Master’s Degree programme – Second Cycle (D.M. 270/2004) in Informatica — Computer Science

Final Thesis

Dynamic Load Balancing With Openstack Cloud

Supervisor
Prof. Simonetta Balsamo

Co Supervisor
Prof. Andrea Marin

Candidate
Momodou Njie

Matriculation Number: 854938

Academic Year
2017 / 2018
Abstract

Dynamic load balancing with Openstack Cloud

by

The goal of the thesis is to present Openstack and the implementation of its load balancing services for cloud datacenter. In this work, we will introduce the various algorithms developed in the literature and in this light, we also study the implementation mechanism proposed by Openstack. Openstack is a cloud computing based Operating system software which controls large pools of computing, storage, and networking resources managed through a dashboard or via the Openstack API. . .
Acknowledgements

First of all, I would like to thank the Government of the Gambia for availing me the opportunity to do a master’s degree. Also wish to express my sincere appreciation to my professors, Prof. Simonetta Balsamo and Prof. Andrea Marin. There is no greater honour than working with you both. I am grateful for your guidance and every moment shared working towards a successful completion of my thesis. Thank you for your endless support and supervision, and for the push that got me to believe in myself even more. You kept me going when I didn’t believe I would find a way on a road-less journey. Despite all challenges, you helped me to achieve the right result, opening the door for my future career.

Thanks, the Yeoman’s and Salinas family, Mom Kathy, Gary, Tiana for welcoming me into your lovely home. Thanks for the beautiful experiences, introducing me to rafting and so many other wonderful activities. Those memories I will forever treasure. To my beautiful family, my Mum Oumie, Aunty Adama, Neneh, Ndey, Papa, Yusupha, Muhammed, Aminata, Ousman, Khadijatou, Dawda Sally Jatta and my newly born baby girl Oumie Njie, I am grateful for your prayers, encouragement and the endless love that I will forever live to appreciate. Finally, to my beautiful wife, Absa Baye Njie, my love, the shining star that brings light and brightens my days and nights with beautiful smiles and encouragement. You are always there for me in every step on the way and we have a lifetime more to go. Your love makes it all complete thank you....
Contents

Abstract i
Acknowledgements ii
1 Introduction 1
2 Review on Cloud Computing 3
  2.1 Cloud Services ........................................ 3
    2.1.1 Software-as-a-Service (SaaS) ..................... 3
    2.1.2 Platform-as-a-Service (PaaS) .................... 4
    2.1.3 Infrastructure-as-a-Service (IaaS) ............. 5
  2.2 Cloud architectures ................................... 5
  2.3 Cloud Deployment Models ............................ 7
    2.3.1 Public Cloud ...................................... 7
    2.3.2 Community Clouds .................................. 9
    2.3.3 Private Clouds ................................... 10
    2.3.4 Hybrid Clouds ................................... 11
  2.4 Commercial Cloud Solutions ........................ 12
3 Load balancing in the cloud 14
  3.1 Needs for Load Balancing ............................. 16
    3.1.1 Performance metrics to optimize ................. 17
  3.2 Static load balancing algorithms .................... 17
  3.3 Dynamic Load Balancing Algorithms ................. 18
    3.3.1 Sender initiated migration policy vs Receiver initiated migration policy .................. 21
    3.3.2 Global vs. Local Strategies ..................... 22
    3.3.3 Centralized vs. Distributed Strategies ........ 22
4 Openstack Documentation 23
  4.1 History of Openstack ................................. 23
  4.2 Openstack Architecture ............................... 24
  4.3 Openstack Components ................................. 25
  4.4 The basic components ................................. 25
  4.5 Openstack Networking Components .................. 35
    4.5.1 Standard networking components ................. 36
    4.5.2 Network Traffic Load Balancer .................. 39
Chapter 1

Introduction

In about ten years ago, not so much could have been predicted that the cloud computing would become this big in such a short period of time. Today, with the presence of the cloud the nature of computing and how business operates has evolved tremendously. The adoption of cloud computing solution in this era of big data and IoT (Internet of Things) in enterprises is rising significantly and debatably every organization is either planning or has already moved to the cloud. Cloud computing attempts to realize the vision of utility computing, through the provisioning of virtualized hardware, software platforms and software applications using services over the Internet. In another word, cloud computing is a model for provisioning resources dynamically and elastically. Even without realizing, in most of our day to day computing activities we use cloud services such as email services, Google documentation etc.

Cloud computing may be referred to many different forms of technologies and services and its concepts are often associated with virtualized infrastructure or hardware on demand, utility computing, IT outsourcing and many more. The Internet act as the medium and plays a fundamental role in service delivery aspect of cloud computing. Even though there are many definitions of cloud computing, in this thesis, we adopted the proposed definition by the U.S. National Institute of Standards and Technology (NIST): **Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. Such resources are a range of servers, network storage and applications that can be provisioned without an interaction of a service provider.** The NIST cloud definition gives three main components that will be covered in brief.

Infrastructure-as-a-Service (IaaS) such as Amazon Web Services demoted (AWS) offers cloud services the ability to acquire resources on-demand, usually in the form of virtual machines (VMs). Platform-as-a-service (PaaS) clouds offer platform as services such as application development, testing, and run-time environments. Lastly, Software-as-a-Service (SaaS) clouds deliver specialized software as web-based services.

Infrastructure, Platform, and Software as a service herein this article referred as I.P.S together with load balancing plays a fundamental role in the needs
for service delivery in the cloud and in the management of data centers where a fair workload distribution may be hard to achieve. The benefits of using cloud computing varied, and amongst many, includes a cloud’s flexibility and resiliency, the ability for reducing costs, the availability of very large amounts of centralized data storage, rapid deployment of computing resources and scalability.

The goal of the thesis is to present Openstack and the implementation of its load balancing services for cloud data centers and how the optimal performance of such environments can be achieved with Openstack. We will introduce the various algorithms developed in the literature and in this light, we study the implementation proposed by Openstack.
Chapter 2

Review on Cloud Computing

Adopting the NIST (National Institute of Standard Technology) definition’s, cloud computing outlines five essential key characteristics and thus as follows: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service also presenting the three service models of the cloud (IaaS) as Infrastructure as a Service, (PaaS) Platform as a Service, and (SaaS) Software as a Service coupled with four deployment models Public, Private, Community, and Hybrid cloud.

2.1 Cloud Services

The diagram below presents how cloud computing services are classified into the three layers that form its cloud ecosystem

![Cloud Service Models](image)

**Figure 2.1:** Cloud Computing Services

The Application layer forms the basis for Software as a Service (SaaS), whilst the Platforms forms the basis for Platform as a Service (PaaS) models and lastly, Infrastructure as a Service (IaaS) which creates what may be determined to be the utility computing model.

2.1.1 Software-as-a-Service (SaaS)

The Software as a Service denoted as (SaaS), is the popularly regarded as the “software-on-demand” and is usually priced on a pay-per-use basis or most
a common term a payment plan with a subscription fee. SaaS represents a range of hosted applications that are available over the Internet through a web browser. With the SaaS model, the user interacts directly with the hosted software, vendor or service providers manage the infrastructure and platforms that run the applications. Giving the cloud service providers, the sole power and responsibilities to install and operate application software in the cloud whilst cloud users can gain access to the application software databases from cloud clients platform. Software as a Service provides the complete infrastructure, software and solution stack offering. Examples of SaaS applications are Microsoft 365, Google Gmials services, QuickBooks etc.

