller_2` as follows:

```
openstack@openstack-VirtualBox:~$ sudo gedit /etc/keepalived/keepalived.conf
```

The keepalived configuration on `controller_1` node. We have to configure each node in this experimental setup. So we need to do an initial start-up by configuring node 1, called `controller_1`.

You need to add the following items to the `keepalived.conf` file, and you have to set the `router_id` as the hostname, in this case this should be `controller_1`. Then you have to specify the VIP as `192.168.1.32`:

*keepalived.conf

```
global_defs {
  router_id controller_1
}
vrrp_script haproxy {
  script "killall -0 haproxy"
  interval 2
  weight 2
}
vrrp_instance 50 {
  virtual_router_id 50
  advert_int 1
  priority 101
  state MASTER
  interface eth4
  virtual_ipaddress {
    192.168.1.32 dev eth4
  }
  track_script {
    haproxy
  }
}
```

The keepalived configuration on controller_2

Now you need to add the following items to the `keepalived.conf` file and need to set up the `route_id`, which means assign the hostname as `router_id`. You have to set the `router_id` to be the hostname, in this case this should be `controller_2`. Then you have to specify the VIP as `192.168.1.32`:

*keepalived.conf

```
global_defs {
  router_id controller_2
}
vrrp_script haproxy {
  script "killall -0 haproxy"
  interval 2
  weight 2
}
vrrp_instance 50 {
  virtual_router_id 50
  advert_int 1
  priority 101
  state MASTER
  interface eth4
  virtual_ipaddress {
    192.168.1.32 dev eth4
  }
  track_script {
    haproxy
  }
}
```

Defining the HAProxy configuration

Next, you have to define the HAProxy configuration on `controller_1` and `controller_2` as follows:

Open the file called `haproxy.cfg` under the `/etc/haproxy/` directories using any editor with the following command:

```
sudo gedit /etc/haproxy/ haproxy.cfg
```

```
openstack@openstack-VirtualBox:~$ sudo gedit /etc/haproxy/haproxy.cfg
```

HAProxy configuration for the controller_1 node

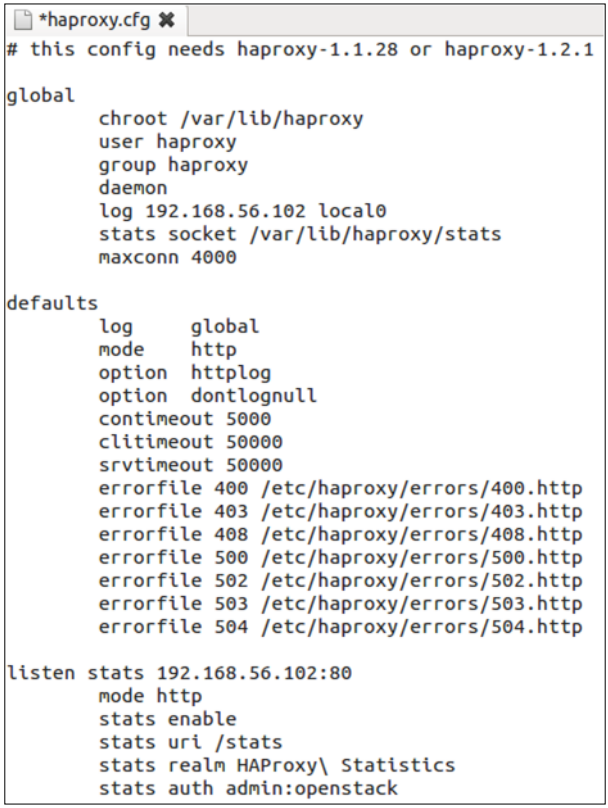
In the previously mentioned file, change the IP address of the log location item in the global section and the stats listener to `controller_1` IP address, `192.168.56.101`. Then set the username and password in the stats auth line. Set the password and username according to your machine. The username and password belongs to the OpenStack user:

*haproxy.cfg

```
# this config needs haproxy-1.1.28 or haproxy-1.2.1

global
        chroot /var/lib/haproxy
        user haproxy
        group haproxy
        daemon
        log 192.168.56.101 local0
        stats socket /var/lib/haproxy/stats
        maxconn 4000

defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        contimeout 5000
        clitimeout 50000
        srvtimeout 50000
        errorfile 400 /etc/haproxy/errors/400.http
        errorfile 403 /etc/haproxy/errors/403.http
        errorfile 408 /etc/haproxy/errors/408.http
        errorfile 500 /etc/haproxy/errors/500.http
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

listen stats 192.168.56.101:80
        mode http
        stats enable
        stats uri /stats
        stats realm HAProxy\ Statistics
        stats auth admin:openstack
```

The HAProxy configuration for the controller_2 node

In the preceding `haproxy.cfg` file, change the IP address of the log location item in the global section and the stats listener to `controller_2` IP address, `192.168.56.102`. Then set the username and password in the stats auth line. Set the password and username according to your machine. The global section of parameters is globally utilized for the configuration of nodes. The default sections with some parameters to set up the process-related attributes and the listening section provides a simple HAProxy configuration to listen on port 80 and forwards all the requests to the server that is listening at `192.168.56.102`:



*haproxy.cfg

```
# this config needs haproxy-1.1.28 or haproxy-1.2.1

global
        chroot /var/lib/haproxy
        user haproxy
        group haproxy
        daemon
        log 192.168.56.102 local0
        stats socket /var/lib/haproxy/stats
        maxconn 4000

defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        contimeout 5000
        clitimeout 50000
        srvtimeout 50000
        errorfile 400 /etc/haproxy/errors/400.http
        errorfile 403 /etc/haproxy/errors/403.http
        errorfile 408 /etc/haproxy/errors/408.http
        errorfile 500 /etc/haproxy/errors/500.http
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

listen stats 192.168.56.102:80
        mode http
        stats enable
        stats uri /stats
        stats realm HAProxy\ Statistics
        stats auth admin:openstack
```

Now we need to enable HAProxy. To do this, edit the `/etc/default/haproxy` file:

```
haproxy
# Set ENABLED to 1 if you want the init script to start haproxy.
ENABLED=0
# Add extra flags here.
#EXTRAOPTS="-de -m 16"
```

The default value of the **ENABLED** attribute value is changed to `1` as follows:

```
*haproxy
# Set ENABLED to 1 if you want the init script to start haproxy.
ENABLED=1
# Add extra flags here.
#EXTRAOPTS="-de -m 16"
```

Then restart the services of HAProxy and keepalived on both the nodes (`controller_1` and `controller_2`) using the following command:

```
sudo service keepalived restart
sudo service haproxy restart
```

```
openstack@openstack-VirtualBox:~$ service keepalived restart
 * Restarting keepalived keepalived...
```

After completing all of these steps on both the nodes, you should now have a highly available load balancer pair. At this point, our VIP should be active on one of the nodes. There are several algorithms available for load balancing in the HAProxy nodes such as round robin, leastcon, and source. In our case, we are using the round robin algorithm as the default algorithm for load balancing.

Therefore, we need to ensure the proper functioning of this whole setup. So, we need to make any one of the nodes, either `controller_1` or `controller_2`, active.

Making the controller_1 node active

The following procedure will make the `controller_1` node active at first. In this case, we have an assumption that the `controller_1` must have VIP in an active mode. To confirm, we can use the following `ip` command:

```
ip a | grep eth4 (in this case eth4 is assigned as main NIC)
```

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
inet 192.168.56.101/24 brd 192.168.1.255 scope global eth4
 inet 192.168.1.32/32 scope global eth4
openstack@openstack-VirtualBox:~$
```

For the confirmation of the activation of `controller_2` node, repeat the preceding step on the `controller_2` node.

Making the controller_2 node active

In this case, we have an assumption that the `controller_2` must have VIP in an active mode. To confirm, we can use the following `ip` command:

```
2: eth4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
inet 192.168.56.102/24 brd 192.168.1.255 scope global eth4
 inet 192.168.1.32/32 scope global eth4
openstack@openstack-VirtualBox:~$
```

Notice that both the local IP and the VIP are shown here. If we now reboot `node 1`, then `node 2` will quickly pick up the VIP.

Therefore, we should ensure that the Virtual IP is present on any one of the nodes, such as `controller_1` or `controller_2`, since VIP should be active on either `controller_1` or `controller_2`.

Summary

In this chapter, we learned the basic and advanced topics of network load balancing with detailed example configurations for HAProxy and keepalived.

In the next chapter, we will learn about classical clustering methods such as pacemaker cluster resources and their agents, start up order, failover and recovery, fencing mechanisms, and the load balancing of HTTP REST API's, MySQL, and the AMQP clusters.

4

Clustering, Fencing, and Active/Passive Services

Generally, OpenStack services offer stateless services and manage them to be as stateful services by providing redundant instances and load balancing them. However, it is very difficult to manage these services because of multiple actions that are involved in every request for them. In this chapter, we will make the stateful services highly available based on active/passive configuration.

An active/passive configuration means bringing additional resources online when other resources fail. Pacemaker or Corosync applications are used to bring back backup resources online whenever it is necessary. High availability will be achieved via a stack of components such as Pacemaker and Corosync.

In addition to Pacemaker cluster configuration, cluster resources and their agents, we will also cove the startup order, fail over and recovery and fencing mechanisms.

In this chapter, we will cover the following main topics:

- Installing Corosync and Pacemaker
- The load balancing of high availability MySQL
- High availability RabbitMQ via AMQP

Installing Corosync and Pacemaker

Before installing Corosync and Pacemaker, we should know the prerequisites for this experimental setup.

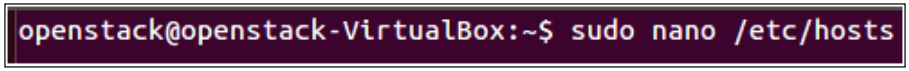
Requirements for the experimental setup

In this experiment, we will set up two node clusters with Ubuntu 12.04 LTS installed on both these nodes. These two nodes are created, and they are named as `controller_1` and `controller_2` and assigned with the `192.168.56.101` and `192.168.56.102` IP addresses, respectively. Then a third IP address, `192.168.1.32`, is allocated to be used as a Virtual IP address (VIP).

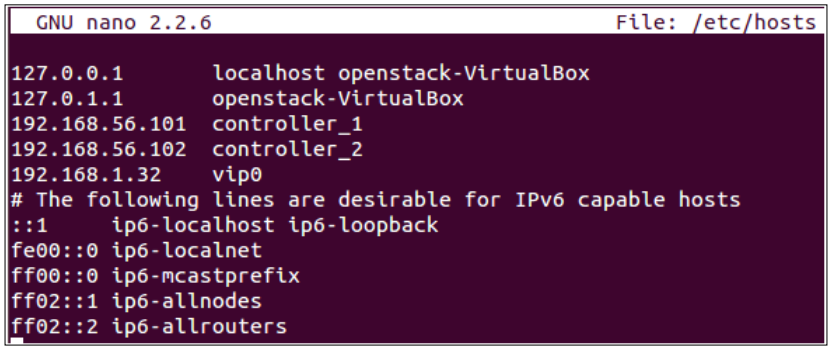
A secure Socket Host setup

We can have a **Secure Socket Host** (**SSH**) setup to access all other nodes through a key exchange so that the `host` file on the node will look like as follows:

```
|sudo nano /etc/hosts
```



After opening the `host` file, make the changes as shown in the following terminal screenshot:

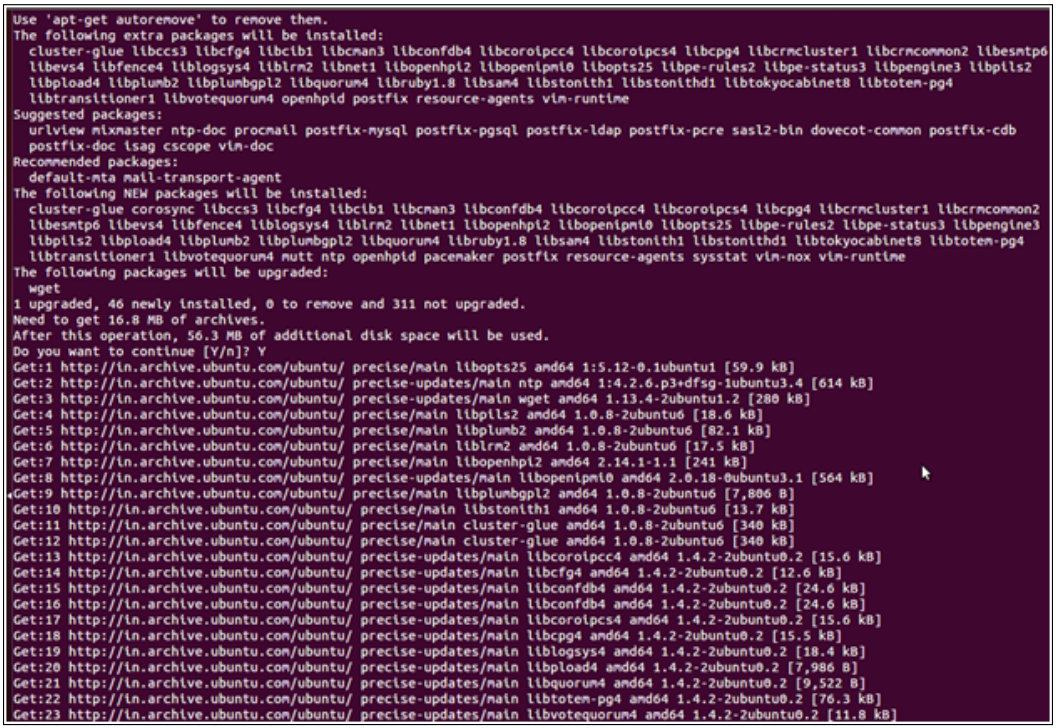


Installing the Corosync package

To make any host node become a part of the pacemaker cluster, we need to establish a cluster communication via Corosync that involves the installation of the following packages:

```
|sudo apt-get install pacemaker corosync crmsh -y
```

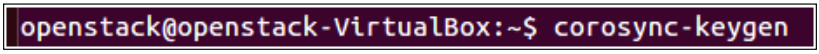
The following image shows that the Pacemaker installation was successful.



Sharing and generating Corosync keys

As an initial step of installation, the Corosync key must be generated and shared among all other nodes in a cluster. It is necessary to log in to each Corosync node and then secure cluster communication is done in an encrypted manner. Then this key is distributed among the cluster nodes.

```
| corosync-keygen
```



Now, share the keys with `node2` (`controller_2`):

```
rsync -a /etc/corosync/authkey controller_2:/etc/corosync/
```

Creating a configuration file

Now, we need to create a configuration for Corosync that is located in `/etc/corosync/corosync.conf`. To edit this file, use any of the editors in Ubuntu (`vi`, `nano`, or `gedit` and so on).

```
Sudo nano /etc/corosync/corosync.conf
```

```
openstack@openstack-VirtualBox:~$ sudo nano /etc/corosync/corosync.conf
```

The cluster name and the IP address must be changed according to our setup as follows:

```
totem {
        # cluster name
        cluster_name: mycluster

        # totem parameters
        version: 2
        secauth: off
        rrp_mode: none
        vsftype: none
        clear_node_high_bit: yes

        # low level network parameters
        token: 3000
        token_retransmits_before_loss_const: 10
        join: 60
        consensus: 4000
        max_messages: 20

        # not documented?
        threads: 0

        # ring0 interface
        interface {
                ringnumber: 0
                mcastport: 5405
                ttl: 1
        }

    # specify we don't use unicast or anycast
        transport: udpu
}

# quorum
quorum {
        provider: corosync_votequorum
        two_node: 1
}

# nodes
nodelist {
        node {
                ring0_addr: 192.168.56.101
        }
        node {
                ring0_addr: 192.168.56.102
        }
}
```

Starting Corosync

To ensure the `corosync` connectivity, we have a couple of tools `corosync-ctgtool` and `corosync-objctl`. The `corosync-ctgtool` is used to check the health of the cluster. Start the `corosync` service as a normal system service, as shown in the following:

```
| sudo /etc/init.d/corosync start
```

```
Printing ring status.
Local node ID 435324542
RING ID 0
   id = 192.168.56.101
status = ring 0 active with no faults
RING ID 1
   id = 192.168.56.102
status = ring 1 active with no faults
openstack@openstack-VirtualBox:~$
```

The `corosync-objctl` tool will list down the member list as follows:

```
corosync-objctl runtime.totem.pg.mrp.srp.members
```

```
openstack@openstack-VirtualBox:~$ corosync-objctl runtime.totem.pg.mrp.srp.members
```

Starting Pacemaker

After Corosync has started, communication must be established to check whether the cluster is communicating properly. The purpose of adding this Pacemaker with Corosync is to deal with failover of nodes in the cluster. Start the Pacemaker with the following commands:

```
Sudo nano /etc/init.d/pacemaker start
```

```
openstack@openstack-VirtualBox:~$ sudo /etc/init.d/pacemaker start
```

On a successful start of the Pacemaker services. It will create an empty cluster configuration by default. This cluster does not have any resources. We can check the status of this cluster using the `crm` utility on the terminal:

```
|crm_mon
```

```
============
Cluster name:myCluster
Last updated: Sun Jun 21 16:15:29 2015
Last change: Sun Jun 21 15:49:47 2015
Stack: corosync
Current DC: contoller_2 - partition with quorum
Version: 1.1.12-a9c8177
2 Nodes configured
0 Resources configured
============

Online: [ controller_2 controller_1 ]
openstack@openstack-VirtualBox:~$
```

Setting the cluster properties

The basic cluster properties need to be set for the pacemaker cluster with the help of the `crm` shell. The configuration file is changed with the `configure` command. The following are some cluster properties:

- `no-quorum-policy="ignore"`: This attribute value is set to `ignore` when we are using a two-node cluster (in our case). If we set this value, both the nodes will remain online and lose communication with one another. This value will be set up when we use three or more nodes in the cluster.
- `pe-warn-series-max ="1000"`, `pe-input-series-max ="1000"` and `pe-error-series-max="1000"`: Setting these values to 1000 sends a request to the Pacemaker to sustain a long history of inputs that are processed by this cluster.
- `cluster-recheck-interval="5min"`: Setting this value to process a cluster state needs an event-driven approach. It is used to make the Pacemaker actions occur at customizable intervals. We can change this value or interval according to the cluster requirement, as shown in the following screenshot:

```
| crm-configure
```

```
property no-quorum-policy=ignore
 pe-warn-series-max=1000
 pe-input-series-max=1000
 pe-error-series-max=1000
 cluster-recheck-interval=5min
openstack@openstack-VirtualBox:~$
```

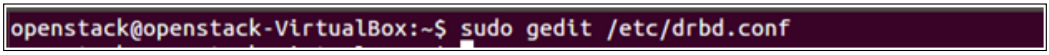
The load balancing of high availability MySQL

Many OpenStack services use MySQL as the default database server. Load balancing is necessary when any of the nodes is overloaded or fails due to some reason. To manage this failover situation, we need to exploit a solution called high availability as follows. To make this MySQL database server highly available, it requires configuring the distributed replicated block service as explained in the following sections.

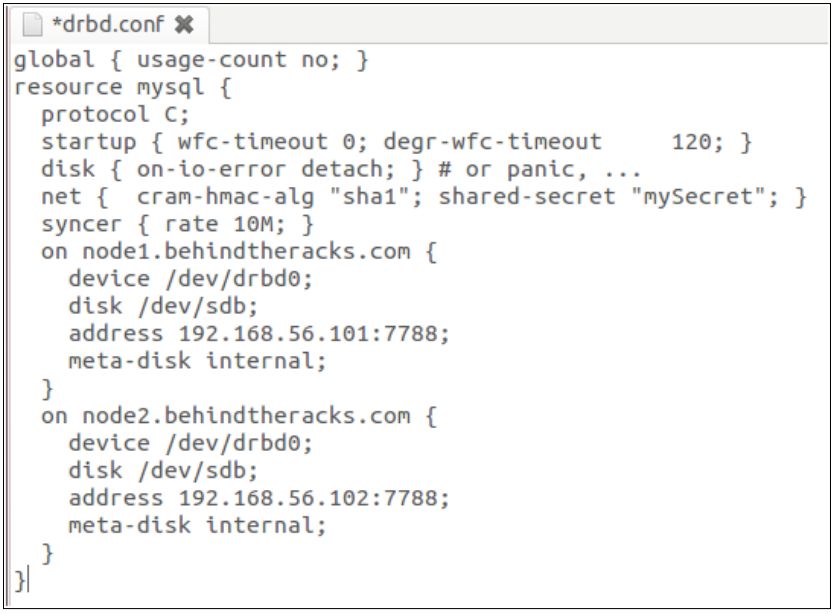
DRBD replicated storage

Replications of data between disks are done with DRDB; in our case, the `/dev/sdb` disk on `controller_1` and `controller_2`. We need to edit the configuration file of DRDB with the following command:

```
sudo gedit /etc/drbd.conf
```



The configuration file looks like the following:



The protocol C is used to create connections between devices, and it is used as a replication protocol. After this, we have to enter some of the DRBD commands for the initialization of the replica, as follows:

```
drbdam create-md mysql
```

```
openstack@openstack-VirtualBox:~$ drbdadm create-md mysql
```

On executing the preceding command, we will get the initial device creation:

```
v08 Magic number not found
Writing meta data...
initialising activity log
NOT initialized bitmap
New drbd meta data block sucessfully created.
success
openstack@openstack-VirtualBox:~$
```

Then, we need to start replicating any one of the nodes, either on `controller_1` or `controller_2` using the following command:

```
drbdam -- --force primary mysql
```

```
openstack@openstack-VirtualBox:~$ drbdadm -- --force primary mysql
```

Now the replication has started. Then, we need to check the status of the replication as follows:

```
cat /proc/drbd
```

```
openstack@openstack-VirtualBox:~$ cat /proc/drbd
```

Installing MySQL

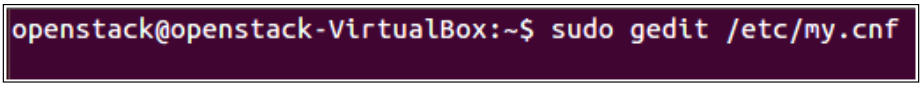
The installation should be done on both nodes (`controller_1` and `controller_2`) with the following command:

```
sudo apt-get install mysql-server
```

```
openstack@openstack-VirtualBox:~$ sudo apt-get install mysql-server
```

Then, we will add our Virtual IP (VIP) to `my.cnf` as follows:

```
sudo gedit /etc/my.cnf
```




```
bind-address = 192.168.1.32
```

The above changes have to be done in `mysql bind-address` to listen to the Pacemaker. So it can understand what is the IP address of MySQL to communicate with Pacemaker. Hence, the address of MySQL will be bound to the mentioned address.

Add the MySQL resources to Pacemaker.

After this, we will add the configuration of Pacemaker for the MySQL resources to the cluster. With the `crm` configuration, connect the Pacemaker cluster and add the following cluster resources:

```
primitive p_ip_mysql ocf:heartbeat:IPaddr2 \
  params ip="192.168.1.32" cidr_netmask="24" \
  op monitor interval="30s"
primitive p_drbd_mysql ocf:linbit:drbd \
  params drbd_resource="mysql" \
  op start timeout="90s" \
  op stop timeout="180s" \
  op promote timeout="180s" \
  op demote timeout="180s" \
  op monitor interval="30s" role="Slave" \
  op monitor interval="29s" role="Master"
primitive p_fs_mysql ocf:heartbeat:Filesystem \
  params device="/dev/drbd/by-res/mysql" \
    directory="/var/lib/mysql" \
    fstype="xfs" \
    options="relatime" \
  op start timeout="60s" \
  op stop timeout="180s" \
  op monitor interval="60s" timeout="60s"
primitive p_mysql ocf:heartbeat:mysql \
  params additional_parameters="--bind-address=192.168.42.101" \
    config="/etc/mysql/my.cnf" \
    pid="/var/run/mysqld/mysqld.pid" \
    socket="/var/run/mysqld/mysqld.sock" \
```

```
      log="/var/log/mysql/mysqld.log" \
    op monitor interval="20s" timeout="10s" \
    op start timeout="120s" \
    op stop timeout="120s"
group g_mysql p_ip_mysql p_fs_mysql p_mysql
ms ms_drbd_mysql p_drbd_mysql \
  meta notify="true" clone-max="2"
colocation c_mysql_on_drbd inf: g_mysql ms_drbd_mysql:Master
order o_drbd_before_mysql inf: ms_drbd_mysql:promote g_mysql:start
```

After adding the cluster details to `mysql`, we will get a status as follows:

```
primitive p_ip_mysql ocf:heartbeat:IPaddr2 params ip=192.168.56.101 cidr_netmask=24 op monitor interval=30s
primitive p_drbd_mysql ocf:linbit:drbd params drbd_resource=mysql op start timeout=90s op stop timeout=180s op promote timeout=180s op
 demote timeout=180s op monitor interval=30s role=Slave op monitor interval=29s role=Master
primitive p_mysql lsb:mysqld op monitor interval=20s timeout=10s op start timeout=120s op stop timeout=120s
primitive p_fs_mysql ocf:heartbeat:Filesystem params device=/dev/drbd0 directory=/var/lib/mysql fstype=xfs options=noatime op start ti
meout=60s op stop timeout=180s op monitor interval=60s timeout=60s
group g_mysql p_ip_mysql p_fs_mysql p_mysql
ms ms_drbd_mysql p_drbd_mysql meta notify=true clone-max=2
colocation c_mysql_on_drbd inf: g_mysql ms_drbd_mysql:Master
order o_drbd_before_mysql inf: ms_drbd_mysql:promote g_mysql:start
commit
exit
openstack@openstack-VirtualBox:~$
```

Once the configuration has been committed, the cluster will bring up the resources, and if all goes well, you should have MySQL running on one of the nodes, accessible via the VIP address. If any communication loss occurs on any of the active nodes in a cluster, that node will be removed or fenced from the cluster. With this fencing mechanism, the fenced node is completely isolated from the cluster.

To check the cluster status, type the following command:

```
| crm_mon -1
```

High availability RabbitMQ via AMQP

The AMQP server is used as a default server for many OpenStack services through RabbitMQ. Making the service highly available involves configuring a DRDB device.

Configuring DRDB

The configuration of DRDB involves the following:

- RabbitMQ can use the configured DRDB device
- Data Directory residing in this RabbitMQ device can be used by this configuration
- Virtual IP (VIP) selects and assigns floating IPs in a free manner among the cluster nodes

The RabbitMQ DRDB resource configuration is done with the following command:

```
sudo nano /etc/drbd.d/rabbit.res
```

```
openstack@openstack-VirtualBox:~$ sudo nano /etc/drdb.d/rabbit.res
```

The `rabbitmq` directory is mounted from the DRDB resource for the Pacemaker-based `rabbirmq` server. The `rabbitmq` resource is configured as follows:

```
  GNU nano 2.2.6                                File: /etc/drdb.d/rabbit.res

resource rabbitmq {
  device    minor 1;
  disk      "/dev/data/rabbitmq";
  meta-disk internal;
  on controller_1 {
    address ipv4 192.168.56.101:7701;
  }
  on controller_2 {
    address ipv4 192.168.56.102:7701;
  }
}
```

A backing device called `/dev/data/rabbitmq` is used by the previously mentioned resource on the cluster nodes, `controller_1` and `controller_2`. We will create an initial device with the following commands to write the initial set of metadata to the `rabitmq` device under the previously-specified directory:

```
drbdadm create-md rabbitmq
```

```
openstack@openstack-VirtualBox:~$ drbdadm create-md rabbitmq
```

Creating a filesystem

On successfully running a DRDB resource, a filesystem needs to be On successfully running a DRDB resource, the file system needs to create for the data that is available in the RabbitMQ. Consider this as a primary step in the filesystem creation process:

```
mkfs  -t xfs /dev/drbd1
```

```
openstack@openstack-VirtualBox:~$ mkfs -t xfs /dev/drbd1
```

Since the resource name is self-explanatory, we can use an alternative device path for an initial DRDB using the following:

```
mkfs -t xfs /dev/drbd/by-res/rabbitmq
```

```
openstack@openstack-VirtualBox:~$ mkfs -t xfs /dev/drbd/by-res/rabbitmq
```

For the device to return to the secondary running process on the cluster, use the following command:

```
drbdadm secondary  rabbitmq
```

```
openstack@openstack-VirtualBox:~$ drbdadm secondary rabbitmq
```

Preparing RabbitMQ for Pacemaker high availability

We need to check that the `erlang.cookie` files on both `controller_1` and `controller_2` are identical to ensure the Pacemaker monitoring functionality. So the `erlang.cookie` file is copied `controller_1` node to `controller_2` node, to the RabbitMQ data directory and to the DRDB file system as follows:

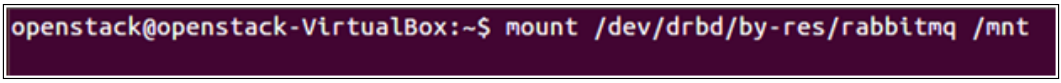
`erlang.cookie` file need to copy from one node to another:

```
scp -p /var/lib/rabbitmq/.erlang.cookie controller_2:/var/lib/rabbitmq/
```

```
openstack@openstack-VirtualBox:~$ scp -p /var/lib/rabbitmq/.erlang.cookie controller_2:/var/lib/rabbitmq/
```

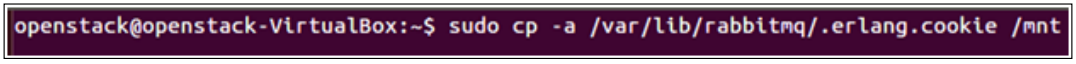
To mount a `rabbitmq` directory, use the following command:

```
mount /dev/drbd/by-res/rabbitmq /mnt
```



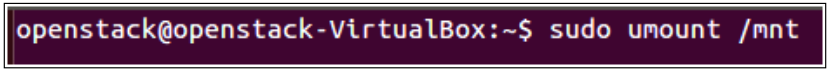
To copy `erlang.cookie` so that it can be mounted on a new device, use the following command:

```
sudo cp -a /var/lib/rabbitmq/.erlang.cookie /mnt
```



Finally, `unmount` an added directory as follows:

```
sudo unmount /mnt
```



Adding the RabbitMQ resources to Pacemaker

Now we add the Pacemaker configuration to the RabbitMQ resources. The `crm` tool is used to configure and add the following lines to the cluster resources as follows:

```
| crm configure
```

Type the previous command in the terminal followed by this code:

```
primitive p_ip_rabbitmq ocf:heartbeat:IPaddr2 \
  params ip="192.168.1.32" cidr_netmask="24" \
  op monitor interval="10s"
primitive p_drbd_rabbitmq ocf:linbit:drbd \
  params drbd_resource="rabbitmq" \
  op start timeout="90s" \
  op stop timeout="180s" \
  op promote timeout="180s" \
  op demote timeout="180s" \
  op monitor interval="30s" role="Slave" \
  op monitor interval="29s" role="Master"
```

```
primitive p_fs_rabbitmq ocf:heartbeat:Filesystem \
  params device="/dev/drbd/by-res/rabbitmq" \
    directory="/var/lib/rabbitmq" \
    fstype="xfs" options="relatime" \
  op start timeout="60s" \
  op stop timeout="180s" \
  op monitor interval="60s" timeout="60s"
primitive p_rabbitmq ocf:rabbitmq:rabbitmq-server \
  params nodename="rabbit@localhost" \
    mnesia_base="/var/lib/rabbitmq" \
  op monitor interval="20s" timeout="10s"
group g_rabbitmq p_ip_rabbitmq p_fs_rabbitmq p_rabbitmq
ms ms_drbd_rabbitmq p_drbd_rabbitmq \
  meta notify="true" master-max="1" clone-max="2"
colocation c_rabbitmq_on_drbd inf: g_rabbitmq ms_drbd_rabbitmq:Master
order o_drbd_before_rabbitmq inf: ms_drbd_rabbitmq:promote g_
rabbitmq:start
```

The preceding configuration file created the following important changes in the cluster:

- `p_ip_rabbitmq`: With this, RabbitMQ will use the Virtual IP address
- `p_fs_rabbitmq`: With this, a filesystem is mounted on the node where RabbitMQ is currently running
- `ms_drbd_rabbitmq`: With this, the `rabbitmq` DRDB service is managed by the master/slave set

Other constraints such as service group, order, and collocation are used to ensure that all the resources are started at each node properly and in the correct sequence.

```
| crm configure
```

After all the changes are made using `crm` configure, we commit the configuration by entering the `commit` command:

```
| commit
```

Configuring OpenStack services for highly available RabbitMQ

Now all the services provided by OpenStack point to the RabbitMQ configuration for high availability through a Virtual IP address instead of a physical IP address.

For example, the OpenStack image service points to the Virtual IP (change `rabbit_host` to our Virtual IP in the `glance-api.con` file). No other changes are required for the OpenStack configuration services. So if any node hosting this RabbitMQ experiences any problems of service failover due to some network problem, the service can continue to run without any interruption.

Summary

In this chapter, we learned that some services in OpenStack are still not fully stateless, thus they require classical clustering methods, such as Pacemaker. We have analyzed in depth the construction of a cluster of services, with their resource agents and dependencies.

We have also learned the step-by-step setting and configuration of load balancing of high availability MySQL and high availability RabbitMQ via AMQP.

In the next chapter, we will have a deeper understanding of how the OpenStack services work in cooperation with each other in a stateless mode to offer a scalable cloud framework.

5
Highly Available OpenStack Services

In this chapter, we will build highly available OpenStack services such as compute (Nova), image (Glance), object storage (Swift), and dashboard (Horizon) services. In the previous chapters, we successfully built high availability HAproxy load balancers, servers, and all other basic OpenStack services. From the knowledge gained from the preceding chapters, we will systematically implement the core services of OpenStack in a systematic manner. In this chapter, we will also learn the load balancing of HTTP REST APIS.

We are going to cover the following major topics in this chapter:

- High availability compute services
- High availability dashboard services
- High availability object storage services
- High availability image services
- The load balancing of HTTP REST API

High availability compute services

In the previous chapters, we utilized a two-node cluster with the nodes named as `controller_1` (`192.168.56.101`) and `controller_2` (`192.168.56.102`). For high availability compute services, we need to add Nova on both the nodes as explained in the following.

Nova is the main component of OpenStack, which hosts cloud computing services for the IaaS system. of OpenStack. Then OpenStack uses this system to manage the services that it offers. All other components interact with this service to provide complete cloud solutions to users.

Installing and configuring the Nova packages

As the initial step of installation, we need to install all the Nova-related packages, and we will do this with a single command:

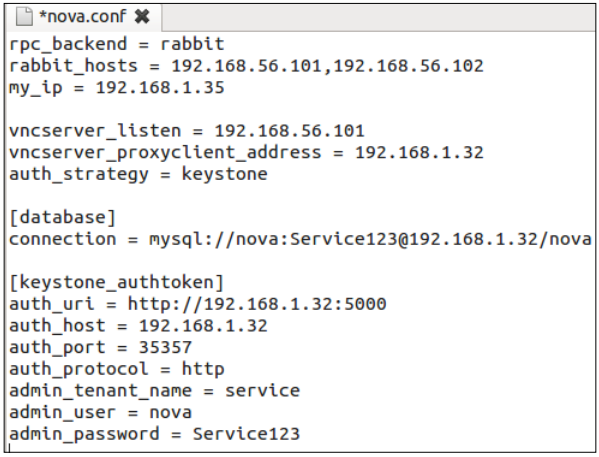
```
sudo apt-get install nova-api nova-cert nova-conductor nova-consoleauth
nova-novncproxy nova-scheduler python-novaclient
```

- `nova-api`: This API component is used to identify queues and send and receive HTTP requests from/to other services.
- `nova-cert`: This daemon is used to generate certificates for bundled images for EC2 API.
- `nova-conductor`: This aids in the functioning of all the services of OpenStack without accessing the database.
- `nova-consoleauth`: This is used to communicate with the backend database. All the responsibilities of users, projects, and groups are controlled by this component.
- `nova-novncproxy`: This component is used to access the compute services via VNC Client.
- `nova-scheduler`: This is used to allocate an appropriate host on each VM request based on the available algorithms.
- `python-novaclient`: This is the client for OpenStack Nova API.

Now for the configuration, modify the `nova.conf` file under `/etc/nova` using any of the editors such as `vi`, `nano`, or `edit`, with the following command:

```
sudo  gedit  /etc/nova/nova.conf
```

Then add the settings as specified in the following `nova.conf` file:



In the preceding settings, we assigned the `controller_1 ip address` to `my_ip` and `vncserver_listen`. We assigned the `controller_2 ip address` to `my_ip` and `vncserver_listen` of `controller_2`. The `rabbit_hosts` points to both `controller_1` and `controller_2`. The remaining IP address points to the load balancing Virtual IP, in our case, the VIP is pointing to `192.168.1.32`.

Creating the Nova database

1. Connect the local `mysql` database with the Nova database using the following command:

   ```
   mysql -h 192.168.1.32 -u root -p
   ```

2. Create a Nova database using the following command:

   ```
   create database nova;
   ```

```
ck@openstack-VirtualBox: ~
openstack@openstack-VirtualBox:~$ mysql -u root -p -h 127.0.0.1
Enter password:
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)
openstack@openstack-VirtualBox:~$
openstack@openstack-VirtualBox:~$ sudo mysql -u root -p -h 127.0.0.1
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 45
Server version: 5.5.37-0ubuntu0.12.04.1-log (Ubuntu)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database nova;
```

3. Grant all privileges with the following command to the database that we have created:

   ```
   grant all on nova.* to nova@'%' identified by 'Service123';
   ```

4. Exit from MySQL using the `exit` command.

Populating a database

The new databases of any of the nodes are populated (in our case, we populate the database available on `controller_1`) with the following command:

```
sudo nova-manage db sync
```

Then we restart all the Nova services as follows:

```
service nova-api restart
service nova-cert restart
service nova-consoleauth restart
service nova-scheduler restart
service nova-conductor restart
service nova-novncproxy restart
```

Instead of using the preceding commands, we can use this command:

```
Service nova-api nova-cert nova-consoleauth nova-scheduler nova-conductor
nova-novncproxy restart
```

The load balancing of compute services

For compute services to be load balanced, we need to edit `haproxy.cfg` by adding the following lines of code both to `controller_1` and `controller_2`.

To edit the `haproxy.cfg` file under `/etc/haproxy/`, we use the following command:

```
sudo gedit /etc/haproxy/haproxy.cfg
```

Then, we'll make changes according to the following `haproxy.cfg` file:

This `config` file comprises a simple configuration of HA proxy, where the proxy has a frontend and backend and each end has an IP address and a port. The frontend has an IP address and a port to listen to the HAProxy. The backend server defines a group of servers (`node1` and `node2` in our case) and IP addresses for load balancing purposes using specific algorithms.

*haproxy.cfg