Software-as-a-Service (SaaS) applications share the following characteristics:

- Centralized hosting which provides software applications to users through different distribution channels.
- Uniform platform allowing the users with different operating systems and operating platforms function on a single platform the (Internet) with a uniform type of client interface a (web browser).
- Open collaboration allows consumers with the necessary tools to control how and when they share jobs, data and information with other users either internal or external to their organization.
- The Software-as-a-Service software’s is made available over the Internet accessible through a browser on demand.
- The typical license as a subscription-based or usage-based and is billed on a recurring basis.
- Reduces distribution and maintenance of costs generally make SaaS applications cheaper to use than their shrink-wrapped versions.

2.1.2 Platform-as-a-Service (PaaS)

Platform as a Service (PaaS) provides a cloud computing platform through remote utilization of an application and creates a development environment upon which applications may be built. PaaS providing a complete solution stack for cloud developers, provisioning the operating system, programming language execution environment, database and much more to build the application needed. One of the primary benefits of PaaS is having software development and deployment capability entirely in the cloud hence, no management or maintenance efforts are required for the infrastructure exposing source-code management, testing, and deployment available in the cloud.

Platform-as-a-Service (PaaS) solutions have the following characteristics in common:

- PaaS must offer some type of development language so professional developers in some cases users can add value.
• Almost all PaaS platforms are based on a multi-tenancy architecture which allows multiple clients to run their own copy separately from each other through virtualization making sure each customer data is isolated from others.

• A PaaS environment needs to support the cloud application development life-cycle and development process including testing.

• A PaaS platform must be able to deploy, manage, test, and maintain the developed applications in its infrastructure.

2.1.3 Infrastructure-as-a-Service (IaaS)

Infrastructure as a Service is widely denoted as IaaS. Is the fundamental building blocks for cloud services enabling cloud providers the sole responsibilities for maintaining and managing all the infrastructure in the cloud. The consumers on the hand, control a certain aspect of the services including the operating systems, storage and other deployed applications within an infrastructure. The Amazon Web Service and Google Cloud Platform as (GCP) are good examples and description of IaaS providers.

Infrastructure as a Service allows consumers to have direct access and manage their servers and storage from anywhere and thus helps an organization to grow or shrink their business infrastructure. Cloud’s agility, availability, mobility, stability, and elasticity in business are made possible in cloud computing while incorporating of IaaS. Infrastructure as a Service is the most flexible cloud computing model and allows for automated deployment of servers, processing power, storage, and networking.

Infrastructure as a Service applications share the following characteristics:

• Cloud resources are provided as a service

• It allows for dynamic scaling and elasticity

• Usage-based pricing model (pay per go and pay per use)

• Multi-tenant architecture and includes multiple users on a single piece of hardware

• Infrastructure as a Service typically has enterprise-grade infrastructure

2.2 Cloud architectures

Cloud architecture can couple software running on virtualized hardware in multiple locations to provide an on-demand service to users. This unique combination of abstraction and metered service that separates the architectural requirements of cloud computing systems from the general description given for an entire Internet application.
Cloud architecture details outline the components and subcomponents and can be classified into two layers:

- A thin client as the front end
- The cloud as the back end
- In other works of literature, a third layer is regarded as a cloud-based deliver such as the internet, intranet.

A cloud can be created within an organization’s own infrastructure or outsourced to another datacenter. While resources in a cloud can be real physical resources, more often they are virtualized resources because virtualized resources are easier to modify and optimize. Many service providers claim to offer the best cloud services, even though in reality they really do not. Just the presence of a Web-based application does not mean that it is fully qualified to be classified as a cloud base service. The application and the service must exhibit certain characteristics before it can be considered a true cloud base. According to the National Institute of Standard Technology (NIST), an application or service can be regarded as a cloud computing outlines five key cloud characteristics: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. All five of these characteristics must be present for the offering to be considered a true cloud offering.

Key Cloud Characteristics

**On-Demand Self-Service**  On-demand self-service means that a cloud user can request or receive access to new and existing service offering without the intervention of an administrator. The request and service delivery processes are all fully automated. This offers advantages to both the provider and the consumer of the service in getting the resources needed very easily and within a short time frame. The implementation of on-demand self-service allows customers to quickly procure and access the services they want. The self-service capability allows the administrators to focus less on collecting customer requirements and more on planning and designing new capabilities. This makes the on-demand self-service a very attractive feature of the cloud.

**Broad Network Access**  Services are available over the network and accessed through standard mechanisms (e.g. mobile phones, laptops). According to (NIST Report 2011) cloud services should be easily accessed over a network and the services should be able to be accessed by a wide variety of client devices from any location via a simple web-based interface. Users should only be required to have a basic network connection to connect to services or applications. Administrators can also access and provision cloud resources from outside their corporate organization, allowing rapid satisfaction of service requests. This mobility is especially advantageous to consumers who frequently need to access information while mobile or from a telework location.
Resource Pooling  Clients often do not have a constant need for all the resources available to them. Resource pooling, therefore, helps organizations save cost and allows flexibility on the side of the provider. When resources are not being used by a customer, instead of sitting idle those resources can be relayed to another customer who is in need to use those resources. This gives providers the ability to serve as many customers as possible. Resource pooling is often achieved using virtualization and cloud users don’t necessarily know where the services are coming from. Virtualization allows providers to increase the density of their systems. Means, they can host multiple virtual sessions on a single system. In a virtualized environment, the resources on one physical system are placed into a pool that can be used by multiple other virtual systems.

Rapid Elasticity  Rapid elasticity describes the ability of a cloud environment to easily grow to satisfy user demand hence resources can be rapidly and elastically provisioned. Rapid elasticity is usually accomplished with automation and orchestration. When resource usage hits a certain point, a trigger is set off. This trigger automatically begins the process of capacity expansion.

Measured Service  A cloud computing system must have the capabilities to measure, controlled, monitored and report usage of resources. This is necessary to achieve transparency for both the provider and consumer over the utilized service. Usage of measure services can be quantified using various metrics, such as time consumed, bandwidth, logging activities, data transferred etc. The measured service characteristic is the core system that enables the pay-as-you-go features of the cloud.

2.3 Cloud Deployment Models

Every organization has its own requirements as to what services it wants to access from a cloud and how much control it wants to have or give over. To accommodate these varying requirements, a cloud environment can be implemented using different service models and techniques. Each cloud service model has its own sets of requirements and benefits. Following the National Institute of Standard Technology (NIST) definition which identifies the four different cloud deployment models to be: Public cloud, Private cloud, Community cloud, and Hybrid cloud. The cloud deployment model, therefore, defines the purpose of the cloud and the nature of how the cloud is located.

2.3.1 Public Cloud

The public cloud infrastructure is available for the public and refers to the cloud services that are open to the wider community, usually owned by the organization selling cloud services. The public cloud is the realization of the canonical view of cloud computing in which the services offered are made available to anyone, from anywhere, and at any time through the Internet. The public cloud service providers can host a wide variety of services from leasing
In the Public Cloud models, cloud providers take on the responsibilities of installation, management, provisioning and maintenance. The customers access and use the services and physical resources and consumers are charged only for the resources and services they use. It follows a pay-as-you-go approach to determine the usage and this is the main attraction of Cloud environment. An example amongst many and some of the well-known Public Cloud providers and products are Amazon Web Services, Microsoft Azure, Google Apps, SalesForce.com

There are tonnes of advantage and some of the benefits of using the Public cloud includes:

- Public Clouds Utility Model- Typically delivers a pay-as-you-go model. This model allows cloud users to pay for services for a certain time as per the hour for the computing resources used.

- No Contracts- A client only pay as per use and by the hour as used, if a customer wants to shut down a server or stop a service after an hour of usage, there won’t be any contract requiring the ongoing use of the server or that specific service.

- Shared-Hardware- The public cloud by definition is a multi-tenant environment, that is, servers share the same hardware, storage and network resources as the other tenants in the cloud.

- Self Managed; With the pay-as-you-go utility model, self-managed systems are required for this business model to excel and deliver as promised.

- Reliability- The high number of servers and networks involved in creating a public cloud and the redundancy configurations means that should one physical component fail, the cloud service would still run unaffected on the remaining components

- Location- The high availability of public cloud services through an internet ensures that services are available wherever ignoring the client place or
location. This feature provides invaluable opportunities for organization and enterprise for the operation of remote service invocation such as online document collaboration from multiple locations.

However, despite the numerous advantages and benefits, there are disadvantages to using the public cloud.

- Performance can be a key issue in the public cloud.
- Data transmission could be affected by spikes in use across the internet.
- The client has no control of data or infrastructure.
- There are issues of data privacy and integrity.

2.3.2 Community Clouds

![Community Cloud](image_url)

**Figure 2.3: Representation of a Community Cloud**

A community cloud is a collaborative effort in which infrastructure is shared between several organizations, a multi-administrative domain involving different deployment models (public, private, and hybrid). Community clouds are distributed systems created by integrating the services of different clouds to address the specific needs. Community clouds can sometimes be public clouds for dedicated resource infrastructure. In such a cloud model, smaller organizations may collaborate together to pool their resources for building private community clouds.

**Community Cloud Advantages**

- Ability to easily share and collaborate
- Lower cost helps providers to save money
- Sufficient flexibility to meet community’s need
- Standardization to prevent lock-in
- Easy Network integration
Community Cloud Disadvantage

- Not the suitable choice for every organization
- Slow adoption

2.3.3 Private Clouds

![Private Cloud](image)

**Figure 2.4: Representation of a Private Cloud**

A private cloud as a cloud infrastructure operated solely for an organization, either managed on-premises or off-premises by the organization or a third party. Obviously, not every resource of an organization can be made available to the public and hence the sheer needs of this cloud model. Private cloud is a proprietary network and is implemented within the private premises of an institution. Generally made accessible only to the members of the organization or subset partners.

Private clouds are virtually distributed systems that rely on a private infrastructure and provide internal users with dynamic provisioning of computing resources. Instead of the usual "Pay-as-you-go" model as in the public clouds, other schemes are put in place, the enterprise is in charge of setting up and maintaining the cloud resources and, thus, the enterprise can take better control of security and regulatory compliance issues. Giving the Private Cloud the advantage of privacy keeping the core business operations in-house by relying on the existing IT infrastructure and reducing the burden of maintenance once the cloud has been set up. Moreover, existing IT resources can be better utilized because the private cloud can provide services to a different range of users.

**Private cloud computing also offers a number of benefits:**

- Security: Since the private clouds are dedicated to a single organization, hardware, storage and network can be designed to assure high levels of security that cannot be accessed by other clients outside the organization.

- Customizable: The hardware, system performance, network performance and storage performance can be specified and customized.
• Controls: With the private cloud there better controls of data, users and information assets.

• Efficiency: Private clouds are hosted either on-site or on in a third-party data center, that is also a privately hosted environment. Thus gives private cloud owner more control over the data and infrastructure.

Challenges that comes with the private cloud

• Underlying cost involved in the implementation especially if it outsourced, the running cost usually include periodic hardware upgrade.

• Resources under-utilization- In some instances, the computing resources can be under-utilized.

• Vendor lock-in- This is very common in private cloud especially when the hardware and infrastructure are outsourced. Vendor Lock-in is a service delivery technique where the client company is forced to continue with the same service provider, thus preventing the client or cloud user to migrate to another vendor.

2.3.4 Hybrid Clouds

A Hybrid cloud as defined in the National Institute of Standard Technology (NIST) is a combination of two or more clouds combined together (private, community, or public) that forms a unique entity and enables data and application portability. A hybrid cloud deployment allows an organization to divide its set of applications based on its sensitivity. Thus, giving enterprises the opportunities to maintain noncritical software applications in its public cloud, whilst keeping critical or sensitive information in the private cloud.

Hybrid Cloud is usually referring to one-size-fits-all solution models and offers such benefits to cloud providers.
• Business can reduce the overall total cost of ownership and improve cost efficiency
• Improved Security is another major benefit of hybrid clouds.
• Implementing a hybrid cloud approach is cost savings
• Hybrid clouds is enhanced organizational agility

Challenges involved in the implementation of a Hybrid Cloud solution.

• As much as the Hybrid cloud model can be cost savings, financial cost plays a major role in planning to execute a hybrid cloud strategy
• Data and application integration is a challenge

2.4 Commercial Cloud Solutions

There are so many underlying benefits of using cloud computing especially for commercial benefits and one of the most outstanding reason is its ability to generate revenue for the organization, promoting its commercial development. Commercial Cloud computing allows cloud owner or providers to set up virtualized offices environment to provide flexibility and connecting to the business anywhere and at any time using web-enabled computing devices.

Below presents a quick summary of some of the significant benefits that surround the commercial cloud.

1. Fresh Software
   In a Software-as-a-service (SaaS) business environment, the latest versions of the applications that are needed to run the business are made available to all customers as soon as an upgrade, or new updated of the software version is released.

2. Cost Flexibility
   Perhaps, one of the most significant benefits of cloud computing is considerable savings for the company. Companies rather than spending lot of money on hardware, software, licensing and renewal fees cloud providers are able to cut down cost by using the resources of your cloud service provider.

3. Mobility Anytime, Anywhere Access
   Whether an employee of an organization is in a business meeting, on vacation or working remotely, the cloud allows organization and its employee to access vital business information no matter where they are in the world.

4. Business Continuity
   In the event of a disaster, be it natural or technical, businesses must be confident that their data was protected, and that it can be restored in a timely manner to ensure the smooth running of the business without any disruption.
5. **Enhanced Security**  
The implementation and strict adherence to a high-level security protocol to ensure data protection by the cloud service providers

6. **Improved Collaboration**  
Google doc is a good example of a cloud application that allows collaboration between employees of an organization to virtually and easily share information in real time.
Chapter 3

Load balancing in the cloud

Knowing what cloud computing is, its underlying components and benefits, load balancing as a key factor, plays a fundamental role in the optimal service delivery in the cloud. Despite the numerous success of the cloud, load balancing is still one of the many challenges of cloud computing. Load balancing is the process of distributing workload across multiple computing nodes to ensure that no single node is overloaded with ongoing tasks whilst other nodes remain idle. Load Balancing It is a technique that ensures no workstation or computer node on the network is over-utilized, using a selective mechanism which in turn can send the incoming jobs to the nodes that are idle by preventing them from going to the nodes that are already overloaded. It can, therefore, be described as the allocation of tasks or jobs to (systems) processors to increase overall systems utilization and throughput, so that the execution time involved in the running of the application is minimal. A common situation where load balancing is used and plays a crucial role is in data centers where the objective of the system is to service incoming client requests with the least turnaround time. Thus clearly shows load balancing is a significant component in cloud computing network infrastructure in achieving a great high-speed network and service delivery.

Basic load balancing principle works like this with the assumption that the client is connected to the internet and all service up running.

1. The user connects to the network and requests for a service through visiting a website. eg (www.cloud.unive.it)

2. The (DNS) Domain Name System of the client IP is routed to a specific IP address at a specific data center offered by cloud.unive.it domain

3. The client request is granted access and is connected to the load balancer.

4. The load balancer based on algorithms then decides on which of the servers should the clients be routed to.

5. The selected server by the load balancer accepts the incoming connection request from the load balancer and ready give back the source to the client.
6. The load balancer always intercepts the packet and changes the source IP before forwarding the packet back to the clients that made the request.