```
listen nova_ec2 192.168.1.32:8773
        balance source
        option tcpka
        option httpchk
        maxconn 10000
        server node1 192.168.56.101:8773 check inter 2000 rise 2 fall 5
        server node2 192.168.56.102:8773 check inter 2000 rise 2 fall 5

listen nova_osapi 192.168.1.32:8774
        balance source
        option tcpka
        option httpchk
        maxconn 10000
        server node1 192.168.56.101:8774 check inter 2000 rise 2 fall 5
        server node2 192.168.56.102:8774 check inter 2000 rise 2 fall 5

listen nova_metadata 192.168.1.32:8775
        balance source
        option tcpka
        option httpchk
        maxconn 10000
        server node1 192.168.56.101:8775 check inter 2000 rise 2 fall 5
        server node2 192.168.56.102:8775 check inter 2000 rise 2 fall 5

listen novnc 192.168.1.32:6080
        balance source
        option tcpka
        maxconn 10000
        server node1 192.168.56.101:6080 check inter 2000 rise 2 fall 5
        server node2 192.168.56.102:6080 check inter 2000 rise 2 fall 5
```

Reloading the HAProxy services

Finally, we need to reconfigure the HAProxy services using the following command:

```
sudo service haproxy reload
```

Then, check for high available Nova services by sourcing our credentials using the following command. This sourcing command helps to check whether the high availability of the Nova service has been enforced in the given images list:

```
sudo source credentials
nova image-list
```

```
openstack@openstack-VirtualBox:~/devstack$ nova image-list
+--------------------------------------+---------------------------------+--------+--------+
| ID                                   | Name                            | Status | Server |
+--------------------------------------+---------------------------------+--------+--------+
| 10d1fc37-2e93-44b6-b5b7-195f3b3abd78 | cirros-0.3.1-x86_64-uec         | ACTIVE |        |
| 7c34ea17-b88e-4435-badc-7fc93b67f9f5 | cirros-0.3.1-x86_64-uec-kernel  | ACTIVE |        |
| 85e042f8-92b2-4fd7-98ba-66e3d8926b26 | cirros-0.3.1-x86_64-uec-ramdisk | ACTIVE |        |
+--------------------------------------+---------------------------------+--------+--------+
```

High availability dashboard services

The highly available dashboard service called Horizon is used to implement a web-based dashboard interface for all the OpenStack services including Swift, Cinder, Keystone, Nova and so on.

Installing and configuring the dashboard

As the initial step of dashboard installation, we need to install the complete package of dashboard services using the following command:

```
sudo apt-get install apache2 memcached libapache2-mod-wsgi openstack-dashboard
```

Then, we need to edit the `local_settings.py` file under `/etc/openstack-dashboard/` using any of the editors such as `gedit`, `vi`, and `nano`, as follows:

```
sudo gedit  /etc/openstack-dashboard/local_settings.py
```

Change the `OPENSTACK_HOST` IP address to our own **Virtual IP** (**VIP**) (`192.168.1.32`) as follows:

*local_settings.py

```
CACHES = {
   'default': {
       'BACKEND' : 'django.core.cache.backends.memcached.MemcachedCache',
       'LOCATION' : '192.168.1.32:11211',
   }
}
...
OPENSTACK_HOST = "192.168.1.32"
```

Configuring Memcache

We need to edit the `memcached.conf` file under `/etc/` using any of the editors, as follows:

```
sudo gedit /etc/memcached.conf
```

After this, we need to change the listening address from `127.0.0.1` to the `192.168.56.101` address of `controller_1` and `192.168.56.102` of `controller_2`:

*memcached.conf

```
#listenting address
-l 192.168.1.35
```

Restarting the Memcache services

Since we made some changes in the `memcached.conf` file, we need to restart the services using the following commands:

```
service apache2 restart
service memcached restart
```

Load balancing of dashboard services

The `haproxy` load balancers need some modification in the `haproxy.conf` file under `/etc/haproxy/haproxy.cfg` as follows:

```
sudo gedit /etc/haproxy/haproxy.cfg
```

The `haproxy` file needs to change as shown here:

*haproxy.cfg

```
listen dashboard 192.168.1.32:80
        balance  source
        capture  cookie vgnvisitor= len 32
        cookie  SERVERID insert indirect nocache
        mode  http
        option  forwardfor
        option  httpchk
        option  httpclose
        rspidel  ^Set-cookie:\ IP=
        server controller_1 192.168.56.101:80 cookie control01 check inter 2000 rise 2 fall 5
        server controller_2 192.168.56.102:80 cookie control02 check inter 2000 rise 2 fall 5

listen memcached 192.168.1.32:11211
        balance source
        option tcpka
        option httpchk
        maxconn 10000
        server controller_1 192.168.56.101:11211 check inter 2000 rise 2 fall 5
        server controller_2 192.168.56.102:11211 check inter 2000 rise 2 fall 5
```

The preceding file contains the dashboard configuration to access all the services on the web interface and Memcache to listen to the controller using `192.168.1.32`.

Reloading the HAProxy services

We need to reconfigure the HAProxy services using the following command:

```
service haproxy reload
```

```
openstack@openstack-VirtualBox:~$ sudo service haproxy reload
```

Then, we can open our dashboard (Horizon) using the `http://192.168.1.32/horizon` URL and log in with `admin` as a username and `password` as password.

High availability object storage services

The high availability object storage services offer a facility to store and retrieve data with a simple API. We can scale and optimize these services for the concurrency, availability, and durability of the entire dataset. Then, this Swift service acts as a backing storage for high availability glance services.

Installing and configuring object storage

First, we will install all the required packages for the object storage using the following command:

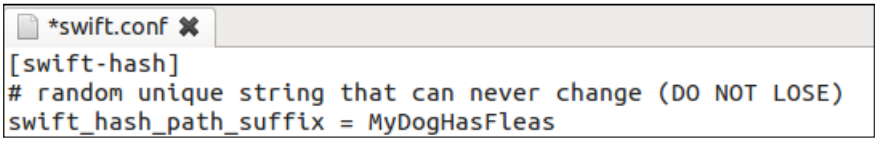
```
sudo apt-get install swift swift-account swift-container swift-object
xfsprogs
```

After the installation of the packages, we must create a `swift` configuration file for the object storage, in which we have to define a hash for all the Swift servers uniquely. The `swift.conf` file must be edited and the same must be copied to all the swift nodes uniquely.

To configure the `swift` configuration file, we must use the following command:

```
sudo gedit /etc/swift/swift.conf
```

Then, make the following changes in the preceding file:



Creating a disk partition

We have to create a disk partition for a Swift storage, then only we can use this object storage. However, we must format storage node as an XFS filesystem, as shown in the following:

```
fdisk /dev/sdb
n

(enter)
(enter)
(enter)
(enter)
w
mkfs.xfs /dev/sdb1
```

Next, create a directory to mount disk partition and provide ownership for a Swift user:

```
echo "/dev/sdb1 /srv/node/sdb1 xfs
noatime,nodiratime,nobarrier,logbufs=8 0 0" >> /etc/fstab
```

Create a new directory with the name `sdb1`, as follows:

```
sudo mkdir -p /srv/node/sdb1
```

Now, provide read ownership for the node as follows:

```
sudo chown -R swift:swift /srv/node
```

Creating directories

We need to create some directories for Swift as follows:

```
mkdir -p /var/swift/recon
chown -R swift:swift /var/swift/recon
```

Create a new directory with the name Swift and provide change ownership to read the file as follows:

```
mkdir /home/swift
chown -R swift:swift /home/swift
```

Replicating data on storage nodes

All Swift must do a `resync` to replicate data among the storage nodes using the following command:

```
sudo gedit /etc/rsync.conf
```

Now, add the following changes in the preceding file:

*rsyncd.conf

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 192.168.1.37

[account]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/account.lock

[container]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/container.lock

[object]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/object.lock
```

We must change the IP address to `192.168.1.37` for `swift node_1`, `192.168.1.38` for `swift node_2`, and `192.168.1.40` for `swift node_3`.

Then, enable `RSYNC_ENABLE` to be true. Make the value true by modifying the `rsync` file under `/etc/`, as follows:

`sudo gedit /etc/default/rsync`

*rsync

```
# start rsync in daemon mode from init.d script?
#  only allowed values are "true", "false", and "inetd"
#  Use "inetd" if you want to start the rsyncd from inetd,
#  all this does is prevent the init.d script from printing a message
#  about not starting rsyncd (you still need to modify inetd's config yourself).
RSYNC_ENABLE=true

# which file should be used as the configuration file for rsync.
# This file is used instead of the default /etc/rsyncd.conf
# Warning: This option has no effect if the daemon is accessed
#          using a remote shell. When using a different file for
#          rsync you might want to symlink /etc/rsyncd.conf to
#          that file.
# RSYNC_CONFIG_FILE=
```

Now, we restart the `rsync` service by using the following command:

```
service rsync start
```

Follow the preceding steps on both the Swift nodes to complete the process.

Installing a Swift proxy

Once again, we install the following components on all the Swift nodes, `swift node_1`, `swift node_2` and `swift node_3`:

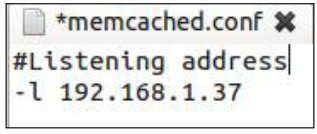
```
sudo apt-get install swift-proxy memcached python-keystoneclient python-swiftclient python-webob
```

Configuring Memcache

We configure the `memcached` file to listen with the local IP address (in this case, the IP is `192.168.1.37`):

```
sudo gedit /etc/memcached.conf
```

Change the listening address to VIP, as follows:



Then, restart `memcached` with the following command:

```
service memcached restart
```

Creating a proxy configuration file

The configuration file will be created with the following command:

```
sudo gedit /etc/swift/proxy-server.conf
```

Now, make the following changes in the preceding file:

*proxy-server.conf

```
[DEFAULT]
bind_port = 8080
user = swift

[pipeline:main]
pipeline = healthcheck cache authtoken keystoneauth proxy-server

[app:proxy-server]
use = egg:swift#proxy
allow_account_management = true
account_autocreate = true

[filter:keystoneauth]
use = egg:swift#keystoneauth
operator_roles = Member,admin,swiftoperator

[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory

# Delaying the auth decision is required to support token-less
# usage for anonymous referrers ('.r:*').
delay_auth_decision = true

# cache directory for signing certificate
signing_dir = /home/swift/keystone-signing

# auth_* settings refer to the Keystone server
auth_protocol = http
auth_host = 192.168.1.32
auth_port = 35357

# the service tenant and swift username and password created in Keystone
admin_tenant_name = service
admin_user = swift
admin_password = Service123

[filter:cache]
use = egg:swift#memcache
memcache_servers = 192.168.1.37:11211,192.168.1.38:11211,192.168.1.42:11211
```

Configuring a Swift ring

In this step, we will create a Swift ring configuration and then add our storage locations. We need to copy the file from `/etc/swift` to the other two Swift nodes (`swift node_2` and `swift node_3`) with the following command:

```
scpscp -r *.ring.gz root@192.168.1.38:/etc/swift
scpscp -r *.ring.gz root@192.168.1.42:/etc/swift
```

Then, we will restart all the `swift` services using this command:

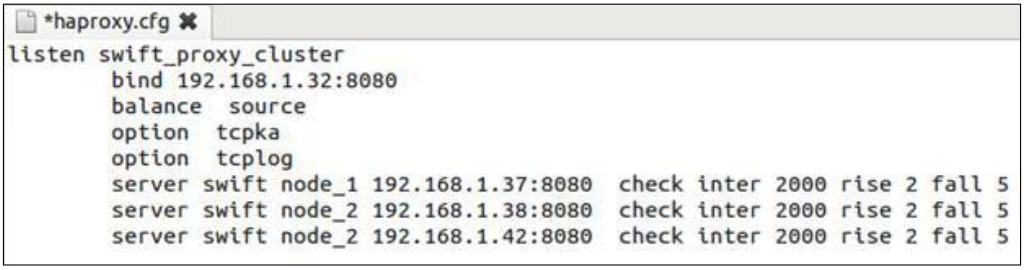
```
swiftswift-init restart all
```

The load balancing object store services

Separately add all the swift nodes for the balancer configuration using `haproxy.cfg`. To do this, we need to edit the `haproxy.cfg` file under `/etc/haproxy/` using the following command:

```
sudo gedit /etc/haproxy/haproxy.cfg
```

The `haproxy.cfg` file needs to be edited as follows:



These changes have to take effect only after the reload of the `haproxy` configuration that is done using the following command:

```
sudo service haproxy reload
```

Then, check for the high availability swift service with this:

```
sudo swift stat
```

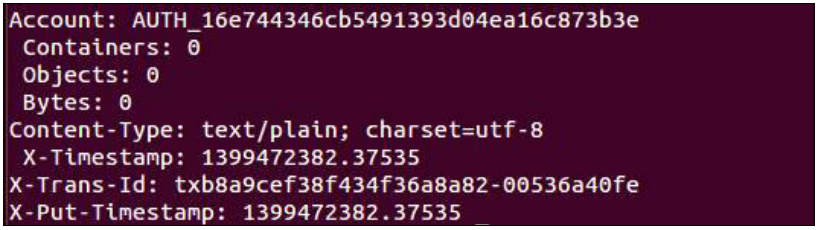
However, we must source some enviromental variables as follows:

```
export OS_USERNAME=admin
export OS_PASSWORD=password
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://192.168.1.32:35357/v2.0
```

On successfully obtaining the variables, run the following command:

```
sudo swift stat,
```

We will get the following output:



High availability image services

In high availability glance, a service acts as a backend storage for the object storage called Swift. Image service is used to store the images in a shared manner.

Installing and configuring image services

As the initial step of activating image services, we have to install the Python glance client using this:

```
sudo apt-get install glance python-glanceclient
```

Now, we need to modify several image service files for this implementation. The first file that we must modify is `glance-api.conf` that is available under `/etc/glance/`. The modification is done using following command:

```
sudo gedit /etc/glance/glance-api.conf
```

We changed our Virtual IP for Swift, database, and authentication:

*glance-api.conf

```
default_store = swift
registry_host = 192.168.1.32
auth_strategy = keystone

#rabbit_host = localhost
rabbit_hosts = 192.168.56.101,192.168.56.102

swift_store_auth_address = http://192.168.1.32:5000/v2.0/
swift_store_user = service:swift
swift_store_key = Service123
swift_store_container = glance
swift_store_create_container_on_put = True
...
[database]
#sqlite_db = /var/lib/glance/glance.sqlite
connection=mysql://glance:Service123@192.168.1.32/glance

[keystone_authtoken]
auth_host = 192.168.1.32
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = glance
admin_password = Service123
```

Then, we need to modify the next file called `glance-registry.conf` available under `/etc/glance`. This is done as follows:

```
sudo gedit /etc/glance/glance-registry.conf
```

The following changes are to be done in the preceding glance-registry.conf file:

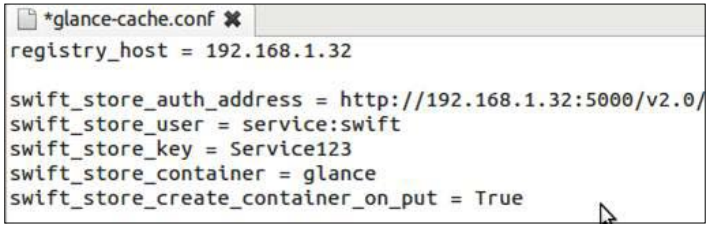
*glance-registry.conf

```
[DEFAULT]
[database]
# The file name to use with SQLite (string value)
#sqlite_db = /var/lib/glance/glance.sqlite
connection=mysql://glance:Service123@192.168.1.32/glance

[keystone_authtoken]
auth_host = 192.168.1.32
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = glance
admin_password = Service123
```

The next file to be edited is the `glance-cache.conf` file, available under `/etc/glance`:

```
sudo gedit /etc/glance/glance-cache.conf
```

The following changes have to be done in the preceding file:

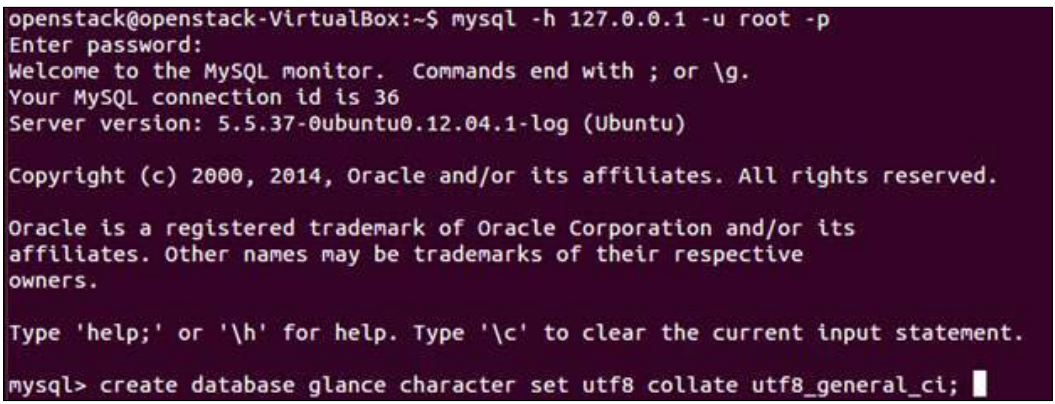


Creating the Glance database

Create a new database and grant all privileges with the following command on the mysql terminal:

```
mysql -h 192.168.1.32 -u root -p
create database glance character set utf8 collate utf8_general_ci;
```

Here is the output after running the first command:



The Glance database offers all privileges so that it can be accessed from all other OpenStack services:

```
grant all on glance.* to glance@'%' identified by 'Service123';
flush privileges;
```

Finally, quit with the `exit` command.