7. The client receives the returned packet of the displayed content of www.cloud.unive.it. Not knowing which servers the packet is from.

![Diagram of load balancing principle]

**Figure 3.1: Load Balancing Principle**

Global Load Balancing

Unlike as in the basic load balancing principle, the global load balancing uses the same concept but at a significantly larger scale.

1. As in the previous, we assuming the client is connected to the internet and all service up running.

2. The user connects to the Internet and requests for a service on www.cloud.unive.it.

3. The Domain Name System (DNS) routes the user to a specific IP address which is connected to the www.cloud.unive.it networks.

4. The user gets connected to the cloud.unive.it (data center).

5. The domain www.cloud.unive.it network’s accepts the connection and, based on a specified policy, decides which data centers to send the user.

6. The user is directed to the datacenter containing the desired application content.

7. Content is delivered to the user via the unive cloud node.

8. The client receives the return packet, not knowing which servers the service came from, and the content of www.cloud.unive.it displays.
3.1 Needs for Load Balancing

The needs for load balancing is connected with the fast growth of the internet and our daily activities. Ranging from surfing websites, data processing, analyzing, uploading, downloading of data etc... Our need for higher internet traffic usages increases dramatically as the ongoing tasks and hence the workload on the servers increases. Some servers particularly those managing some popular websites can easily be overloaded if there is no proper mechanism of load balancing put in place. A common example is the supermarket model where a customer arrives according to an arrival process at a counter of $N$ identical servers, each having a separate queue. The arriving jobs need to be allocated to the servers (counter) so that the load is well balanced and the arrive task are complete in the shortest turnaround time. There is always a possibility that a server (counter) will be overloaded whilst other servers(counter) are idle or under-utilized. As the arriving jobs continue to grow under such demands there is a need for a mechanism of implementing a method where nodes on the network may be heavily loaded whilst other nodes maybe lesser loaded or in an idle state.

The need for load balancing and as well as its problems are not new in distributed systems. A distributed system is simply a collection of independent computers that appears to its users as a single coherent system. In a distributed system the term load balancing and load sharing are often interchangeable. In load sharing, the problem is to develop scheduling algorithms that will automatically transfer processes from heavily loaded workstations to lightly loaded one, with a primary goal to ensure that no
processor is idle while there are processes waiting for services. As in load balancing the algorithms aims at equalizing the processors’ workloads amongst the nodes in the network.

3.1.1 Performance metrics to optimize

Utilization, Mean Response time, Throughput

3.2 Static load balancing algorithms

![Static Load Balancer Diagram]

Load balancing can be broadly categorized into two distinct states, a static and a dynamic load balancing algorithms. In the static approach, load balancing is achieved by providing prior knowledge about the systems and the performance of the node is determined at execution time. The workload is distributed at execution start without much consideration to the current system load. Static load balancing methods are usually non-preemptive. That is, once the load is allocated to a node the load cannot be transferred. This method requires less communication and hence reduces the execution time. However, a setback of this approach is that the algorithm does not take the state of the system into consideration whilst making allocation decisions. This may impact on the overall performance of the system due to load fluctuation. The followings are examples of static load balancing algorithms: Round robin, Central Manager, Threshold algorithm and Randomized algorithm.

1. Static Algorithms and how they worked

   (a) **Round Robin Algorithm:** This algorithm derived its name from the round-robin principle, where each system takes an equal share of something in turn. This algorithm distributes loads to nodes in a round-robin order and assigned equal load to each node in circular
order without any priority. Each node on the network maintains a load index independent from the remote node. The round-robin algorithm is simple and starvation free. Round robin algorithm does not require interprocess communication and gives the best performance for special purpose applications.

(b) **Central Manager Algorithm:** This algorithm is a master-slave relationship. Where the central node is referred as the (master node) and the subsequent the slaves. The master or central node maintains an index of all the slave connected to the network and so their load indexes. The slave having the least load is selected and assigned the job. Whenever a load index changes on the slave node, a message is sent to the master or central node. This algorithm requires a high level of interprocess communication, which can sometimes lead to the bottleneck state.

(c) **Threshold Load Algorithm:** With this algorithm, system load is characterize into three levels: “under-load”, “medium-load” and “overloaded”. Each of the nodes keeps a private copy of its system’s load. Two given parameters are introduced when presenting this algorithm thresh-under when the threshold is underloaded and thresh-upper for the load above the threshold.

\[
\text{Under loaded} = \text{load} < \text{thresh-under} \\
\text{Medium} = \text{thresh-under} \leq \text{load} \leq \text{thresh-upper} \\
\text{Overloaded} = \text{load} > \text{thresh-upper}
\]

At runtime, all the nodes are considered to be underloaded, as services continue running a node may exceed a load threshold limit. In such situation, the threshold algorithm is expected to send a message regarding the new load state to all remote nodes, regularly updating load indexes as the actual load. If the local node is not overloaded and no underloaded node exists then the allocation of a load is done locally otherwise, a remote underloaded node is selected. This algorithm has low interprocess communication and a large number of local process allocations which reduces the overhead and eventually leads to improvement in performance.

(d) **Randomized Algorithm:** Nodes are selected on a random selection basis without any prior information about the current or previous load state of the node. Each node maintains its own load record hence no interprocess communication is required. But sometimes it may cause a single node overloaded while the other node is underloaded.

3.3 Dynamic Load Balancing Algorithms

**Dynamic Load Balancing:** The dynamic load balancing algorithm, unlike the static algorithm, is capable of making system changes and load redistribution to nodes accordingly. Decisions are made at system runtime and no prior
information about the nodes are taken into consideration or needed. The algorithm is usually composed of the following strategies: transfer strategy, location strategy and information strategy. These strategies will further be highlighted in brief in sections below. The transfer strategy basically decides on which tasks are eligible for transfer to other nodes for processing. Location strategy nominates a node for the execution of a transferred task. Information strategy act as the hub or the information center for load balancing algorithm and is responsible for providing functional information to the two previous strategies (location and transfer). The dynamic load balancing algorithms provide a significant improvement in performance over the static algorithms. Most common examples of dynamic algorithms are; Least connection algorithm, local queue algorithm and central queue algorithm.

1. Dynamic Load Balancing Algorithms and how they worked

   (a) **Least Connection Algorithm:** Least Connection Algorithm decides its distribution of load based on the number of connection present within the network. The least connection algorithm maintains the log of all the numbers of connections on each of the nodes on the network. The log of the load balancer increases along with the increasing number of connection and when a connection times out the log on the load balancer decreases. The nodes with the least connection are always selected first.

   (b) **Central Queue Algorithm:** Central queue algorithm is a master-slave relationship which employs a system-wide priority scheduling. However, it is handling of low priority tasks can be poor under high loads, when most of the migration overhead is passed on to the low priority tasks. The primary benefit of using Central Queue scheduling algorithm is its ability to adherence to pure priority scheduling such as Earliest deadline first (EDF) algorithm.

   (c) **Local Queue Algorithm:** This algorithm requires interprocess communication but in a lesser manner as compared to central queue algorithm. When the node becomes under-loaded it makes a request for task activities from the remote hosts. The remote hosts then look up its local list for readily available activities, if there exist any some of the activities are passed on to the host that made the request. Some form of acknowledgement will be required from the remote to the host that made the request.

A good load distribution algorithm typically has the following policies as mechanisms in which various load balancing algorithms are based.

- **Selection Policy:** The selection policy selects a task for transfer, once the transfer policy decides that the processor is a sender. The selection policies may be classified into two set of policies such as the preemptive and a non-preemptive policy. The preemptive policy selects a partially
executed task whilst the non-preemptive policy performs only tasks that are yet to start execution. The selection policy determines if a node is ready to partake in a transfer process, either as a sender or a receiver. The selection policy also determines which of the ongoing processes should be transferred. The transfer policy decides that a node is in a heavily-loaded state, the selection policy then selects a task for transferring. A selection policy considers several factors in selecting a task

1. The overhead incurred by the transfer should be minimal, that is a small task carries less overhead.
2. The selected task should be long-lived so that it is worthwhile to incur the transfer overhead.
3. The number of location dependent system invocation calls made by the selected task should be minimal.

- **Transfer Policy:** Once the selection policy has identified the node the transfer policy then transfer the task. The transfer policy determines the conditions when a task is due for transferred and if the selected candidate is suitable to participate in the transfer, either as a sender or a receiver. This policy is constantly aware of the status the current load of the host and the size of the task. When a new process starts, the transfer policy decides on the condition that the node is a sender if the load on the node exceeds a threshold T1. Otherwise, the threshold T2 is regarded the node as a receiver.

- **Information Policy:** This policy, determined when information about specific nodes is to be collected, from where the collection is made, and what information should be collected. There are three types of information policies:

  1. A demand-driven information policy: This is inherently a dynamic policy, as its actions depend on the system state, the demand-driven policies may be either, sender, receiver or symmetrically initiated.
  2. **Periodic policy:** This policy may be either centralized or decentralized and as denoted, the system collection of states information is done periodically. The transfer policy decides to transfer tasks based on the periodic information collected. Periodic information policies generally do not adapt their rate of activity to the system state.
  3. A state-change-driven policy: The nodes disseminate information about their states whenever their states change by a certain amount. Thus differs from a demand-driven in that, it disseminates information about the state of a node, rather than collecting information about other nodes.
• **Location Policy**: The objectives of the location policy is to find a suitable transfer partner for a node either a (sender or receiver), once the information policy reports the status of the node, the transfer policy determines the node status if its either heavily-loaded or lightly-loaded then selects the destinations node for process the execution. Examples of location policies include the random selection, threshold selection, bidding, pairing and state polling.

1. **Random method**: Means a task is randomly transferred to a node selected at random with no information exchange between the nodes to help with decision making.

2. **Threshold method**: This policy selects a random node and checks whether the node is able to receive the process then it transfers the process. The threshold method intelligently avoids useless task transfers and provides a substantial performance improvement over the random location policy. Select remote sites randomly, but before sending task, poll queue length, per say, if $|Q| > T$, don’t send a task to choose another node. (where $|Q| >$ represent the queue and T as the threshold)

3. **Shortest**: Task is transferred to the node with the shortest queue length. The shortest policy uses more state information, in a more complex manner than the threshold policy. However, according to literature, the shortest policies performance is not significantly better than the threshold. Select K nodes at random, poll them and send to the node with the smallest queue. $Q|$ and $|Q| < T$

4. **Bidding method**: In this method nodes contain managers to send processes and contractors to receive processes. The winning contractor is notified and asked whether it accepts the process for execution or not. However, a contractor is never forced to accept remote process.

**Load Balancing Strategies** Three major parameters exist which usually define the strategies a specific load balancing algorithm will employ and thus provides answers to the three most important questions as to who, what and where load decision is made.

1. **Who makes the load balancing decision**

2. **What information is used to make the load balancing decision**

3. **Where the load balancing decision is made.**

Based on the answer to above questions, we can categorize load balancing strategies as:

**3.3.1 Sender initiated migration policy vs Receiver initiated migration policy**

This answers the question of **Who makes the load balancing decision** and whether the system implemented follows the principle of a sender-initiated or
receiver-initiated policy. In sender-initiated policies, overloaded nodes attempt to move work to lightly loaded nodes whilst in a receiver-initiated policy, lightly loaded nodes look for heavily loaded nodes from which work may be received. Queues are formed at senders if a receiver-initiative policy is going to be used, also for receivers, queues are formed at the receiver if a sender-initiative policy is used. In many pieces of literature, it has been demonstrated that using analytical models and simulations that, sender-initiated strategies generally perform better at low to moderate system loads whilst receiver-initiated strategies perform better at high system loads. Similarly, at high system loads, the receiver-initiated policy performs better than the sender-initiated since it is much easier to find a heavily loaded node.

### 3.3.2 Global vs. Local Strategies

Global or local policies answer the question **What information is used to make the load balancing decision.** In global policies, the performance profiles of all the available workstations on the network are used whilst In local policies, workstations are grouped or partitioned into different sets. Depending on the proposed application a decision can be made in selecting either a global or local policy. For global schemes, balanced load convergence is faster compared to a local scheme. The local schemes minimize extra overhead, however, the reduced synchronization between workstations is also a downfall of the local schemes.

### 3.3.3 Centralized vs. Distributed Strategies

A load balancer is either centralized or distributed, both of which answers the question **Where the load balancing decision is made.** Centralized systems are easy to maintain as there is only one single point of failure. In centralized load balancing strategy decisions are made at a central location for the entire network load and thus can be highly unstable once the main server is disconnected the entire network goes down. In contrast to a distributed strategy, in the even one node fails other nodes will be readily available to pick up for continuity. Centralized strategies are easier to maintain in the event of a single point of failure since the load balancer is located in the master node. In a distributed approach, there is no central node, all the nodes connected to the network have copies of information of other nodes regarding the status of their profile. As soon as the state of a node changes such information is updated to all other nodes in the network.

**Centralized strategy has the following characteristics:**

- A master/node architecture exist and the master node holds the collection of all the tasks to be performed
- The master selects the node to perform the execution
- When a node completes its executing process, the node makes another request for a task from the master.
Chapter 4

Openstack Documentation

In this section, we focus on the history, creation, maintenance and structure of Openstack. Also, dive into its basic components and explain how these components contributed to the successful implementation of a cloud infrastructure,

4.1 History of Openstack

The OpenStack project began through the work of two well-known organizations in 2010. Rackspace, a giant hosting US firm wanted to rewrite the infrastructure code running on its cloud servers offering and considered open sourcing the existing cloud files code. At the same time, AnsoLabs contracting firm for NASA had published the beta code for Nova, a Python-based “cloud computing fabric controller”. These companies converged efforts that shaped the base for Openstack.

Openstack is a set of software tools for building and managing cloud computing platforms for public and private clouds. It is backed by some of the biggest companies in software development and hosting. Openstack is supported by thousands of individuals and community members. Thus, as a result, many think and believe that Openstack is the future of cloud computing and I strongly share the same opinion. OpenStack is managed by the Openstack Foundation that was formed in 2012 independent body providing shared resources to protect, empower, and promote OpenStack software and the community around it.

A non-profitable based that oversees both development and community-building around the project. The Openstack foundation mission is to: To produce the ubiquitous Open Source Cloud Computing platform that will meet the needs of public and private clouds regardless of the organization size, by being simple to implement and massively scalable. Openstack projects do not produce “open-core” software. All the software produced is purely Open Source software.

Moreover, the software is produced by a community and contributor accepted license base. Openstack is purely an open source software, this means that anyone can access the source code and make any changes or modifications
they need to customize their own cloud environment, and can freely share these changes back to the community at large for future adoptions. The technology behind OpenStack consists of a series of interrelated projects delivering various components for a cloud infrastructure solution. All service provides access through an Application Programming Interface (API) so that all the resources can be managed through a single dashboard (Horizon) thus, gives Openstack cloud administrators control, also empowering users to provision resources through a web interface.