Populating the databases

Populate the new tables on any node with some data using the following command:

```
sudo glance-manage db_sync
```

Restart the `glance` service with the following commands:

```
sudo service glance-api restart
sudo service glance-registry restart
```

The load balancing of image services

To load the balance image service, we need to edit the `haproxy.cfg` file under `/etc/haproxy/` using the following command:

```
sudo gedit /etc/haproxy/haproxy.cfg
```

Then, we need to edit the file with the following changes:

*haproxy.cfg

```
listen glance-api 192.168.1.32:9292
        balance source
        option tcpka
        option httpchk
        maxconn 10000
        server controller_1 192.168.56.101:9292 check inter 2000 rise 2 fall 5
        server controller_2 192.168.56.102:9292 check inter 2000 rise 2 fall 5

listen glance-registry 192.168.1.32:9191
        balance source
        option tcpka
        option httpchk
        maxconn 10000
        server controller_1 192.168.56.101:9191 check inter 2000 rise 2 fall 5
        server controller_2 192.168.56.102:9191 check inter 2000 rise 2 fall 5
```

For the preceding changes to take effect, we need to reload the `haproxy` file with the following command:

```
sudo service haproxy reload
```

Next, we load balance the Glance service backend with Swift storage. For testing purpose, we can add a new image to the Glance service using the following commands:

```
wget http://download.cirros-cloud.net/0.3.1/cirros-0.3.1-x86_64-
disk.img
glance image-create --name cirros --is-public=true --disk-
format=qcow2 --container-format=ovf < cirros-0.3.1-x86_64-disk.img
```

You can view all glance images with this command:

```
sudo glance image-list
```

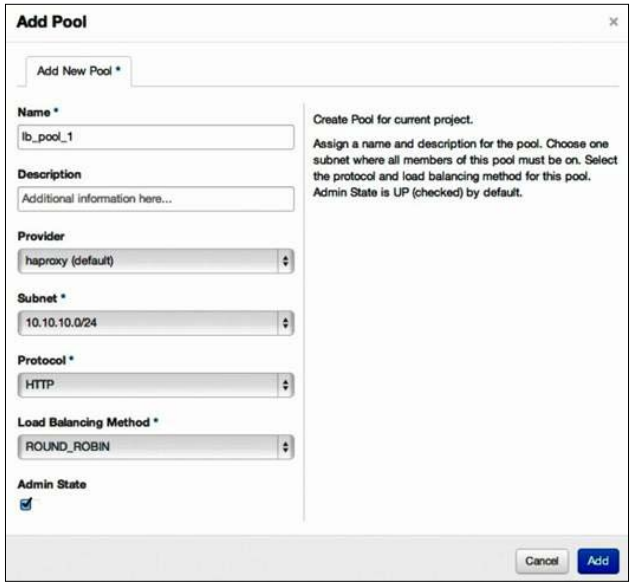
```
+--------------------------------------+---------------------------------+-------------+------------------+----------+--------+
| ID                                   | Name                            | Disk Format | Container Format | Size     | Status |
+--------------------------------------+---------------------------------+-------------+------------------+----------+--------+
| 10d1fc37-2e93-44b6-b5b7-195f3b3abd78 | cirros-0.3.1-x86_64-uec         | ami         | ami              | 25165824 | active |
| 7c34ea17-b88e-4435-badc-7fc93b67f9f5 | cirros-0.3.1-x86_64-uec-kernel  | aki         | aki              | 4955792  | active |
| 85e042f8-92b2-4fd7-98ba-66e3d8926b26 | cirros-0.3.1-x86_64-uec-ramdisk | ari         | ari              | 3714968  | active |
+--------------------------------------+---------------------------------+-------------+------------------+----------+--------+
```

The load balancing HTTP REST API

Here we consider a Python-based module to act as a web server that is running on two different instances (`web_1` and `web_2`) on the OpenStack infrastructure. The basic `index.html` file running on these nodes acts as sample HTTP application.

Creating a load balancing pool

An IP pool is created, as follows, with the name `lb_pool_1`:

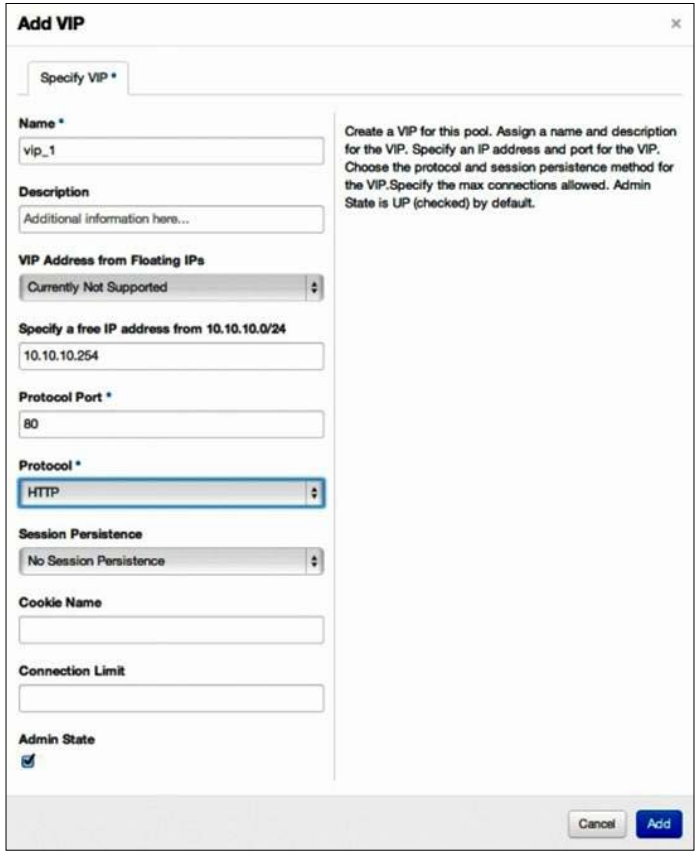


The Add Pool table dropdown menu is selected with the following data:

- Provider = haproxy: This is the default option
- Subnet = 10.10.10.0/24: This attribute is a subnet used to attach the instances to the subnet and for load balance
- Protocol = HTTP: In this, we can have other protocol options also
- Load Balancing Method = ROUND_ROBIN: In this, we can have other load balancing method options also

Adding a Virtual IP (VIP)

All the running servers are attached with this **Virtual IP** (**VIP**). The VIP setting is done as follows:



All fields are filled with the following data:

- Name = vip_1: This is the name of the VIP
- Specify a free IP = 10.10.10.254: This is used as the VIP
- Protocol Port = 80: The VIP listens via this port, and this must match with the incoming request
- Protocol = HTTP: In this, we can have other protocol options also

Launching instances

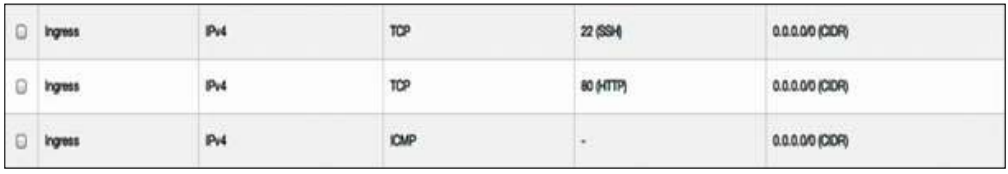
There are two instances running as follows. The two instances (`web_1` and `web_2`) run via the Horizon services of OpenStack:

Instance Name	Image Name	IP Address	Size	Keypair	Status
web_2	Fedora20	10.10.10.4	demo | 512MB RAM | 1 VCPU | 5.0GB Disk	aio-key	Active
web_1	Fedora20	10.10.10.2	demo | 512MB RAM | 1 VCPU | 5.0GB Disk	aio-key	Active

Displaying 2 items

Security group creation

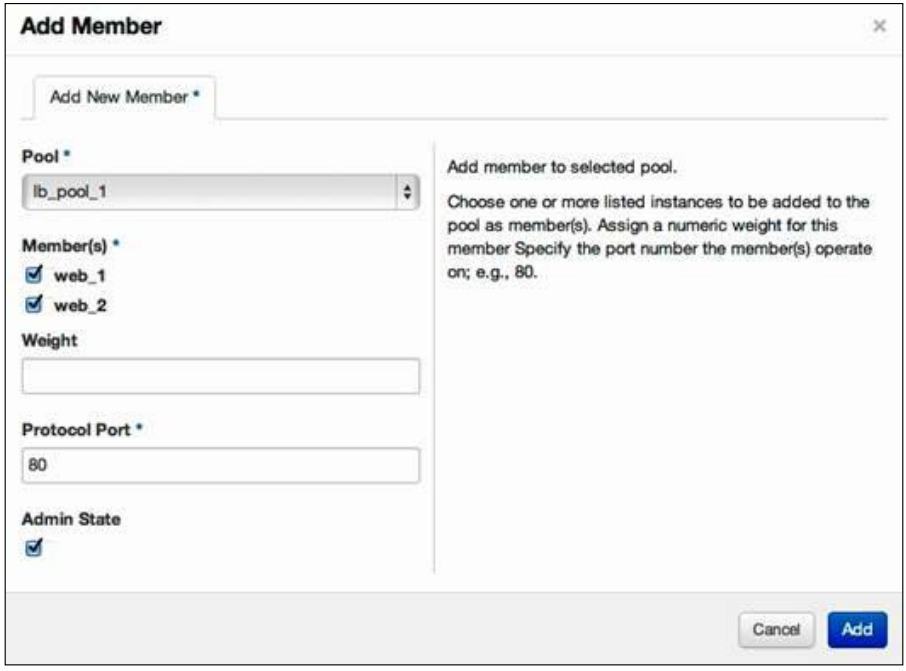
The SSH and HTTP access is given to the launched instances, and the security group must be configured as follows.



Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0 (CIDR)
Ingress	IPv4	TCP	80 (HTTP)	0.0.0.0/0 (CIDR)
Ingress	IPv4	ICMP	-	0.0.0.0/0 (CIDR)

Adding members to the load balancing pool

The add members interface will show the launched instances as follows:



The following attributes of load balancing are changed as per requirements. In this case, we have to change the values of the attributes as follows:

- **Pool = lb_pool_1**: This is the load balancing pool name.
- **Members = web_1 and web_2**: These are the two instances.
- **Protocol Port = 80**: The port between the VIP and the other member servers. This must match with the listening port of the instance.

Setting a sample web server

To get a list of the instances created in the previous section, use the following command:

```
nova list
```

```
openstack@opensatck-virtualBox:~$nova list
+--------------------------------------+---------+--------+------------+-------------+-----------------------+
| ID                                   | Name    | Status | Task State | Power State | Networks              |
+--------------------------------------+---------+--------+------------+-------------+-----------------------+
| 877c496c-309c-4588-b378-9f7491882b83 | web_1 | ACTIVE | None       | Running     | Private_Net10=10.10.10.2 |
| cb1d84cd-9a78-4cac-b54e-ada8cf5b7b2 | web_2 | ACTIVE | None       | Running     | Private_Net10=10.10.10.4 |
```

We need to do the `ssh` for an instance called `web_1`::

```
sudo ip netns exec qrouter-e9c35b5e-1778-431c-8b86-5f1a45dfafa9 ssh
fedora@10.10.10.2
```

```
openstack@openstack-virtualBox:~$sudo ip netns exec qrouter-e9c35b5e-1778-431c-8b86-5f1a45dfafa9 ssh fedora@10.10.10.2
The authenticity of host '10.10.10.2 (10.10.10.2)' can't be established.
RSA key fingerprint is bd:40:10:cd:21:96:5d:e6:ac:db:f8:17:20:14:1f:51.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.10.10.2' (RSA) to the list of known hosts.
```

To create a basic `index.html` file on the first instance (`web_1`), use the following command:

```
Sudo –i
Cat > index.html
```

```
[fedora@web-1 ~]$ sudo -i
[root@web-1 ~]# cat > index.html
HTTP/1.0 200 OK Web-1
^C
```

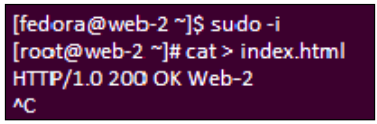
SSH into the first instance (`web_2`) using this:

```
sudo ip netns exec qrouter-e9c35b5e-1778-431c-8b86-5f1a45dfafa9 ssh
fedora@10.10.10.4
```

```
openstack@openstack-virtualBox:~$sudo ip netns exec qrouter-e9c35b5e-1778-431c-8b86-5f1a45dfafa9 ssh fedora@10.10.10.4
The authenticity of host '10.10.10.4 (10.10.10.4)' can't be established.
RSA key fingerprint is e0:8a:3c:b2:c8:68:93:45:c9:f6:28:fe:7b:df:b2:49.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.10.10.4' (RSA) to the list of known hosts.
```

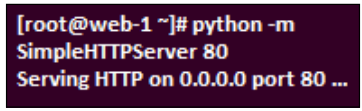
To create a basic `index.html` file on the first instance (`web_2`), use the following command:

```
Sudo –i
Cat > index.html
```



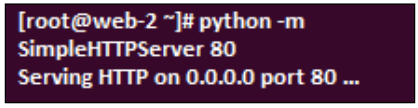
To run the Python simple server (the HTTP server) module that comes along with an instance (Fedora) on `web_1`, run the following command:

```
Python -m
```



On `web_2`, run this command:

```
Python -m
```

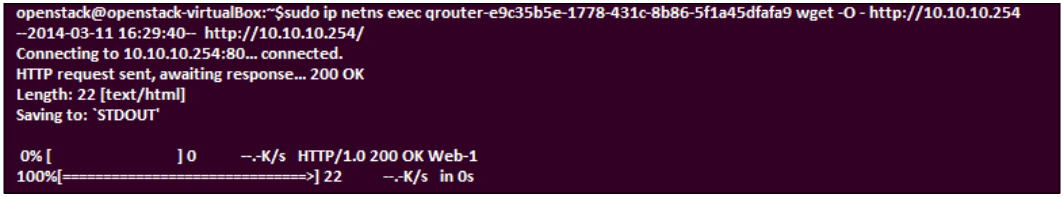


Validating web servers with index.html

Run `index.html` with the `WGET` command and ensure that the VIP is listening and can be used for load balancing on each instance.

On `web_1`, we need to run the following command:

```
sudo ip netns exec qrouter-e9c35b5e-1778-431c-8b86-5f1a45dfafa9 wget -O -
http://10.10.10.254
```



On `web_2`, we need to run this command:

```
sudo ip netns exec qrouter-e9c35b5e-1778-431c-8b86-5f1a45dfafa9 wget -O -
http://10.10.10.254
```

```
openstack@openstack-virtualBox:~$sudo ip netns exec qrouter-e9c35b5e-1778-431c-8b86-5f1a45dfafa9 wget -O - http://10.10.10.254
--2014-03-11 16:29:42--  http://10.10.10.254/
Connecting to 10.10.10.254:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 28 [text/html]
Saving to: `STDOUT'

 0% [                    ] 0         --.-K/s   HTTP/1.0 200 OK Web-2
100%[====================================>] 28          --.-K/s   in 0s
```

Summary

From this chapter, we have a better understanding of how OpenStack services work in a stateless mode in cooperation with one another to offer a scalable cloud framework. We have learned the step-by-step and detailed configurations setup of high availability compute services, dashboard, object storage, and image services and also, the load balancing of HTTP REST API.

In the next chapter, we will learn OpenStack networking services, which are the most difficult to scale and render highly available.

6
Distributed Networking

OpenStack networking services are the most difficult to scale and render highly available. This chapter will delve into the details of the Neutron **Distributed Virtual Routers** (**DVR**), multiple L3 agents in active/passive configuration, and third-party networking drivers that offer high availability options.

The DVR architecture expands the heritage architecture by giving direct connectivity to other external networks on compute nodes. Floating IP address-enabled instances route between project and external networks, live on the compute nodes to eliminate a single point of failure, and solve performance issues with heritage network nodes.

With a fixed or floating IP address, the routing also exists completely on the compute nodes for the instances that use networks on the same distributed virtual router. However, the fixed IP address instances depend on the network node for routing and for SNAT services between project and external networks.

In the Juno (initial) release, DVR supports VXLAN and GRE project networks. All other releases support traditional flat and VLAN external networks. In this chapter, we are going to learn the setting up of three different nodes such as controller node, compute node, and network node for the installation of a high availability **Distributed Virtual Routing** (**DVR**).

Installing a high availability distributed virtual routing

To install the Open VSwitch Level 3 High Availability, we need to set up the three different nodes such as controller node, compute node, and network node. This provides external connectivity on the compute nodes. The following sections explain the procedure for the installation of network services on these nodes.

Control node setup

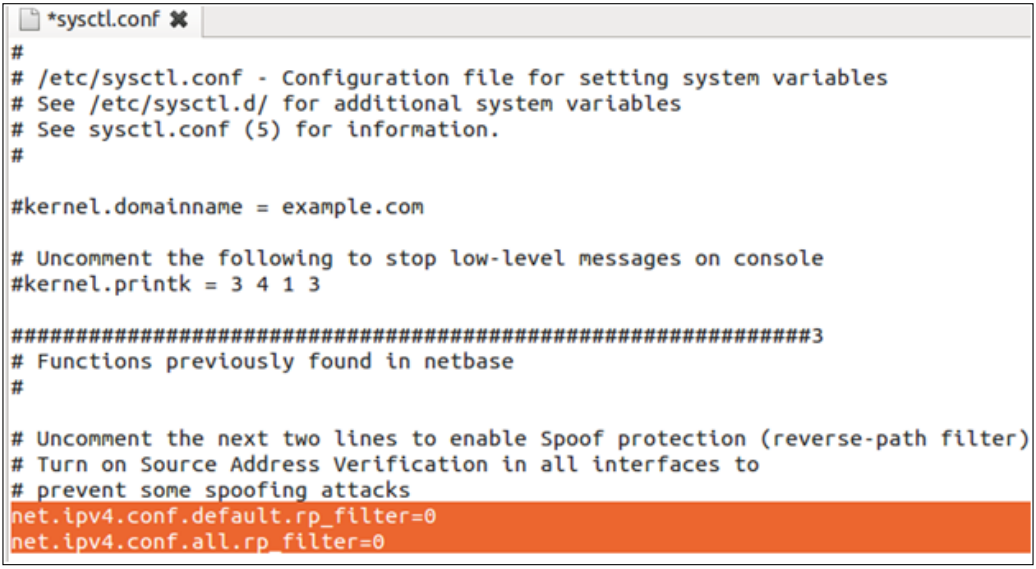
The following is the procedure for a control node setup:

- SQL server with neutron DB and the `neutron-server.conf` file must be configured
- The message queue service and the `neutron-server.conf` file configuration must be done according to the message queue server
- Configuration must be done in the `neutron-server.conf` file according to the OpenStack Identity Service
- The `Nova.conf` file is also configured to adopt the OpenStack compute services, neutron server service, and ML2 plugin and its dependencies

Disabling reverse path filtering

The kernel needs to be configured to disable the reverse path filtering by editing the `sysctl.conf` file under the `/etc` directory using any one of the editors (such as `vi`, `nano`, and `gedit`) and changing the values of the `net.ipv4.conf.default.rp_filter` and `net.ipv4.conf.all.rp_filter` attributes to zero:

```
| sudo gedit /etc/sysctl.conf
```



Loading a new kernel

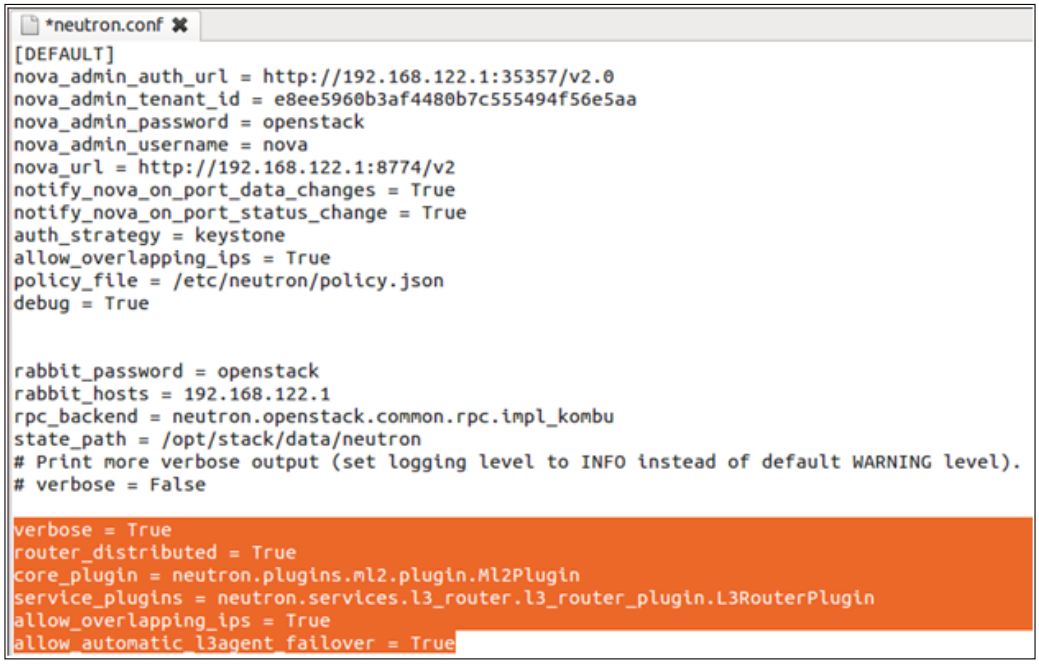
The new kernel must be loaded with the help of the configuration of the kernel file, as follows:

```
| sudo sysctl -p
```