Openstack API’s are extensible, meaning you can keep compatibility with a core set of calls while providing access to more resources. The Openstack project is a global collaboration of cloud developers which produces an open standard cloud computing platform for both the public and private clouds. Focusing on the implementation, massive scalability, a variety of rich features and a tremendous extensibility, the project aims to deliver a practical and reliable cloud solution for all types of organizations.

Openstack orchestration solves the problem of virtualization by adds a layer on top of many types of hypervisors within the cloud infrastructure allowing for a more efficient way for the management of hardware and provides Openstack with the ability to distribute application workloads based on demand.

4.2 Openstack Architecture

The conceptual architecture showing the relationships among the openstack services

![Diagram showing the relationships among the Openstack services](image)

FIGURE 4.1: Architectural Components of Openstack!

The Openstack architecture is divided into two sections, the Conceptual and the Logical component. As shown in the picture we can clearly support that
Openstack consists of several independent parts. Most services or components are composed of several processes and all services have at least one API process, which listens for API’s processes requests then passes them on to other components or services. With the exception of the Identity service, each component performs a distinct processes work. All the required services are authenticated through a common Identity service. Individual services on the other hand also interact with each other through public APIs. For communication between the Openstack processes, an AMQP message broker is used storing the service’s state in a database. When deploying and configuring your Openstack cloud, we can choose among several message broker and database solutions, such as RabbitMQ, MySQL, MariaDB, and SQLite. In this work, RabbitMQ was used. OpenStack is accessing through a web-based user interface implemented by the Horizon Dashboard or via command-line clients and by issuing API requests through tools like browser plug-ins or curl. It is very important to understand the architecture before we get into designing, configuration and deploying of Openstack.

4.3 Openstack Components

The Openstack cloud architecture provides us with the information on planning and designing Openstack computing cloud. Further detailing the core concepts, requirements, design criteria of key components and services. The Openstack architecture tries to make each project components as independent as possible. Thus, gives users the option to deploy only a subset of the functionality and integrate it with other systems and technologies that offer similar or complementary functions. However, amongst the components three of which interacts with all the components in the system. Horizon is the Graphical User Interface, front-end that allows administrators to easily and efficiently manage all the projects. Keystone handles the management of authorization of users and Neutron which defines the networks topology and provide connectivity between all the components.

4.4 The basic components

Openstack is like a moving car which is made up of several moving components. Openstack core system allows anyone with a sound knowledge and understanding to add additional components to suit their cloud environment needs. However, there is a ground rule, in collaboration, the Openstack community identified nine key components as their core components. Thus as presented below. Nova is the primary engine behind Openstack cloud with a purpose for deploying and managing large numbers of virtual machines. Nova is capable of handling the automation of large pools of computer resources and can work with widely available virtualized servers, Bare metal MAAS and high-performance computing environment. Nova daemons run as a set on top
of the existing Linux servers to provide virtualization services.

Nova requires these three additional services for its basic functional operations:

- **Keystone**: Keystone services are needed to provide identity and authentication for all Openstack services.

- **Glance**: The glance services provide the compute image repository where all glance images instances are launch.

- **Neutron**: The neutron services are responsible for the provisioning of the virtual or physical networks that compute instances connected and boot from.

![Nova API Diagram](image)

**Figure 4.2: The Nova API**

As presented above, the **nova-api** daemon is the heart of the Openstack Nova. It provides an endpoint for all application programming interface (API) queries. The **nova-schedule** process is a piece of code in Openstack Nova which takes a virtual machine instance from the queue and determines if that instance should run. The nova-schedule implements a pluggable architecture that allows developers to choose or write their own algorithm for scheduling. Through the **nova-scheduling** the daemon dynamic balancing algorithm can be implemented to achieve the desired goal.

The **nova-compute** is a worker daemon, that is responsible in creating and terminating virtual machine instances.

The **nova-volume** is responsible for the creation, attaching and detaching volumes to compute instances.
The **nova-network** this network worker daemon accepts networking requests related from the queue and then performs tasks to manipulate the network.

The **nova-queue** provides a centralized hub for passing messages between daemons.

The **nova-database** stores most of the build-time and run-time state for a cloud infrastructure.

**Swift** is regarded as the storage and file system objects and doesn’t follow the traditional idea of referring to file systems by their location on a disk drive. Openstack Swift enables cloud administrators to refer a unique identifier referencing to the file or piece of information and the allow Openstack to decide where to store this information. Thus, makes scaling easy, as cloud developers need not worry about the capacity of a single system and making sure data is backed up in case of network failure. Swift aims to provide a massive scalability and redundant by writing multiple copies of each object to multiple storage servers within separate “regions”. Regions are a logical grouping of storage servers that are isolated from one another to prevent against failures.

Swift handles authentication through a three-step process:

- User authenticates through the authentication system and receives a unique token.
- User issues a second request to swift passing the token along with the request in the HTTP headers.
- Swift-proxy validates the token and responds to the user request with the help of swift account, swift-container, or swift-object.

Swift authentication can be implemented through web server gateway interface (WSGI).

The swift-proxy service is accessed by clients via the load balancer on the management network. The swift-proxy service communicates with the Account, Container, and Object services on the Object Storage hosts via the storage network. Replication is done via the replication network.

**Cinder** is the block storage service for Openstack and fully integrated with other components such as Nova and Neutron enabling cloud users to manage their own storage. Cinder also allows cloud administrators to create, attach and detach block of devices to servers and as well provide the ability to create storage snapshot. Storage snapshots provide powerful functionalities for backing up data stored and can be restored or used to create a new block storage volume.

The above shows a relation what happens behind the scene as cinder instances are connected to the volumes via the storage network by the hypervisor on the Compute host.
1. When a user requests an image, the “glance-api” service accesses the appropriate store on the storage device over the storage network and pulls it into its cache. When the same image is requested again, it is given to the client directly from the cache registering a shorten the request time.

2. When an instance is scheduled for creation on a compute host, the “nova-compute” service requests the image from the glance-API service over the management network.

3. After the image is retrieved, the nova-compute service stores the image in its own image cache. When another instance is created with the same image, the image is retrieved from the local base image cache.

**Neutron** Networking is a standalone service that often deploys several processes across a number of nodes providing networking capability for
Openstack. Neutron is based on the idea of enabling resource management and network virtualization giving cloud administrators the ability to create their own networks, control traffic and devices attached to one or more networks. Neutron networking ensures that each of the Openstack components already deployed can communicate with one another quickly and efficiently by providing networking models for different applications or user groups. Neutron, also, allows the assigning of dedicated internet protocol address either as static IP addresses or DHCP to dynamically rerouted resources in the cloud infrastructure.

The Openstack Networking components are as follows:

1. Neutron server (neutron-server and neutron-plugin) The neutron-server through the neutron-plugin uses AMQP (Advanced Message Queuing Protocol) to communicate with the database.

2. Plugin agent (neutron-agent) runs on each node to manage local virtual switch. This plug-in based on the plugin determines which neutron agents need to run.

3. DHCP agent (neutron-DHCP-agent) This agent is the same across in all neutron plug-ins and is responsible for maintaining DHCP configuration.

4. Layer 3 agent (neutron-l3-agent) This agent plugins, enables forwarding NAT packets to external network access then to virtual machines on tenant networks.

5. Network provider services This offers more networking features and capabilities to the tenant networks and interact with neutron-server, neutron-plugin, and plugin-agents.
The Neutron project provides networking services between the interface of connected devices.

- Neutron allows user access to an (API) to build rich networking topologies and to configure advanced network policies.
- Neutron support two varieties of plugins (open and closed source) and both introduce advanced network capabilities.
- Uses Layer2 and Layer3 tunnelling protocols to avoid VLAN limits.
- Provides end-to-end quality of service (QoS) guarantees and uses protocols like NetFlow for monitoring of services.
- Layer2 and Layer3 network subnet creation and deletion are easily achieved within Neutron services.
- Offers to boot of Virtual Machines on specific Neutron networks.
- The neutron networking services enable tenants to create advanced virtual network services such as a Firewall as a Service, Load balancer as a Service, and a Virtual Private Network as a Service.

Horizon The Openstack dashboard is a web-based interface that allows you to manage Openstack components and services. Horizon is built on Django, which is a web application framework in Python and represent the only graphical interface which provides a web-based user interface to Openstack services including Nova, Swift, Keystone, etc. The Django-based project aimed at providing a complete Openstack dashboard along with an extensive framework for building new reusable components. Horizon act as the entry point and allows developers to access all the components of Openstack individually through an (API) Application Programming Interface. Its dashboard provides an opportunity to monitor what is going on in the cloud.

Horizon holds several key values at the core of its design and architecture:

- Core Support: Out-of-the-box support for all core Openstack projects.
- Extensibility: So that anyone can add a new component as a “first-class citizen”.
- Manageability: The core codebase of Openstack should be simple and easy-to-navigate.
- Consistent: Visual and interaction paradigms are to be maintained throughout.
- Stable: A reliable application programming interface (API) with a strong emphasis on backwards compatibility.
- Usable: Presenting a simple and awesome interface that convenient users.

The Horizon extensible allows the exposure of third-party products and services, such as billing, monitoring, and other additional management tools integrated together. The core support of Openstack applications and service delivery, ships with three central dashboards, a “User Dashboard”, a “System Dashboard”, and a “Settings Dashboard” can also be made specific for service providers and other enterprises who require customization. Developers can automate access or build tools to manage their resources, developers working on Horizon don’t need to be intimately familiar with the APIs of each Openstack project.

**Keystone** is known as the (Identity Service) and as in all other components, Keystone plays a fundamental role in providing authentication, authorization and identity services for Openstack. Keystone maintains the list of all active users and mapped the list against all the services provided by the Openstack to determine which users have permission to use what service and which group that user belongs to. Keystone supports multiple means of access allowing developers to easily map their existing user access against the Keystone database. Keystone Identity Service supports different plug-ins for its authentication, authorization and identity storage decisions.

**Such plugins include:**

- In-memory key-value a simplified internal storage structure
- SQL database such as MySQL or MariaDB
- PAM Pluggable Authentication Module
- LDAP either OpenLDAP or Microsoft’s Active Directory

Keystone currently supports token-based authentication and user-service authorization. It has recently been redesigned to allow for the expansion to
support proxying external services, authentication and authorization mechanisms such as Open-Authorization, this concept will be very instrumental in the “intercloud” and “intracloud” realization. Open Authorization is a standard for token-based authentication and authorization on the Internet. With “Open Authorization”, cloud user’s account information can be access by third-party services, such as Facebook, Twitter etc., without exposing the user’s password credentials.

**Glance**

Openstack Glance has a client-server architecture that provides a RESTful API to the user through which resources can be requested from its servers component. Glance allows the querying and retrieval of virtual machines image metadata. Images on Glance can be stored in a variety of ways, ranging from simple filesystems to object-storage systems. Image sharing provides cloud moderator to make a private image available to the consumers. Glance projects share images by creating its member’s list and maintain read-only privileges on the image for those members. Glance introduces advanced features enabling high availability, dynamic systems and self-automated optimized data center, allowing hardware maintenance for the under-performing servers without downtimes. Glance is the only module that can add, delete, share, duplicate, store and as well retrieves virtual machine disk images use during instance provisioning.

- **Glance-API**: The Glance-API accepts images and Application Programming Interface (API) calls for image discovery, retrieval, and storage.

- **Glance-registry**: The Glance-registry stores’ processes and retrieves metadata about images such as size and type.
Figure 4.8: The Glance API

- Database: The Glance-database stores’ images metadata and supports many backends including MySQL, SQLite and MongoDB.

- Storage repository: The repository for image files systems supports normal file systems such as AmazonS3, swift, and HTTP for image storage.

Ceilometer is a component of the telemetry project. Telemetry provides the data collection service for Openstack, a single point of contact for billing services to users of the cloud. It provides all the requirements needed to establish customer billing systems across all other components such as metering, monitoring, and system alerts.

Ceilometer telemetry services provide the following functions to Openstack:

1. Efficiently poll data from other Openstack components.

2. Manage events and metering data by monitoring system notifications in other components.

3. Publish collected data on various targets including the data stores and message queues.

The Ceilometer Telemetry consists of the following components:

- Ceilometer-agent-compute which runs on each Compute node and polls for data resource utilized.

- Ceilometer-agent-central located at the central management server to poll for data resource utilization statistics.

- Ceilometer-agent-notification consumes messages from the message queue(s) and notification alert to build metering data.
The above services communicate by using the Openstack messaging bus and data collected from other components is designed to be published to various endpoints for storage and analysis by cloud administrators.

**Heat** is the orchestration component of Openstack with a mission to create a human and machine accessible service for managing the entire lifecycle of the infrastructure and applications within Openstack clouds. The Heat orchestration component integrates with other components allowing the creation of most resources such as compute instances, floating-IP address, volumes images etc. Also, the creation of more advanced functionality such as instance high availability and autoscaling. In simple terms, Heat provides the Openstack users with a way to automate the creation of cloud components like networks, instances, storage devices and much more. Also, enable the launching of multiple composite cloud applications based on templates in the form of text files that can be treated like codes.

Heat architecture is comprised of several Python applications highlighting amongst the four main most common components of the Heat project and each performing a unique function.

- heat-API provides an Openstack-native ReST-API that processes API requests by sending them to the heat-engine over a remote procedure call (RPC).
- heat-API-cfn component provides an API that is compatible with Amazon Web Service CloudFormation and processes the API requests by sending them to the heat-engine using a remote procedure call (RPC)
- heat-engine is the brains of the operation and does the main work of orchestrating, such as the launch of templates and providing events back to the API
4.5 Openstack Networking Components

The Openstack networking service widely refers as the neutron, provides the primary connectivity to all Openstack API’s services also enables the configuration and management of varieties of network services from Layer3 forwarding and NAT to load balancing, firewalls, and virtual private networks. Allows users to set up defined network connectivity in the cloud. Amongst those services, also offers a load balancing feature called “LBaaSv.2” through the neutron-LBaaS service plug-in. The Openstack neutron networking component is responsible for the creation and the management of virtual networking infrastructure. These include switches, subnets, advanced services such as firewalls or virtual private networks. Through the neutron networking services, load balancing is possible to implement due to the network scaling capabilities.

Openstack neutron networking consists of the neutron-server, a database for storage either SQL or MariaDB, and a number of other plug-in agents. Openstack plugins are implemented to accommodate different service and provide flexibility in deploying and maintaining services. Openstack networking is entirely standalone and can be deployed to a dedicated host and integrates with various other components such as keystone for authentication and authorization, Nova to connect each virtual machine to a particular network. Also, horizon to manage users through a web interface.

As presented in the above diagram Openstack networking has four distinct physical data center networks such as Management center, Guess center, External and API network

- Management network is responsible for the internal communication between Openstack components. The IP addresses on the management network should be reachable only within the data center.
4.5.1 Standard networking components

- The Openstack load balancers can be either a software-based or hardware-based devices that allow the even distribution of traffic network several servers in the cloud system. The load balancer should be smart enough to avoid sending traffic to overload server to prevent points of failure. This further improves, the performance, network throughput, and response time of the servers. In the Openstack model, a load balancer receives a request from the frontend web server, which then forwards the request to one of the available back-end database servers for processing. The response from the database server is then passed back to the web server and displayed.