Configuring the neutron

We need to configure the neutron file by changing the following base configurations using the `neutron.conf` file under `/etc/neutron`:

```
| sudo gedit /etc/neutron/neutron.conf
```



```
*neutron.conf
[DEFAULT]
nova_admin_auth_url = http://192.168.122.1:35357/v2.0
nova_admin_tenant_id = e8ee5960b3af4480b7c555494f56e5aa
nova_admin_password = openstack
nova_admin_username = nova
nova_url = http://192.168.122.1:8774/v2
notify_nova_on_port_data_changes = True
notify_nova_on_port_status_change = True
auth_strategy = keystone
allow_overlapping_ips = True
policy_file = /etc/neutron/policy.json
debug = True


rabbit_password = openstack
rabbit_hosts = 192.168.122.1
rpc_backend = neutron.openstack.common.rpc.impl_kombu
state_path = /opt/stack/data/neutron
# Print more verbose output (set logging level to INFO instead of default WARNING level).
# verbose = False

verbose = True
router_distributed = True
core_plugin = neutron.plugins.ml2.plugin.Ml2Plugin
service_plugins = neutron.services.l3_router.l3_router_plugin.L3RouterPlugin
allow_overlapping_ips = True
allow_automatic_l3agent_failover = True
```

Configuring the ML2 plugin

This ML2 plugin is used for the OpenStack networking service to develop numerous network-layering technologies, which are very difficult in the real-time data center networking. The networking technologies, such as Linux bridge, hyper 2 agents, and openvswitch are created to be used instead of traditional monolithic plugins related with L2 agents. Hence, ML2 simplifies the addition of all new networking technologies related to L2. We have to change the VLAN and VXLAN according to our environment, as follows:

*ml2_conf.ini

```
[ml2]
type_drivers = flat,vlan,vxlan
tenant_network_types = vxlan
mechanism_drivers = openvswitch,l2population
# (ListOpt) List of network type driver entrypoints to be loaded from
# the neutron.ml2.type_drivers namespace.
#
# type_drivers = local,flat,vlan,gre,vxlan
# Example: type_drivers = flat,vlan,gre,vxlan
# (ListOpt) Ordered list of networking mechanism driver entrypoints
# to be loaded from the neutron.ml2.mechanism_drivers namespace.
# mechanism_drivers =
# Example: mechanism drivers = openvswitch,mlnx
# Example: mechanism_drivers = arista
# Example: mechanism_drivers = cisco,logger
# Example: mechanism_drivers = openvswitch,brocade
# Example: mechanism_drivers = linuxbridge,brocade

[ml2_type_flat]
flat_networks = external

[ml2_type_vlan]
network_vlan_ranges = external:1001:2000

[ml2_type_vxlan]
vni_ranges = 1001:2000
vxlan_group = 239.1.1.1

[securitygroup]
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

Restarting the services

Restart the services to make changes over the servers using the following command:

```
sudo service openstack-neutron restart
```

A network node setup

The following is the procedure for a network note setup:

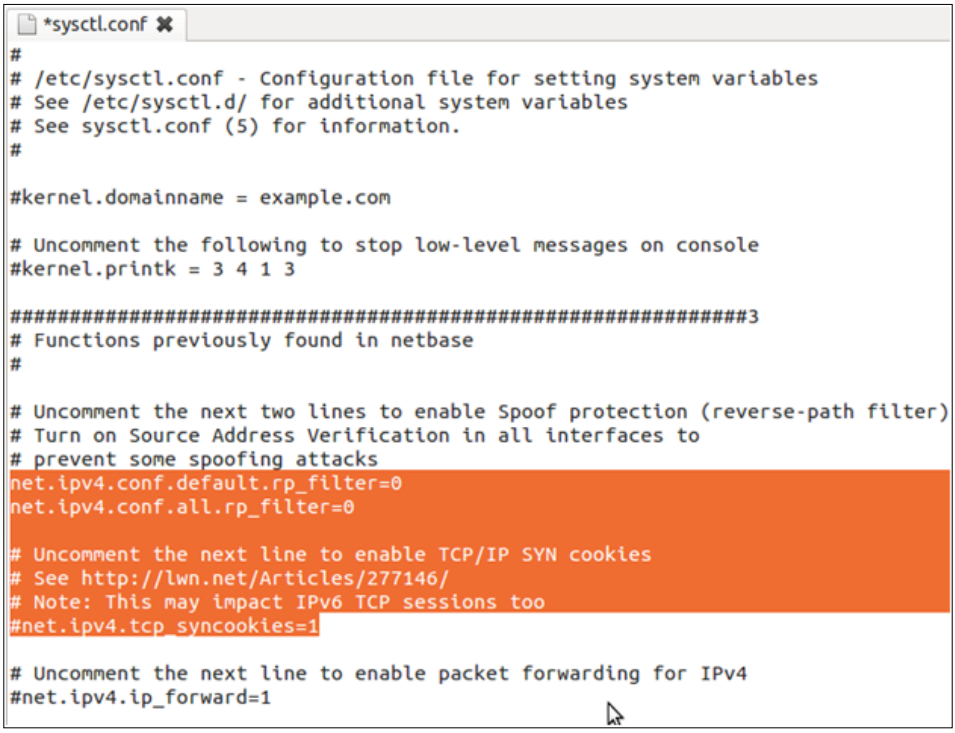
- Configuration must be done in the `neutron-server.conf` file according to the OpenStack identity service
- The ML2 plugin, the L3 agent, the Open vSwitch agent, the metadata agent, the open vSwitch service, the DHCP agent, and dependencies.

Enabling packet forwarding and disabling reverse path filtering

The kernel needs to be configured to disable the reverse path filtering and to enable packet forwarding by editing the file under the `/etc/` directory using any one of the editors (such as `vi`, `nano`, and `gedit`) and changing the values of the `net.ipv4.conf.default.rp_filter` and `net.ipv4.conf.all.rp_filter` attributes to zero, and then changing the value of an attribute called `net.ipv4.ip_forward` to `1`.

Edit the `/etc/sysctl.conf` file using following command:

```
sudo gedit /etc/sysctl.conf
```



Loading a new kernel

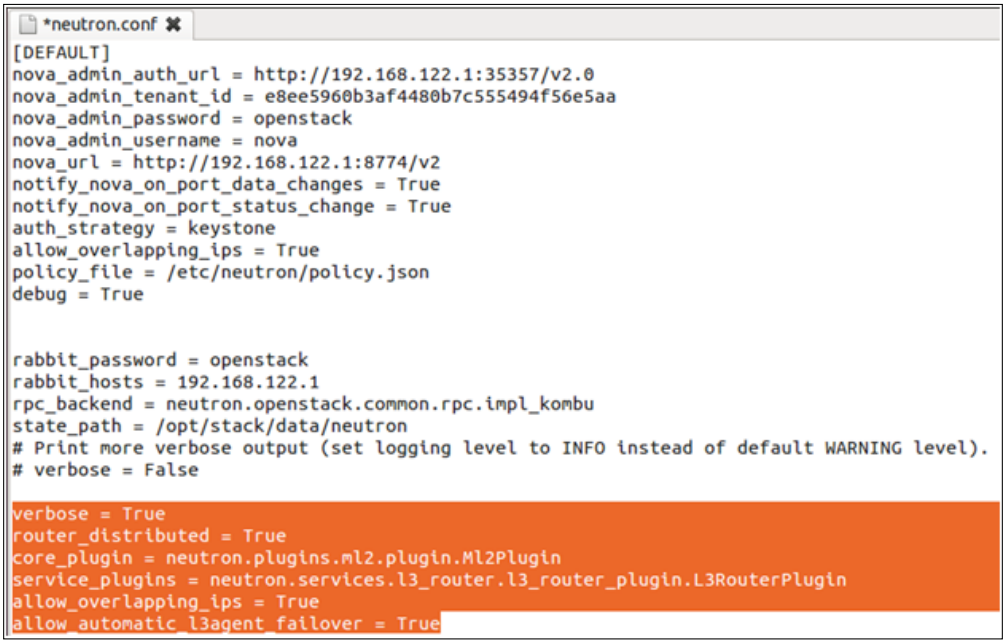
The new kernel must be loaded with the help of the configuration of the kernel file as follows:

```
| sudo sysctl -p
```

Configuring the neutron

We need to configure the neutron file by changing the following base configurations using the `neutron.conf` file under `/etc/neutron`:

```
| sudo gedit /etc/neutron/neutron.conf
```



Configuring the ML2 plugin

Configure the ML2 plugin using any one of these editors: `vi`, `nano`, and `gedit`, as follows:

```
| sudo gedit /etc/neutron/plugins/ml2/ml2_conf.ini
```

*ml2_conf.ini

```
[ml2]
type_drivers = flat,vlan,vxlan
tenant_network_types = vxlan
mechanism_drivers = openvswitch,l2population

[ml2_type_flat]
flat_networks = external

[ml2_type_vlan]
network_vlan_ranges = external:1001:2000

[ml2_type_vxlan]
vni_ranges = 1001:2000
vxlan_group = 239.1.1.1

[securitygroup]
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver

[ovs]
local_ip = TENANT_NETWORK_TUNNELS_INTERFACE_IP_ADDRESS
bridge_mappings = external:br-ex

[agent]
l2_population = True
tunnel_types = vxlan
enable_distributed_routing = True
arp_responder = True
```

We have to change the VLAN and VXLAN according to our environment, as shown in the following sections.

Configuring the L3 agent

Configure the L3 agent using one of the editors as follows and change `external_network_bridge` to no value:

```
| sudo gedit /etc/neutron/l3_agent.ini
```

*l3_agent.ini

```
[DEFAULT]
l3_agent_manager = neutron.agent.l3_agent.L3NATAgentWithStateReport
external_network_bridge = br-ex
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
ovs_use_veth = False
root_helper = sudo /usr/local/bin/neutron-rootwrap /etc/neutron/rootwrap.conf
use_namespaces = True
debug = True
verbose = True
router_delete_namespaces = True
agent_mode = dvr_snat
# Show debugging output in log (sets DEBUG log level output)
# debug = False
```

Configuring the DHCP agent

Configure the DHCP agent using any one of the editors, as follows:

```
sudo gedit /etc/neutron/dhcp_agent.ini
```

```
*dhcp_agent.ini
[DEFAULT]
dhcp_agent_manager = neutron.agent.dhcp_agent.DhcpAgentWithStateReport
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
ovs_use_veth = False
root_helper = sudo /usr/local/bin/neutron-rootwrap /etc/neutron/rootwrap.conf
use_namespaces = True
debug = True
verbose = True
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
dhcp_delete_namespaces = True
```

Configuring the metadata agent

Configure the metadata agent using any one of the editors, as follows, and change the `METADATA_SECRET` value according to the environment:

```
sudo gedit /etc/neutron/metadata_agent.ini
```

```
*metadata_agent.ini
[DEFAULT]
root_helper = sudo /usr/local/bin/neutron-rootwrap /etc/neutron/rootwrap.conf
nova_metadata_ip = 192.168.122.1
debug = True
verbose = True
# Show debugging output in log (sets DEBUG log level output)
# debug = True
metadata_proxy_shared_secret = METADATA_SECRET
# The Neutron user information for accessing the Neutron API.
auth_url = http://192.168.122.1:5000/v2.0
auth_region = RegionOne
# Turn off verification of the certificate for ssl
# auth_insecure = False
# Certificate Authority public key (CA cert) file for ssl
# auth_ca_cert =
admin_tenant_name = service
admin_user = neutron
admin_password = openstack
```

Restarting the services

The services (Open vSwitch, the Open vSwitch agent, the L3 agent, the DHCP agent, the Metadata agent, and neutron) are restarted, as follows to reflect the changes:

```
sudo service neutron-server restart
sudo service neutron-dhcp-agent restart
sudo service neutron-l3-agent restart
sudo neutron-metadata-agent restart
sudo service neutron-openvswitch-agent restart
```

A compute node setup

The following is the procedure for a compute node setup:

- Configuration must be done in the `neutron-server.conf` file according to the OpenStack identity service
- Configuration must be done in the `neutron-server.conf` file according to the OpenStack computer hypervisor service
- The ML2 plugin, the L3 agent, the Open vSwitch agent, the metadata agent, the Open vSwitch service, the DHCP agent, and dependencies

Enabling packet forwarding and disabling reverse path filtering

The kernel needs to be configured to disable the reverse path filtering and to enable packet forwarding by editing the file under the `/etc/` directory using any of the editors (such as `vi`, `nano`, and `gedit`) and changing the values of the `net.ipv4.conf.default.rp_filter` and `net.ipv4.conf.all.rp_filter` attributes to zero, and then changing the value of an attribute called `net.ipv4.ip_forward` to `1`.

Edit the `/etc/sysctl.conf` file using following command:

```
sudo gedit /etc/sysctl.conf
```

Loading a new kernel

A new kernel must be loaded with the help of the configuration of the kernel file, as follows:

```
sudo sysctl -p
```

Configuring neutron

We need to configure the neutron file by changing the following base configurations using a `neutron.conf` file under `/etc/neutron`:

```
sudo gedit /etc/neutron/neutron.conf
```

*neutron.conf

```
[DEFAULT]
verbose = True
router_distributed = True
core_plugin = ml2
service_plugins = neutron.services.l3_router.l3_router_plugin.L3RouterPlugin
allow_overlapping_ips = True
allow_automatic_l3agent_failover = True
nova_admin_auth_url = http://192.168.122.1:35357/v2.0
nova_admin_tenant_id = e8ee5960b3af4480b7c555494f56e5aa
nova_admin_password = openstack
nova_admin_username = nova
nova_url = http://192.168.122.1:8774/v2
notify_nova_on_port_data_changes = True
notify_nova_on_port_status_change = True
auth_strategy = keystone
allow_overlapping_ips = True
policy_file = /etc/neutron/policy.json
debug = True
```

Also, change the value of `router_distributed` to `true`.

Configuring the ML2 plugin

Configure the ML2 plugin using any one of these editors: `vi`, `nano`, and `gedit`, as follows:

```
sudo gedit /etc/neutron/plugins/ml2/ml2_conf.ini
```

*ml2_conf.ini

```
[ml2]
type_drivers = flat,vlan,vxlan
tenant_network_types = vxlan
mechanism_drivers = openvswitch,l2population

[ml2_type_flat]
flat_networks = external

[ml2_type_vlan]
network_vlan_ranges = external:1001:2000

[ml2_type_vxlan]
vni_ranges = 1001:2000
vxlan_group = 239.1.1.1

[securitygroup]
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver

[ovs]
local_ip = TENANT_NETWORK_TUNNELS_INTERFACE_IP_ADDRESS
bridge_mappings = external:br-ex

[agent]
l2_population = True
tunnel_types = vxlan
enable_distributed_routing = True
arp_responder = True
```

We have to change the VLAN and VXLAN according to the environment, as explained in the following sections.

Configure the L3 agent

Configure the L3 agent using one of the editors, as follows and change the `external_network_bridge` option to have no value:

```
sudo gedit /etc/neutron/l3_agent.ini
```

*l3_agent.ini

```
[DEFAULT]
l3_agent_manager = neutron.agent.l3_agent.L3NATAgentWithStateReport
external_network_bridge = br-ex
verbose = True
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
use_namespaces = True
external_network_bridge =
router_delete_namespaces = True
agent_mode = dvr
# Show debugging output in log (sets DEBUG log level output)
# debug = False
```

Configuring the metadata agent

Configure the metadata agent using any one of the editors and change the `METADATA_SECRET` value according to the environment, as follows:

```
sudo gedit /etc/neutron/metadata_agent.ini
```

*metadata_agent.ini

```
[DEFAULT]
root_helper = sudo /usr/local/bin/neutron-rootwrap /etc/neutron/rootwrap.conf
nova_metadata_ip = 192.168.122.1
debug = True
verbose = True
# Show debugging output in log (sets DEBUG log level output)
# debug = True
metadata_proxy_shared_secret = METADATA_SECRET
# The Neutron user information for accessing the Neutron API.
auth_url = http://192.168.122.1:5000/v2.0
auth_region = RegionOne
# Turn off verification of the certificate for ssl
# auth_insecure = False
# Certificate Authority public key (CA cert) file for ssl
# auth_ca_cert =
admin_tenant_name = service
admin_user = neutron
admin_password = openstack
```

Restarting the services

The services (Open vSwitch, the Open vSwitch agent, the L3 agent, and the Metadata agent) are restarted, as follows, to reflect the changes:

```
sudo service neutron-server restart
sudo service neutron-l3-agent restart
sudo neutron-metadata-agent restart
sudo service neutron-openvswitch-agent restart
```

Verifying the service operation

The presence and operations of the agents are verified with following command:

```
sudo neutron agent-list
```

```
openstack@openstack-VirtualBox:~/devstack$ neutron agent-list
+--------------------------------------+--------------------+---------------------+-------+----------------+
| id                                   | agent_type         | host                | alive | admin_state_up |
+--------------------------------------+--------------------+---------------------+-------+----------------+
| 374c08b5-d7a5-4b11-9276-56ab745a9c9f | Open vSwitch agent | openstack-VirtualBox | :-)   | True           |
| 78bd4bf2-d45e-4cbc-aa18-dbe42cda82ad | DHCP agent         | openstack-VirtualBox | :-)   | True           |
| f7200ca9-ff30-4234-9ad5-523f1a1ff08a | L3 agent           | openstack-VirtualBox | :-)   | True           |
| fd8f77c0-9761-492e-a9c5-6c73aa9a4a86 | Metadata agent     | openstack-VirtualBox | :-)   | True           |
+--------------------------------------+--------------------+---------------------+-------+----------------+
```

Summary

In this chapter, we have learned the setting up of the controller node, compute node, and network node for the installation of a high availability distributed virtual router. which includes disabling the reverse path filtering, the loading of the new kernel, the configuration of Neutron, the ML2 plugin, the L3 agent, the DHCP agent, and the metadata agent.

In the next chapter, we are going to learn about the configuration of different shared storage options for OpenStack.

7
Shared Storage

For any OpenStack operator who strives to bring the cloud they manage to a higher availability, understanding all the different ways that a failure can occur and how these can impact the performance and availability of the service is a fundamental skill. This chapter is a fundamental requirement for high availability and quick recovery from a failure. In this chapter, different options to provide shared storage to OpenStack will be presented, and their configuration and setup will be explained to some extent.

In this chapter, we will cover the following main topics:

- An introduction to GlusterFS
- Installing GlusterFS
- An introduction to Ceph
- Installing Ceph

An introduction to GlusterFS

GlusterFS is designed for today's high-performance, virtualized cloud environments. Unlike traditional data centers, cloud environments require multitenancy along with the ability to grow or shrink resources on demand.

Installing GlusterFS

The block storage service of OpenStack is an iSCSI solution that uses a **Logical Volume Manager** (**LVM**). The volumes provided by this service will be attached to only one instance at a time. However, OpenStack offers drivers for using different backend storage from different vendors. NFS is a conventional method used as a backup store for OpenStack, such as GlusterFS and Ceph.

In the following section, we will see how GlusterFS is configured as backend storage for OpenStack block storage. Hence, cinder volumes services will be accessed from their server for GlusterFS.

Configuring GlusterFS for block storage

Even though OpenStack provides most of the services to host the virtual machines on its own cloud, it still depends on some other third-party tools to support the OpenStack backend storage for cinder services.

The following sections explain the steps for the configuration of GlusterFS.

Installation of GlusterFS

The following command must be typed on the terminal to install GlusterFS:

```
| sudo apt-get install glusterfs-server
```

Then, ensure that GlusterFS was properly installed on the two nodes (`controller_1` and `controller _2`) using the following command:

```
| sudo glusterfs --version
```

After a successful installation, we will get the following result:

```
openatck@openstack:~# glusterfs --version
glusterfs 3.4.2 built on Jan 14 2014 18:05:35
Repository revision: git://git.gluster.com/glusterfs.git
Copyright (c) 2006-2013 Red Hat, Inc. <http://www.redhat.com/>
GlusterFS comes with ABSOLUTELY NO WARRANTY.
It is licensed to you under your choice of the GNU Lesser
General Public License, version 3 or any later version (LGPLv3
or later), or the GNU General Public License, version 2 (GPLv2),
in all cases as published by the Free Software Foundation.
```

Configuring the nodes for communication

Now both the servers want to communicate with each other by editing the entries in `/etc/hosts` using the following command:

```
| sudo gedit /etc/hosts
```

Now we need to run the following command on both the nodes:

```
sudo gluster peer probe gluster1
sudo gluster peer probe gluster2
```

After a successful communication between these nodes, we will ensure that the `GlusterFS` nodes are properly communicating with each other.

The status of peers

Now we have to check the status of peers on the two servers. On server 1, we have to check the peer status by the following command:

```
sudo gluster peer status
```

```
openstack@opensatck:~$ sudo gluster peer status
Number of Peers: 1
Hostname: gluster1
Port: 24007
Uuid: 8d865314-af12-4950-a784-6a5308ec501b
State: Peer in Cluster (Connected)
```

On server 2, we have to check the peer status by the following command:

```
sudo gluster peer status
```

```
openstack@opensatck:~$ sudo gluster peer status
Number of Peers: 1
Hostname: gluster2
Port: 24007
Uuid: 5c5a045c-34b9-44ac-b5c0-8acb461d8523
State: Peer in Cluster (Connected)
```

Creating a data point

As an initial step in data point creation, we need to create a new directory called `gluster` under a `/mnt/gluster` directory using the following command:

```
| sudo mkdir -p /mnt/gluster
```

Then we have to create a volume in which we will have all the other data residing in it. This volume is called a data point. After this, we will run the following command on the machines (that is, the servers):

```
| sudo gluster  volume create datapoint replica 2 transport tcp
gluster1:/mnt/gluster  gluster2:/mnt/gluster
```

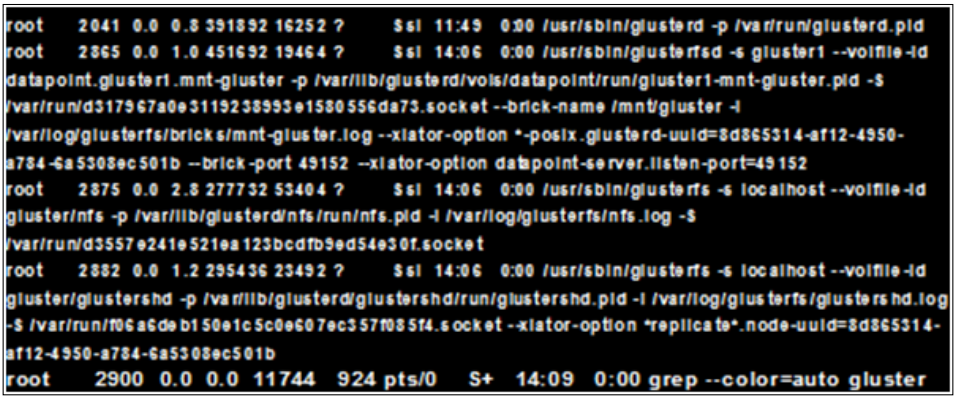
Starting the volume services

We start the data point volume services using the following command:

```
| sudo gluster volume start datapoint
```

Finally, we have to run the commands to ensure that the `GlusterFS` is up and running, using `PS` on both of the nodes (`controller_1` and `controller_2`):

```
| sudo ps aux | grep gluster
```



To make sure of the availability of the volumes, run the following `gluster` command:

```
sudo gluster volume info
```

An introduction to Ceph

Ceph is an open source, scalable, and software-defined object store system, which provides object, block, and file system storage in a single platform. Ceph has a capability to self-heal, self-manage, and does not have a single point of failure. It is a perfect replacement for a traditional storage system and an efficient storage solution for the object and block storage of cloud environments.

Installing Ceph

The following command is used to install `ceph`:

```
sudo apt-get update && sudo apt-get install ceph-deploy
```

Installing Openssh

`Openssh` is installed using the following command:

```
sudo aptitude -y install openssh-server
```

The required privileges are provided to `ceph` using the following command:

```
sudo chmod 440 /etc/sudoers.d/ceph
```

Connecting to the Ceph node

An `ssh` pair is created and sent to be connected to the `ceph` nodes with non-passphrase with a command called | `sudo ssh-keygen`. Then the configuration file of the `.ssh` file is to be edited as follows:

```
| sudo vi ~/.ssh/config
```

```
openstack@openstack:~$ sudo vi ~/.ssh/config
# create new ( define all Ceph Nodes and user )

Host ceph-mds
   Hostname ceph-mds.server.world
   User trusty
Host ceph01
   Hostname ceph01.server.world
   User trusty
Host ceph02
   Hostname ceph02.server.world
   User trusty
Host ceph03
   Hostname ceph03.server.world
   User trusty
```

Then the `ssh` key is sent to the `ceph` nodes as follows:

```
openstack@openstack:~$ sudo ssh-copy-id ceph01

The authenticity of host 'ceph01.server.world (10.0.0.81)' can't be established.
ECDSA key fingerprint is xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:2b:4b:6e.
Are you sure you want to continue connecting (yes/no)?
yes

/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any
that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now
it is to install the new keys
trusty@ceph01.server.world's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'ceph01'"
and check to make sure that only the key(s) you wanted were added
```

This key is sent to all other nodes using the following commands:

```
| sudo ssh-copy-id ceph02
| sudo ssh-copy-id ceph02
```

Configuring the Ceph node

The `Ceph` configuration must be done on the servers as follows:

```
| sudo aptitude -y install ceph-deploy ceph-common ceph-mds
```

```
openstack@openstack:~$ sudo aptitude -y install ceph-deploy ceph-common ceph-
mds

openstack@openstack:~$ mkdir ceph

openstack@openstack:~$ cd ceph
# configure cluster
openstack@openstack:~$ ceph-deploy new ceph01 ceph02 ceph03

# install Ceph on all Nodes
openstack@openstack:~$ ceph-deploy install ceph01 ceph02 ceph03

# initial configuration for monitoring and keys
openstack@openstack:~$ ceph-deploy mon create-initial
```

Configuring a storage node

Now we configure the storage node on the server. In our scenario, we have to create the `/storage01`, `/storage02`, and `/storage03` directories on the `ceph01`, `ceph02`, and `ceph03` nodes, as follows:

```
| sudo  ceph-deploy osd prepare ceph01:/storage01 ceph02:/storage02
ceph03:/storage03
```

```
| sudo ceph-deploy osd activate ceph01:/storage01 ceph02:/storage02
ceph03:/storage03
```

```
# prepare Object Storage Daemon
opensatck@openstack:~/ceph$
ceph-deploy osd prepare ceph01:/storage01 ceph02:/storage02 ceph03:/storage03


# activate Object Storage Daemon
opensatck@openstack:~/ceph$
ceph-deploy osd activate ceph01:/storage01 ceph02:/storage02 ceph03:/storage03


# Configure Meta Data Server
opensatck@openstack:~/ceph$
ceph-deploy admin ceph-mds

opensatck@openstack:~/ceph$
ceph-deploy mds create ceph-mds
```

Checking the status of Ceph

Finally, we check the status of `ceph` with the following command:

```
| sudo ceph mds stat
```

On successful execution of the preceding command, we get the status of `ceph`, as follows. Then the live migration must be implemented on each compute node available in the OpenStack cloud environment. Through this, we can recover from node failures with a host evacuation:

```
# show status
openstack@openstack:~/ceph$
ceph mds stat
e4: 1/1/1 up {0=ceph-mds=up:active}
openstack@openstack:~/ceph$
ceph health
HEALTH_OK
```

Summary

In this chapter, we have learned the installation and usage procedures of GlusterFS and Ceph that provide us with a shared storage for OpenStack, their configuration and the detailed steps for setup.

In the next chapter, we will analyze different failure scenarios and propose solutions to provide a swift and effective recovery to achieve a normal operational level.

8
Failure Scenario and Disaster Recovery

In this chapter, we will cover the context of recovering from different failure scenarios. This includes network partition split-brain, automatic failover, and geo-replication.

We are going to learn the following topics in this chapter:

- Network partition split-brain
- Automatic failover
- Geo-replication

Network partition split-brain

When two or more replicated copies of a file in two different clusters become divergent and independent from each other, then such a situation is called split-brain. In this situation, the members of the two clusters assume that the other nodes are dead even though they are not dead in their own cluster. Because of this network failure, there is no way to avoid this situation programmatically. So cluster will run as two different independent clusters. When this spilt-brain occurs, the applications will not be able to do their operations, such as file read and write, on the clusters.

There are numerous split-brain situations in real time. Split-brain occurs at various levels such as storage and file. For example, the automatic healing of a split-brain situation is not possible since the contents of the file a split-brain are different. The only way to resolve a split-brain is to manually inspect the file contents and decide on the true copy of the file called as the source copy. The modified copy is called as sink copy.

Preventing a split-brain

The best way to prevent a split-brain is by enabling the quorum enforcement on the server and client. As an initial step of prevention, the nodes in any one of the partitions must stop running to avoid inconsistencies.

Setting the server-side quorum

The quorum enablement on a particular volume to participate in the server quorum is as follows:

```
| sudo gluster volume set VOLNAME cluster.server-quorum-type server
```

We have to configure the quorum percentage ratio for a trusted storage pool as follows. The bricks of the volume that are participating in the quorum on all the nodes go offline when network outages occur and if the quorum is not met:

```
| sudo gluster volume set all cluster.server-quorum-ratio 51%
```

51% of the nodes are on a storage pool online and the pool's network connectivity is given to the nodes. On disconnecting the storage pool from the network, the bricks that are running on the nodes will prevent write operations from running.

Setting the client-side quorum

We have to change the quorum-type value to auto. So this will permit write operations to all the files only if the percentage of active replicate bricks is more than half the percentage of the total nodes that constitute the replica. This is done using the following command:

```
| sudo gluster volume set VOLNAME quorum-type auto
```

In this example, any one of the two bricks in the replica must be running to allow a write operation.

A real-time failure scenario of split-brain

The split-brain real-time scenario is explained in this section by considering DRBD.

In this, after the detachment of the replication network and in case the default procedure is set for fencing, this will lead to an occurrence of split-brain.

The divergence occurred because the contents of the backup devices of the distributed replicated storage system's resources are available on both clusters.

After this, the write operations will have to be done on the both sides of a cluster.

On a successful reconnection, the DR will not be able to find the right and wrong set of data in this operation.

In this DRBD failure scenario, the peer will not reconnect to the DRDB in the beginning, but it will be available in a state of connection, that is, in a state of standalone or WFC Connection. We will receive the kernel log messages, which is similar to what we received in the split-brains:

```
kernel: block drbd0: Split-Brain detected, dropping connection!
```

This means that the split-brain is detected on priority and the connection is shut down.

Steps to resolve a split-brain

The following are the steps that we need to follow when there is an occurrence of a split-brain situation.

Choosing a split-brain victim

Suppose we have to `opt` for a node and if there is any modification in it, it will be discarded manually. We call this as a split-brain victim node. Then we need to enable the maintain mode when we run the pacemaker cluster. If the victim is in a primary role, bring down all the applications that are using this resource. After this, the victim must be switched to a secondary role as follows:

```
| victim# drbdadm secondary resource
```

When the WF Connection of a state occurs in the connection of a resource, we can disconnect it with the help of the following command:

```
| victim# drbdadm disconnect resource
```

Force discard of the victim

With the following command, the changes required on the victims must be done as a force discard:

```
. | victim# drbdadm -- --discard-my-data connect resource
```

Resynchronization

We begin the resynchronization process automatically when WFC Connection is in the network state. If the survivors available in a split-brain are in a standalone state of connection, then it will reconnect with the following commands:

```
| survivor# drbdadm connect resource
```

Then the victims of the spiltbrain will immediately get started. The SyncSource data is used to overwrite the victims.

Automatic failover

Network as a Service (**NaaS**) is a layer of OpenStack with the name `neutron`. The plugin of Neutron is the **Load Balancer as a Service** (**LBaaS**) that offers an abstraction layer. Network as a service is one of the important services provided by OpenStack. This network layer is provided under the name of neutron, which is the technical name of the OpenStack network service. The Load Balancer as a service provides an abstraction layer via plugins. This is used to handle communication with other load balancers. It is feasible to configure the LBaaS module with the diverse drivers of different load balancers.

The Load balance and the neutron server network host run on the same host. This acts as a gateway to our own cloud. For the production system, the high availability must be ensured when none of the instances running on the cloud are reachable.

This scenario is a level 1 incident, and each administrator or system architect must try to eliminate such **Single Point of Failure** (**SPOF**) services to guarantee a maximum accessibility for the cloud.

Load balance as a service

The load balance service does not have an automated and integrated failover for all the load balancing driver modules. Redware is a load balancer driver, which supports the Havana HA functionality. A VRP protocol such as keepvid is a layer 7 load balancer that is not supported by LbaaS.

The neutron and LBaaS will work on an HA LBaaS agent with a **virtual router redundancy protocol** (**vrrp**) that supports most software load balancer plugins, but this characteristic is unfortunately not ready yet.

Another move towards a small development overhead is modifying the responsible LBaaS—an agent of the LB instances.

We describe the procedure to make the LBaaS highly available in a significant duration of time. This is a process similar to the general high availability implementation of Neutron. The load balance instance is changed to another instance, yet when running agents of the LBaaS with a pacemaker and for a shared storage, the `config` files do not have the capability to create a new LBaas instance that can be supported with an explicit load balancer agent.

A couple of LBaaS agents are deployed on a single server to get the high availability. A storage volume must be attached to each networking server. As the LB agents start all the instances, the directory files are available on both the servers using stored and shared configurations:

```
/var/lib /neutron /lbaas/
```

The working of a failover

A working of a failover is described here with CLI tools and an API calling example. On breaking the host and LBaaS that are running together, the pacemaker will control the resources such as the failover of an L3 agent and LBaaS. These two resources are implemented for a working failover:

```
| sudo l3 agent failover
```

This works in a very simple way, as the failover of running routers is handled and ports are attached with the instances. If any node fails, we must detach it from the ports.

Getting all the failed routers

As we seen in the following code:

```
| sudo neutron router-list-on-l3-agent
```

The following two methods are used to get the agent id:

- Each pacemaker `config` file is deposited with the id.
- Then, the good server host name is read and its agent id is got. The same is done for the load balance as a service host through the neutron API call.

  ```
  | sudo neutron agent-list | grep L3
  ```

With the pacemaker, the failover must be done for a situation where all the routers fail for the L3 agent. Then, detaching and attaching of all the routers to the running L3 agent takes place.

An LBaaS agent failover

The previous L3 agent failover method cannot carry out the load balancing service through the Application Programming Interface, because only a few functions are available here. So, we need to change the DB entries. For a better data loss recovery, users must take a backup of the old DB entries while we change the DB entries. There are plenty of ways to do so for the database credentials. All the preceding information resides on all the servers and in the `config` files of each agent. The OpenStack OSLO Python library is the best choice to get all the available and required information.

If we have a working DB connection to the Neutron database, we will need to revise some entries in the load balancer agent binding pool table of the Neutron database. Then, all the node ids are changed from a good to a bad state.

For example, see the following:

```
bad agent id: c784b7fb-8094-4d3b-a8b1-804d90a80784

good agent id: 7e7700a3-02b2-4bd3-9c45-eca938c3f975

update poolloadbalanceragentbindings set agent_id='7e7700a3-02b2-4bd3-
9c45-eca938c3f975' where agent_id='c784b7fb-8094-4d3b-a8b1-804d90a80784';
```

We can use the API call example of the L3 agent failover to get both agent IDs. When this is done, we can restart the agents on the servers that are termed as good, and all the agent instances will be spawned with VIP ports. These L3 agents switch between the routers and the agents before we start the LBaaS agent.

Geo-replication

Geo-replication is a continuous, distributed, synchronous, and incremental replication service used to replicate from one host/site to another. The mirror process and the replication process are done among the master and slave agents using a graphical replication approach to represent the master and slave.

The master stores volumes. The slaves store local volumes or remote volumes.
The replication of data across the geographically distributed storage pools is done through mirroring the data. Thus, this produces a backup of data for disaster recovery.

Creating geo-replication sessions

To create a common `pem` file, we need to run the following command on the master, and it should have a password-less SSH connection configuration:

```
gluster system:: execute gsec_create
```

Create the geo-replication session using the following command:

```
| gluster volume geo-replication MASTER_VOL SLAVE_HOST::SLAVE_VOL create
push-pem [force]
```

To set up the `pem` file on the slaves or slave nodes, the `pem` option is required. For example, see the following:

```
|  gluster volume geo-replication master-vol example.com::slave-vol
create push-pem
```

Verify the status of the created session by running the following command:

The master and slave volumes of geo-replication are created and the status of these volumes is checked via the following command:

```
| gluster volume geo-replication MASTER_VOL SLAVE_HOST::SLAVE_VOL status
```

Starting geo-replication

To start geo-replication, use one of the following commands, and as an initial step we need to start a geo-replication session between the hosts:

```
#gluster volume geo-replication master-vol example.com::slave-vol start
```

The command will start the geo-replication on all the nodes because they are a part of the volumes of the master node. Even if a node is lost, the command will be successful because the node is a part of a volume of the master node. The replica pair that is used for the replication session must be active on any one of the replica nodes and will be passive on all remaining nodes.

On a successful execution of the command, it will take some more minutes for the entire session to initialize and become stable. Then, we need to start the geo-replication session forcefully between the hosts:

```
# gluster volume geo
```

For example, type the following command:

```
| gluster volume geo-replication master-vol example.com::slave-vol start
force
```

As a part of the master volume, the preceding command will force start the geo-replication on the nodes. This command should be able to start the geo-replication sessions on many nodes when there is a lack of ability to successfully initiate the geo-replication session. The previous command is used to start over a replication session when the session has terminated or it has not started.

Verifying a successful geo-replication deployment

In a successful geo replication environment, the status of the replication is given the following command:

```
# gluster volume geo-replication MASTER_VOL SLAVE_HOST::SLAVE_VOL status
```

For example, type the following command:

```
| gluster volume geo-replication master-vol example.com::slave-vol status
```

A real-time failure scenario

The location of log files is an important issue in Gluster geo-replication that will occur in real time. In a cluster slave in a general volume, the sessions of this replication try to associate their log files to the slave volumes. The master file monitors the volume of the master, and initiation of changes in the slaves is done using the slave files. The mount point that is used for the master volume monitor is maintained using the `master0gluster` file.

Issues in the master log file

To do geo-replication, the master log file is retrieved via the following command:

```
| gluster volume geo-replication <MASTER><SLAVE> config log-file
```

For example, `voulme1` is considered to be a volume for a sample master and slaves such as example.com:

```
| gluster volume geo-replication Volume1 example.com:/data/remote_dir
config log-file
```

Issues in the Slave log file

When we get log files on the slave nodes, the geo-replication is done through the following commands:

```
The command to run geo-replication on the master node of the cluster:
|gluster volume geo-replication Volume1 example.com:/data/remote_dir
config session-owner5f6e5200-756f-11e0-a1f0-0800200c9a66
```

The details of the owner are displayed in the session as follows:

```
| gluster volume geo-replication /data/remote_dir config log-file
/var/log/gluster/${session-owner}:remote-dir.log
 The session details of the owner is replaced with the following command:
/var/log/gluster/5f6e5200-756f-11e0-a1f0-0800200c9a66:remote-dir.log
```

Issue in data synchronization

Data synchronization on the available host and remote machines is achieved through geo replication. At this time, we may get an issue with data synchronization. The solution to sort out this problem is also given in this section. We start with a real-time data synchronization issue as follows.

Description: GlusterFS geo-replication displays the status as OK, but the files are not synced, only directories and `symlink` is synced with the error messages in the log as follows:

```
[2011-05-02 13:42:13.467644] E [master:288:regjob] GMaster: failed to
sync ./some_file`
```

Solution: GlusterFS geo-replication invokes `rsync v3.07` in the host and the remote machine to check whether we have the desired version installed.

Issues in the geo-replication status display

A regular checking of status is also an important issue of display notification. This status may be updated and displayed very often to know the status of the geo-replication immediately. In this section, we will provide a real-time scenario of Display notification of Geo-replication.

Description: GlusterFS geo-replication displays the status as faulty very often with a back trace similar to the following:

```
2015-08-07 14:06:18.378859] E [syncdutils:131:log_raise_exception]
<top>:
FAIL: Traceback (most recent call last):
File
"/usr/local/libexec/glusterfs/python/syncdaemon/syncdutils.py'',
line 152, in twraptf(*aa)
File "/usr/local/libexec/glusterfs/python/syncdaemon/repce.py'',
line 118, in listen rid, exc, res = recv(self.inf)
File "/usr/local/libexec/glusterfs/python/syncdaemon/repce.py'',
line 42, in recv return pickle.load(inf)
EOFError
```

Solution: This means that the remote procedure call communication between the master module and slave module is divided, and this can happen for various reasons. The following pre-requisites should be satisfied to check for proper communication among the clusters:

- The host and remote nodes or machines are setup with a password-less secure socket host.
- With the help of synchronization of the data called sync data, the geo-replication module can mount the Gluster volume on the machine on which the FUSE is installed. We need to check whether the volume gets started if the slave survives as a volume.
- Then check all the necessary permissions of the directory that is created as a plain directory of the slave.

- If the machine is installed with the 3.2 version of glusterFS, then this will have the master as the default location and will be prefixed to the customized location of master using `gluster` command for customized location where `gsyncd` is available.

Summary

In this chapter, we learned about network partition split-brain, the different methods of preventing split-brain, a real-time failure scenario for split-brain, automatic failover including LBaaS agent failover, and had a detailed understanding of geo-replication with a real-time failure scenario.

In the next chapter, we are going to learn about application design for high availability.

9
The Principles of Design for Highly Available Applications

Having a highly available cloud might not be enough if the application running on top of it does not take advantage of the principles and concepts of resilient design. This chapter will mainly explain how a correct application design can help improve the reliability and uptime of end user services; particular focus will be given to micro services architectures and distributed web applications.

In this chapter, you are going to learn the following topics:

- The principles of design features
- A sample application deployment
- An interaction of the application with OpenStack

The principles of design features

There are numerous design features and principles that need to be considered for the applications that are deployed in the OpenStack cloud. The following are the design principles of the distributed web application deployment.

Micro services and scalability

A software architecture style or design pattern helps to support application modularity. With this pattern, the maintainability and reusability of the independent services are achieved. In this, a complex application is composed of independent processes. The communication among the processes is via special APIs. So this process of decoupling can scale out all the unique components as required. This will ensure the features of scale out and fault tolerance in cloud applications are available.

Generally, cloud applications are run on instances provided by the cloud infrastructure. Instead of utilizing more number of instances on which the applications are, running them utilizes only a few small instances. Modularity is ensure if this capacity with micro services architectures is utilized for the applications that are running on small instances. Hence, the scalability of the running applications is increased, as there is a need for more capacity to be available on all instances.

Fault tolerance

Fault tolerance is an important feature of cloud computing to achieve resilience when there is a change in the environment. In cloud computing, large and expensive servers are replaced by small virtual machines to cut down the cost of server maintenance and human resources. These virtual machines are disposable when they are not being used by the cloud automation capability. When something goes wrong with the virtual machines, the cloud automation processes will shut down the virtual machines and spin up new machines. In the traditional cloud approach, we have an unavoidable situation of failures, which includes underpinning the resources of the cloud infrastructure. Hence, we will design our application with a higher degree of fault tolerance capability that can adapt to all the changes in the cloud.

Cloud automation

Applications that are running on the cloud can scale up and scale down automatically to meet demand. We do not have too many steps for any component deployment process to run the applications on the cloud. The automation reduces the time taken for recovery of the application when it faces any component failures. Thus, resilience and fault tolerance will increase automatically.

RESTful application programming interface (APIs)

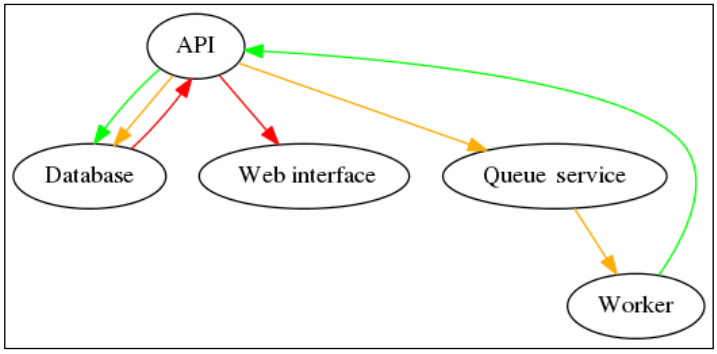
In this OpenStack cloud environment, all the applications that are running on cloud is called as cloud applications. A representational state transfer API is used in fractal applications such as other cloud applications. To connect with the API directly, we must integrate with the cloud components. Thus, we can improve the software quality and we can achieve feasible automated testing.

A sample application deployment

To facilitate the previously mentioned design principles in a real-time cloud deployment, we consider a sample real-time fractal application. This cloud application is used to generate some fractals in which mathematical equations are used. The micro service architecture is used in this application to decouple the application's logic functions so that we can easily handle the changes in independent functions.

The following figure shows this application's architecture. It consists of various components, such as the following:

- Application programming interface
- Database
- Web interface
- Queue service
- Worker services



Fractal Application Service Architecture

The application programming interface

These types of componenst are support with API, by which we can connect to the applications that are running on the OpenStack infrastructure directly

Database

This database is used to service from the other components, and we can get an idea of how these components are communicating with each other. All these API-related services are stored in this database.

Web interface

This web interface is used to work with APIs. Access to all the APIs is done through this interface to view the complete details of repeating pattern images using this cloud application.

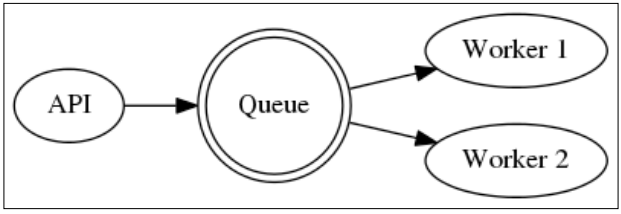
Queue services

Message queue services are used to communicate between the services of the queue service cloud application. This work queue is used to disseminate the tasks among the various work services.

In our application, the message queue offers requests of works that are brought from the work services one at a time even though we will have a unique service or multiple services. This type of cloud application has a request list. All these requests are serviced by any one of the resources that is available in the resource pool.

Worker services

The worker service will receive messages from the queues of a worker as shown in the following figure. Then the messages are processed to create a fractal file that consists of equivalent images. In addition, we can access these images on the web interface provided by OpenStack.



Work Service Request Process

An interaction of the application with OpenStack

We have an assumption of access to the OpenStack cloud. Our application is running on the instances provided through this cloud infrastructure.

Choosing the OpenStack SDK

As an initial step of this process, the application that is interacting with the OpenStack cloud infrastructure has to choose the OpenStack **software development kit** (**SDK**) according to the language we prefer to develop and deploy. The OpenStack SDK is available in various languages such as Python, Java, Ruby, PHP, and Node.js.

Flavors and images

We have to choose the flavor and the type of images that are used to create an instance on which our fractal application is running. The flavors are used to fix the size of an instance, which comprises of a number of VCPU, RAM capacity, and hard disk capacity. As shown in the following command:

```
<NodeImage: id=2cccbea0-cea9-4f86-a3ed-065c652adda5, name=ubuntu-
14.04, driver=OpenStack  ...>

<NodeImage: id=f2a8dadc-7c7b-498f-996a-b5272c715e55, name=cirros-
0.3.3-x86_64, driver=OpenStack  ...>

Our fractal applications can run on any type of distributions such as
Ubuntu, fedora, OpenSUSE, and Debians. These distributions are available
as ready-made operating systems to create virtual machines for our
application.

<OpenStackNodeSize: id=1, name=m1.tiny, ram=512, disk=1,
bandwidth=None, price=0.0, driver=OpenStack, vcpus=1,  ...>

<OpenStackNodeSize: id=2, name=m1.small, ram=2048, disk=20,
bandwidth=None, price=0.0, driver=OpenStack, vcpus=1,  ...>

<OpenStackNodeSize: id=3, name=m1.medium, ram=4096, disk=40,
bandwidth=None, price=0.0, driver=OpenStack, vcpus=2,  ...>
```

Launching an instance

A user has to select the required images and flavor to launch an instance on which the fractal application is going to be deployed. After choosing a flavor and an image from the preceding list, the user has to create an instance using an identified flavored image. Then the new instances will appear as follows:

```
<Node: uuid=1242d56cac5bcd4c110c60d57ccdbff086515133, name=testing,

state=RUNNING, public_ips=[], private_ips=[], provider=OpenStack ...>
```

The launched instance has an instance id, name, and an assigned public and private id. This also includes a state that shows the status as running, and other various states.

Destroying an instance

The following line of Python code will destroy an instance. If we do not destroy an instance, it will incur a cost in utilizing the resources from the cloud. After this, if we try to list the instance, it will disappear from the list of running instances.

```
conn.destroy_node (testing_instance)
```

Deploying the application on a new instance

To deploy an application on the running instances, it may require some additional instances. The first resource is a key pair. By default, the key pair is installed on the new instance. We have a public and private key. The public key is available in the `.ssh/id_rsa.pub` location. Key pair generation is done as follows:

```
print('Checking for existing SSH key pair...')
keypair_name = 'demokey'
pub_key_file = '~/.ssh/id_rsa.pub'
```

The network access is also an important resource for an instance to launch the application. The OpenStack cloud will filter all the traffic that occurs in the network. For this, we need to create a security group and assign it to an instance. Thus, the instance has HTTP and SSH access:

```
if security_group_exists:
    print('Security Group ' + all_in_one_security_group.name + '
      already exists. Skipping creation.')
else:
    all_in_one_security_group =
      conn.ex_create_security_group(security_group_name, 'network
        access for all-in-one application.')
    conn.ex_create_security_group_rule(all_in_one_security_group,
      'TCP', 80, 80)
    conn.ex_create_security_group_rule(all_in_one_security_group,
      'TCP', 22, 22)
```

The user data is also a resource that has to be preinstalled in the image of an instance. Through this, the OpenStack configures the instance to boot with the specified image:

```
userdata = '''#!/usr/bin/env bash
curl -L -s https://git.OpenStack.org/cgit/stackforge/faafo/plain/contrib/
install.sh | bash -s -- \
    -i faafo -i messaging -r api -r worker -r demo
'''
```

Booting and configuring an instance

After configuring with the previously mentioned settings, the instance is ready to be launched on the OpenStack infrastructure. Then the user has to wait for a few seconds to build an instance on which our fractal application will run.

```
if instance_exists:
    print('Instance ' + testing_instance.name + ' already exists.
      Skipping creation.')
else:
    testing_instance = conn.create_node(name=instance_name,
      image=image,
                                        size=flavor,
ex_keyname=keypair_name,
                                        ex_userdata=userdata,
                                       ex_security_groups=[all_in_one_
security_group])
    conn.wait_until_running([testing_instance])
```

Associating a floating IP for external connectivity

The floating point IP is an internet routable IP address for an instance. This id is used for an instance to be reachable from the Internet. Usually, the OpenStack instances have an outbound network access.

```
print('Checking for unused Floating IP...')
unused_floating_ip = None
for floating_ip in conn.ex_list_floating_ips():
    if floating_ip.node_id:
        unused_floating_ip = floating_ip
        break
```

The `conn.ex_list_floating_ips()` function is used to select an IP address from a pool of IP addresses. This address is then assigned to an instance, so the preceding code will return the floating point IP address as follows:

```
<OpenStack_1_1_FloatingIpAddress: id=4536ed1e-4374-4d7f-b02c-
c3be2cb09b67, ip_addr=203.0.113.101,
pool=<OpenStack_1_1_FloatingIpPool: name=floating001>,
driver=<libcloud.compute.drivers.OpenStack.OpenStack_1_1_NodeDriver
object at 0x1310b50>>
```

Then we attach this IP to an instance as follows:

```
if len(testing_instance.public_ips) > 0:
    print('Instance ' + testing_instance.name + ' already has a
      public ip. Skipping attachment.')
else:
    conn.ex_attach_floating_ip_to_node(testing_instance,
      unused_floating_ip)
```

Accessing the application

The following code will start the fractal application deployment. So we can view the running application on the web browser.

```
print('The Fractals app will be deployed to http://%s' %
unused_floating_ip.ip_address)
```

If we need to log in to the instance via `ssh`, we have to get the key pair as follows:

```
$ ssh -i ~/.ssh/id_rsa USERNAME@IP_WORKER_1
```

Summary

In this chapter, we learned the basic features of application design for high availability such as micro services and scalability, fault tolerance, cloud automation, and the RESTful **application programming interface** (**APIs**).We have also understood the various components of application architecture and the detailed interactions between these components in OpenStack.

In the next chapter, we are going to learn about basic and advanced monitoring of the infrastructure components of OpenStack.

10
Monitoring for High Availability

The control and maintenance of a cloud is of the outmost importance; visibility in operations and alerts on failures are the basis of a correct functioning and quick recovery in case of unexpected outages or planned maintenance windows. This chapter will introduce few key concepts and tools to correctly measure and control the operations of an OpenStack cloud.

In this chapter, we will specifically concentrate on open source cloud monitoring services such as Nagios, Graphite, and Elasticsearch/Logstash/Kibana.

In this chapter, you are going to learn about the following topics:

- The Nagios monitoring service
- The Graphite monitoring tool
- Logstash, Elasticsearch, and Kibana

The Nagios monitoring service

The open source Nagios project is used to monitor the complete cloud infrastructure to make sure that the applications, the services running on the systems, and the processes and functions of the business are working properly. Nagios can alert the person who has been involved in the monitoring process of infrastructure through email or any other special intimation service, then the concern staff can start their remediation process. Therefore, outages will not occur in the business process and their customers. All the unseen outages, sudden changes in the service utilization of customers, and other service-related issues on the infrastructure can be easily monitored and reported.

For OpenStack infrastructure-related services, this monitoring system will be used to offer alert and monitoring for the OpenStack network and OpenStack infrastructure.

Installation of the Nagios monitoring service

This open source system will alert or email the administrator by monitoring the hosts' machines and the services running on the machines when an issue arises and will even help to resolve these issues.

Installation of Nagios related packages

As an initial step of installation, we have to install packages such as `nagios-plugins`, `php`, `gd`, `gd level`, `gcc`, `glibc`, `glibc-common`, and `openssl` using the following command:

```
#sudo install nagios nagios-devel nagios-plugins* gd gd-devel php gcc
glibc glibc-common openssl
```

Installation of the Nagios remote plugin executor

Nagios remote plugin Executor (**NRPE**) scripts are used to check the status of the services running on the host, and they help to send reports back to the Nagios system. This Executor is added as an add-on to all the remote nodes available in the cloud infrastructure. To get information about all the nodes, we have to do this for all the nodes. Therefore, this plugin must be installed on all the available remote nodes.

We need to install this plugin as a root user on remote machines. To do this, we need to execute a command as follows:

```
#sudo install -y nrpe nagios-plugins* openssl
```

Then we can view all the plugins under the `/usr/lib64/nagios/plugins` directory of the machine.

Configuring Nagios

We have to do some minimal configuration to set up this monitor system. All the information related to the remote machines and local machines must be sent to the server. It has a web interface that allows you to view the status of related machines together. To do this, we need to do the following:

- The user name and password of the web interface will check and do some basic configuration according to the environment
- We need to add a service of monitoring using OpenStack to the local server

- Finally, we need to inform the Nagios server to identify the host under monitoring and the service of these hosts will be monitored in the near future
- On all the remote machines, we have to install and configure the Nagios NRPE of the following distributed hosts

The following files describe the important configuration files for the Nagios system on the OpenStack cloud environment:

- `/etc/nagios/nagios.cfg`: This is Nagios's most important file, and the main `config` file.
- `/etc/nagios/cgi.cfg`: This is the configuration file for the common gateway interface.
- `/etc/httpd/conf.d/nagios.conf`: This is the configuration file for `httpd`.
- `/etc/nagios/passwd`: A Nagios user can get their password from this `config` file.
- `/usr/local/nagios/etc/ResourceName.cfg`: All the user related settings are done on this file.
- `/etc/nagios/objects/ObjectsDir/ObjectsFile.cfg`: All the groups and service related information are stored in the object file, in which all the definition of the users, services, and groups are collated.
- `/etc/nagios/nrpe.cfg`: All the remote machines' NPRE's are available in this `config` file.

HTTPD configuration

The predefined user name and password of the Nagios system are `nagiosadmin` when we initially install the system. We can get this information from a file under the `/etc/nagios/cgi.cfg` directory.

We need to configure this Nagios system in our environment as a root user using the following command. In this, we can change the predefined password of the default user to `nagiosadmin`:

```
#sudo htpasswd -c /etc/nagios/passwd nagiosadmin
```

If we need to create a new user for Nagios, then we will execute the following command:

```
# sudo htpasswd /etc/nagios/passwd newUserName
```

We have to edit a file called `contacts.cfg` under the `/etc/nagios/objects` directory to change the email address of the user called newUserName (in our case) using the following command:

```
define contact{
        contact_name    nagiosadmin             ; Short name of user
        [...snip...]
        email           yourName@example.com    ;
        }
```

After doing the preceding changes, we have to check Nagios for the basic configuration as follows. Then if we get any errors with the basic configuration, we have to check the parameters available in the `nagios.cfg` file:

```
#sudo nagios -v /etc/nagios/nagios.cfg
```

Whenever the system starts, the infrastructure monitoring process also begins. Therefore, we need to check that Nagios starts automatically. To check for the auto start, run the following command:

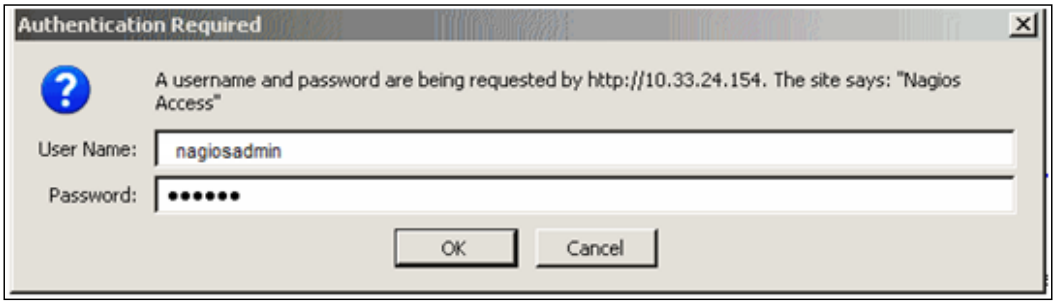
```
#sudo chkconfig --add nagios
#sudo chkconfig nagios on
```

Finally, we have to restart the Nagios system and change the `httpd` service using the following command:

```
#sudoservice httpd restart
#sudo service nagios start
```

Accessing the Nagios web interface

We can access the Nagios web interface for login with the created username and password:



OpenStack services configuration

The OpenStack services that require a monitoring service are added to the file called `commands.cfg` available under `/etc/nagios/objects`. This list consists of different services of OpenStack to be monitored. Therefore, we have to write a code of script that is used to add the services available to the list as follows. The scripts are added to the file under the `/usr/lib64/nagios/plugins` directory. The following script of code is used to check this:

```
#!/bin/env bash
export OS_USERNAME=userName
export OS_TENANT_NAME=tenantName
export OS_PASSWORD=password
export OS_AUTH_URL=http://identityURL:35357/v2.0/
data=$(nova list  2>&1)
rv=$?
if [ "$rv" != "0" ] ; then
    echo $data
    exit $rv
fi
echo "$data" | grep -v -e '--------' -e '| Status |' -e '^$' | wc -l
```

Then in the `commands.cfg` file under the `/etc/nagios/objects/` directory, each script is specified using the following command:

```
define command {
command_line                /usr/lib64/nagios/plugins/nova-list
command_name            nova-list
}
```

Using the define command, all the required services are defined as follows by editing the `localhost.cfg` file under `/etc/nagios/objects/`:

```
define service {
        check_command   nova-list
        host_name       localURL
        name            nova-list
        normal_check_interval   5
        service_description     Number of nova vm instances
        use             generic-service
        }
```

Finally, the services are restarted using this command:

```
#sudo service nagios restart
```

OpenStack services configuration

We set up an NRPE on each remote node, then execute the following command as a root user, then add the Nagios server IP address to the `allowed_host` attribute. Also, add any services for monitoring purposes under a directory called `/etc/nagios/nrpe.cfg`:

```
allowed_hosts=127.0.0.1, NagiosServerIP
command[keystone]=/usr/lib64/nagios/plugins/check_procs -c 1: -w 3: -C keystone-all
```

Then, open the NRPE ports as follows:

```
# sudo iptables -I INPUT -p tcp --dport 5666 -j ACCEPT
# sudo iptables-save > /etc/sysconfig/iptables
```

Finally, restart the NRPE services after all the preceding changes are done properly:

```
# sudo service nrpe start
```

Service definition creation

All the services must be added to the service list for monitoring purposes. Only after this, the Nagios system will automatically monitor the services listed in the file when the system starts. For this, we have to create a file called `services.cfg` under a `/etc/nagios/objects` directory:

```
##Basic remote checks##############
define service{
  use generic-service
  host_name remoteHostName
  service_description PING
  check_command check_ping!100.0,20%!500.0,60%
}
define service{
  use generic-service
  host_name remoteHostName
  service_description Load Average
  check_command check_nrpe!check_load
}
define service{
  use generic-service
  host_name remoteHostName
  service_description Identity Service
  check_command check_nrpe!keystone
}
```

The preceding file consists of the status of the identity service, and it checks for the load and server's starting heartbeat. All these information are reported to the Nagios server.

After the preceding step, verification must be done to ensure that the changes are made to the files.

For this, we need to execute the following command:

```
# sudo nagios -v /etc/nagios/nagios.cfg
```

As a last step of this Nagios installation, we have to restart the Nagios service as follows and then access the Nagios server web interface:

```
# sudo service nagios restart
```

Graphite monitoring tool

The Graphite tool is an open source tool, which is used to store the time series data from various sources and use them to generate a graphical representation. So all the service-related activities are monitored through the graphs generated by this tool. This tool is used to identify resource utilization of the cloud. It comprises of three main components: Carbon, Whisper, and Graphite web app.

In this section, we will try to push the metrics of OpenStack ceilometers into the graphite tool. Due to some of the issues in ceilometer, we will not reach the performance of cloud monitoring. So the data from the compute node will be directly sent to the backend of the Graphite tool to avoid network bottlenecks.

Installing Graphite

We already know the procedure to install the ceilometer service on the OpenStack infrastructure. For this, we need to install the ceilometer OpenStack package in the system.

Ceilometer configuration

The Ceilometer configuration files need the following changes:

- The RabbitMQ settings need to be changed
- The Keystone service settings need to changed

- We have to configure the Graphite settings as mentioned in the following file:

```
rabbit_host=10.10.10.10
rabbit_userid=OpenStack
rabbit_password=bla
rabbit_virtual_host=/
os_username=ceilometer
os_password=pass
os_tenant_name=services
os_auth_url=https://keystone.bla.com/v2.0
os_region_name=region1
[graphite]
prefix = stats.whateverkeyyoulike.endwithdot.
append_hostname = true #This will add the hypervisorname to the
prefix
```

Adding publisher

The Graphite end-points need to be added to the publishers by editing the file under a directory called:

```
#sudo nano /usr/lib/python2.6/site-packages/ceilometer-2014.1.1-
py2.6.egg-info/entry_points.tx
```

The file content needs a few changes as follows:

```
append_hostname = true [ceilometer.publisher]
graphite = ceilometer.publisher.graphite:GraphitePublisher
```

After adding the publishers, we have to restart the ceilometer agents available in the OpenStack infrastructure.

Carbon installation

Carbon is one of the components of Graphite, and is responsible for receiving metrics over the network and writing them to disk using a storage backend.

The initial step of installing a carbon is as follows, after which we need to adjust the default configuration of the carbon according to our requirement and environment:

```
#sudo apt-get install -y graphite-carbon
```

Then we have to enable the carbon cache option available in the Graphite —carbon file under `/etc/default` using the following:

```
# echo "CARBON_CACHE_ENABLE=true" > graphite-carbon
```

After enabling the value to be true, we have to restart the Graphite cache service using the following:

```
#sudo service carbon-cache restart
```

We can access the graphite through the same IP address as we did for Nagios:

Logstash, Elasticsearch and Kibana

These are open source technologies for the efficient storage, analysis and retrieval of any type of log files. The backend storage will be provided using elasticsearch, and Kibana is used to generate reports via various chats instantly. In this OpenStack cloud environment, all the logs files that are generated will be considered for analyzing some resource utilization according to the policies. There is a log of built-in functions available for various purposes with the analyzed log files.

Installing Logstash

To install `logstash`, we need a Java runtime. We can check with the system whether Java will be available using this command as follows:

```
#sudo java –version
```

As an initial step of the Logstash installation, we have to download the binary as follows. Before this, we need to configure `logstash` and run it on the terminal:

```
# sudo curl -O https://download.elasticsearch.org/logstash/logstash/
logstash-1.4.1.tar.gz
```

Then find `logstash-1.4.1` under a `tar` file and run it as follows:

```
#sudo bin/logstash –e 'input { stdin {}} output { stdout {}}'
```

Next, we have to type a few lines (unstructured format) similar to log file contents as follows:

```
Hello world
2015-08-21T01: 22:14.405+0000 0.0.0.0 hello world
Stdin – input for the logstash file
Stdout- outpur for the logstash file
```

With the help of, this `logstash` file can directly get the configuration details from the command line. Therefore, there is no need to edit the file in every iteration and testing can be easily done.

An Elasticsearch store

It is very tedious that every time the log file contents have to be typed using `STDOUT`. So instead of this, Elasticsearch is used to act as a backup store to store all the messages that are send to `logstash`.

As an initial step of the Elasticsearch installation, we need to download the Elasticsearch `tar` file as follows:

```
# sudo curl -O https://download.elasticsearch.org/elasticsearch/
elasticsearch/elasticsearch-1.4.2.tar.gz
```

Then un-tar the file using the `tar xzvf` command and run the following code:

```
# sudo bin/logstash -e 'input { stdin { } } output { elasticsearch { host
=> localhost } }'
```

By default, `logstash` and `elasticsearch` are well suited to run the commands. Hence, we can avoid the other configurations in the `elasticsearch` backend process.

The Kibana frontend

With the analyzed data availability, an open source Kibana will handle the visualization of the required results from the log files stored in the Elasticsearch. We can plot our own tables and charts, and do an advanced level of data analytics using this tool. We will show our own results through the previously mentioned mediums in a frontend dashboard.

Summary

In this chapter, we learned about basic and advanced monitoring of the infrastructure components of an OpenStack cloud with a step-by-step configuration of open source tools such as Nagios and Graphite, and were given an overview of Logstash, Elasticsearch, and Kibana.

In the next chapter, we will learn about the use cases of HA deployments with real-world examples and learn some best practices of OpenStack.

11
Use Cases and Real-World Examples

In this last chapter, we are going to learn from the different case studies of a variety of industries that are reaping the benefits of the high availability of OpenStack. After going through all these case studies, we will get a clear picture in our mind about how the infrastructure challenges of various industries are addressed by OpenStack. In this way, we will attain the approach to implement a high availability of OpenStack in our organization.

The following are the categories of case studies covered in this chapter:

- An Oracle, CISCO, Yahoo, and HP Helion Cloud case study of Cisco Webex
- A case study of Huawei
- A case study of Multiscale Health Networks
- A case study of eBay

A case study of Cisco WebEx

Cisco WebEx is a very critical business service and never goes down. This is one of the most popular web conferencing services adopted by many organizations across the globe.

Challenges with the infrastructure of Cisco WebEx

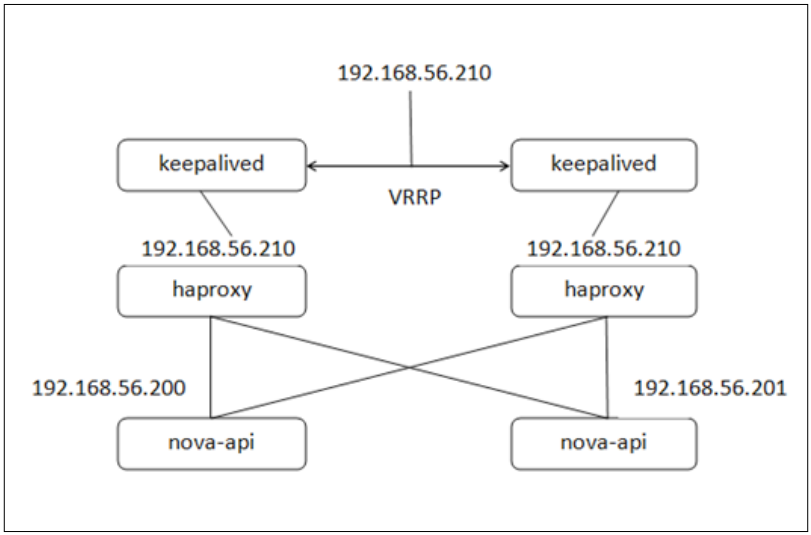
Being a pioneer in software as a service, Cisco WebEx is required to continuously innovate the IT operations infrastructure. The following are some critical challenges of Cisco WebEx :

- Due to its worldwide accessibility, the underlying infrastructure must be highly available and robust
- WebEx services are highly critical and very confidential for every client such as screen sharing, voice and video sharing, and other collaborative services that cannot easily go in the public cloud
- There are many key supporting services such as recording stage, analytics, and system management that require a cloud-based agility solution and cost saving also.

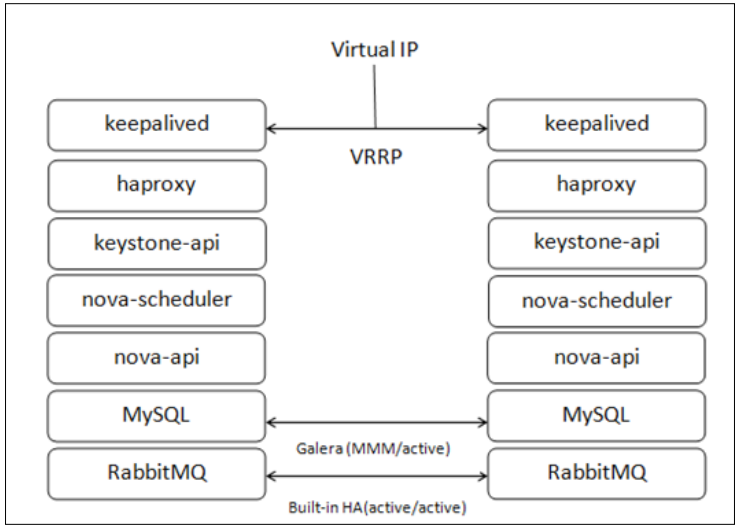
The solution with OpenStack

Cisco recognized the potential of OpenStack, and OpenStack decided to go with it for commercial and overall operational benefits. With the help of OpenStack high availability, all the cloud services are available for all the requests coming from critical applications, which run on the cloud for the WebEx services. Due to a need for a large number of resources and uninterrupted services for the users of WebEx, OpenStack high availability ensures that all the resource requirements are met at every point of the services.

So WebEx users are expected to replace dead virtual machines with the help of relaunched hypervisors immediately on the NOVA computing nodes using this HA facility of OpenStack. Cinder volume protects from data loss if there is any critical loss of storage and ensures that block volume is rarely lost by the service providers, as shown in the following figure:



In the previous figure, we can see the Keepalived nodes and how the VRRP is utilized for HAProxy services on the nodes in real-time use cases. The nova-api is used to do customization among the HAProxy nodes according to requirements.



Then the high availability and load balancing capabilities are provided through pacemaker cluster communications done on these Cisco WebEx services. In addition, a configuration of Distributed Replicated Block Device for the RabbitMQ service is used to get the data from RabbitMQ as shown in the preceding diagram. VRRP will aid movement among the cluster nodes in the HA OpenStack infrastructure.

The final outcome

Following are the final outcome after implementing high availability OpenStack solution :

- Lowering the overall infrastructure's recurring costs and improving roadmap alignment between infrastructure and software layers
- In the event of failures, such as memory out of bounds or failures with the software, and panics in the operating system of WebEx, keepalived and active—the active mode using VIP—are used to remove the failures
- A single point of failure is greatly avoided in the case of any system downtime and if data loss occurs within the WebEx-running servers when data moves between a user and WebEx
- There is a great reduction in the overall cost spent towards the recovery of failures

Case study of Huawei

Huawei is a giant that provides cloud computing-based solutions, and they have in-house development environments such as virtual desktop infrastructure, which is used by more than 45,000 internal employees of Huawei. With this kind of infrastructure, Huawei could recognize the promising benefits of OpenStack for internal infrastructure as well as for client-based applications.

Challenges with the infrastructure of Huawei

Based on the cloud requirements of Huawei corporations, the deployment architecture of the cloud infrastructure will have some serious challenges, as follows, and will have some significant hardware needs for the infrastructure:

- Due to the large number of users, there is a need for the high availability of a service for all the users at all the times
- Since Huawei has isolated network components, there is a need for security using firewalls between internal and external users
- To manage the image-specific use cases of Huawei on Ubuntu machine images and a requirement for high-end Ethernet cards to provide a high speed internet facility

The solution with OpenStack

With the help of OpenStack high availability deployment, the storage component called **Swift** will ensure simplified storage for all storage requirements of Huawei, such as design files, media files, and other logs.

To ensure the security access and storage access of the all the previously mentioned files between internal and external users of Huawei, the HA OpenStack provides the Swift storage component integrated with the keystone security service as a solution. To avoid the previously mentioned challenges, the keystone lists all the authorized users of the swift storage. Using this technique, the token for each client is used to identify each permissible user as well.

The final outcome

Following are the final outcome after implementing high availability OpenStack solution :

- The availability of resources from the OpenStack cloud for all the users of Huawei is satisfied with the high availability architecture of OpenStack with a minimal cost of operations and maintenance since it is an open source technology
- Keystone token facility will ensure the correctness of data available for both internal and externals users with Swift storage
- A huge demand for all Ubuntu images is provided with the Swift object store capability by a highly available MySQL—MySQLMaster Master that has the ability of failover and scalability

Case study of Multiscale Health Networks

Multiscale Health Networks is one of the most popular companies of the health and services industry, which targets the specific requirements of life sciences and health care using high performance computing, cloud computing, and virtualization-based solutions. Multiscale provides services in five western states (US), and has more than 65,000 employees.

Challenges with the infrastructure of Multiscale

The sudden increase in the data storage needs of healthcare industries made them move to the cloud to reduce the computing cost of the customers for any healthcare company. So for any requirements of storage, the healthcare industry has moved towards to the cloud infrastructure for its optimization. The following are the major challenges faced by any health care organization:

- Need to enable multitenancy with full isolation of any type of hypervisors
- Access control mechanisms have to be provided to the heterogeneous storage service for the applications
- Also, they must ensure data correctness via access control for a high level of encryption and should have the capability for high recoverability when there is data loss

The solution with OpenStack

Since Multiscale wants on-premises data entry and remote deployment, therefore after evaluating multiple open source cloud provider options, Multiscale decided to go with OpenStack.

The final outcome

Following are the final outcome after implementing high availability OpenStack solution :

- A high performance computing facility is enabled via a Torque script. Hence, any number of nodes in the cluster can be added or deleted at any time
- To automate the bare metal nodes provisions, the VLAN can cause any one of the projects to plug into the instances of the bare-metal hypervisor

Case study of eBay

Today eBay's on-premise cloud offers various features such as multitenant, self-service, and multiregion. These features have come up on the cloud where all the critical applications, platforms for the development of applications. Therefore, the company needs to build a cloud that has agility, scalability, and robustness.

Challenges with eBay business process

In recent years, the e-commerce technology has revolutionized the online market business in a crucial way that essential workloads are processed in their own data center. This has happened due to the following:

- ebay to avoid supplier lock-in using this infrastructure
- eBay's critical applications are yielding the benefit of high availability and failover for efficient business to compete with other market vendors

The solution with OpenStack

All online marketing businesses have jumped into this data centre environment to deal with very high and decisive workloads. These e-commerce websites are running with millions of users and purchases during every day of their business. To overcome this situation, online marketing businesses have started to adopt cloud technology. Industries are fully operating their heavy computations on their on-premise cloud. Today, enormous market traffic is powered by cloud computing, that is, the OpenStack cloud.

The final outcome

Following are the final outcome after implementing high availability OpenStack solution:

- Transparency of services is the most important feature of any industry to secure their assets and data when their applications are deployed in the cloud
- With an on-premise infrastructure, security is ensured for the client as they run their business in the cloud
- Asset transparency makes this feature secure and useful when a client wants to deploy the app in the cloud

Summary

In this chapter, we saw a variety of the case studies of different industries that includes Cisco, Huawei, Multiscale Health Networks, and eBay.

As we have learned, all types of industry have some critical infrastructure challenges that can be easily overcome with OpenStack high availability-based solutions.

Index

D

E

G

H

I

K

L

M

N

O

P

Q

R

S

V