- Tenant networks In this context, the terms tenant is mapped to a particular business unit or organization.

Example creating a private tenant network named “unive-net” for project “lbnetwork”
OpenStack network create --project unive_net --internal lbnetwork

- Flat Network Mode: In this mode, all instances reside on the same network and each instance receives a fixed IP from the pool.
- VLAN Network Mode: Virtual LAN Networking allows users to create multiple providers or tenant networks instances to communicate with each other across the cloud environment.
- GRE and VXLAN Mode: These are encapsulation networking protocols which provide separation among tenants and create overlay networks to actively control communication between compute instances.

- Provider networks the Openstack administrator creates provider networks. These networks can be dedicated to a particular tenant in the data center.
- Subnets Subnets are basically used for management and allocation of IP addresses especially when new ports are created on a network.
  - Ports is a connection point for attaching a single device or devices. Openstack support a range of port numbers
  - Routers provides Layer3 and Network Address Translation (NAT) forwarding to provide external network access for Virtual Machines on tenant networks within Openstack infrastructure.
  - Security groups a set of rule and virtual permission indicating access and restriction to certain services to control instances.
  - Extensions in Openstack is extensible in the sense it allows the introduction of new features in the API without requiring a version change.

- Openstack Switches connect hosts that belong to the same layer-2 network and therefore enables forwarding of the packet received on one port to another. Switches operate at layer-2 in the networking model. That is, they forward the traffic based on the destination address in the packet header.
- Routers in Openstack enables communication between two nodes on different layer-3 networks that are not directly connected to each other. Routers, unlike switches, operate at layer-3 in the networking model and route traffic by forwarded packets based on the destination IP address in the packet’s header.
- Firewalls are used to restrict, regulate, filter traffic to a host based on some defined rules. An Openstack firewall can be either a specialized device connecting two or more networks or a software-based filtering mechanism implemented on an operating system. They can filter
packets based on several criteria such as source IP address, destination IP address, port numbers, connection state, etc. Firewalls are primarily used to protect the hosts, network resources from unauthorized access.

**Server** is a virtual machine (VM) instances whereby virtual servers are created and resources are made available to customers. Users at a different organizational level and profile can create their own networks and connect devices to one or more networks.

Openstack server contains many attributes that indicate the status of the server and few are as shown below.

ACTIVE: Indicates the server is active.
BUILD: The server building up processes ongoing.
DELETED: The server is deleted.
ERROR: The server reporting an error.
MIGRATING: The server is in live migration action.
PAUSED: The server is paused.
REBOOT: The server is in a reboot state.
REBUILD: The server is currently being rebuilt from an image.
FAILURE: The server is reporting failure.

**Plugins** play a fundamental part in connecting Openstack components and supports third-party plugins and drivers that intends to extend network functionalities and implementation of the Openstack Neutron. Plugins can be created and support multiple networking technologies to implement built-ins network functionalities by operators and users. There are two major types of plugins within the Neutron architecture is the “core-plugin” and the “service-plugin”. The core plugin deals with the implementation of the core Neutron API. Whilst the service plugin primary functions include in providing additional network services such as routing, load balancing, firewall etc... The Layer2 driver enables broadcasting and multicast traffic in Openstack to scale out on large overlay networks also enabling traffic to be sent to the relevant agents via encapsulation as a targeted.

**Agents** Neutron network agent handles various tasks used in the implementation of virtual networks. Such agents include “neutron-dhcp-agent” for when a subnet is created the subnet has DHCP enabled by default, “neutron-l3-agent” that enables layer3 forwarding and support IP floating, “neutron-metering-agent” that enables layer3 traffic metering, and “neutron-LBaaS-v2-agent” allows the configuration of multiple listener ports on a single load balancer IP address, among others. An agent is said to be available when the alive status indicates “True”. Neutron Layer3 agent provides Network Address Translation forwarding to ensure their external network access for the virtual machines. This agent enables Openstack to achieves high availability through Neutron service.
4.5.2 Network Traffic Load Balancer

The Openstack Neutron Load Balancer provides cloud users with the abilities to load balance traffic to applications running on virtual instances on the cloud. Neutron provides an API to manage virtual IPs, pools, pool members, and health monitors. Neutron networking service offers a load balancer feature called “LBaaS-version2” through the neutron-lbaas service plug-in. LBaaS-v2 allows the configuration of multiple listener ports on a single load balancer interface. Openstack provides a reference to two implementations of LBaaS-v2, first using an agent-based implementation with HAProxy and another with the LBaaS-v2 implementation using Octavia. Both Octavia and HAProxy are open source, offers high availability and load balancing solution designed to work with Openstack and use the LBaaS-v2 API. Octavia achieves its goal of load balancing services by managing a fleet of virtual machines, containers, or bare-metal-servers collectively known as “amphorae”.

FIGURE 4.12: Openstack Load Balancer

LBaaS-v2 presents several concepts in which Neutron networking provides an API to manage virtual IPs, pools, pool members and health monitors.

- Load balancer provides a single API and occupies a neutron network enabling the user to make a request for services through the load balancer.

- Listener Load balancers can listen for requests on multiple ports and each port is attached to a specific listener.

- Pool A pool holds a list of members that serve contents through the load balancer.

- Member are regarded as a set of servers that route traffic behind a load balancer. Each of the servers has a dedicated IP address and port number that it uses to serve traffic within a given subnet.
• Health monitor is constantly aware of the status of the server. Whenever a server goes offline the health monitors divert the incoming requests away and relay the traffic to the responding or online.

LBaaS-v2 API implementation based on the HAProxy opensource software load balancer. Creates a load balancer pool by using specific provider.

**LISTING 4.1:** Creating a load balancer pool with various algorithm

```bash
neutron lbnetwork1-pool-create —lbnetwork1—method ROUND_ROBIN
—name mylbpool —protocol HTTP —subnet-id SUBNET_UUID
—provider LBNETWORK1
neutron lbnetwork1-pool-create —lbnetwork2—method IP_HASH
—name mylbpool —protocol HTTP —subnet-id SUBNET_UUID
—provider LBNETWORK2
neutron lbnetwork1-pool-create —lbnetwork3—method LEAST_CONN
—name mylbpool —protocol HTTP —subnet-id SUBNET_UUID
—provider LBNETWORK3
```

Associating the three web servers with the pool as implemented in the project.

**LISTING 4.2:** Associating web servers to myloadbalancer pool

```bash
neutron lbnetwork1-create —address LBNETWORK1_IP — 80 mylbpool
neutron lbnetwork2-create —address LBNETWORK2_IP — 80 mylbpool
neutron lbnetwork3-create —address LBNETWORK3_IP — 80 mylbpool
```

Creates a health monitor that checks to make sure our instances are still running on the specified protocol-port.

**LISTING 4.3:** Creating a health monitor for the load balancer

```bash
neutron lbnetwork1-healthmonitor-create —delay 3 —type HTTP —max-retries 3 —timeout 3
neutron lbnetwork2-healthmonitor-create —delay 3 —type HTTP —max-retries 5 —timeout 5
neutron lbnetwork3-healthmonitor-create —delay 3 —type HTTP —max-retries 6 —timeout 6
```

Associates a health monitor with pool.

**LISTING 4.4:** Associating a health monitor to the 3 load balancer servers

```bash
neutron lbnetwork1-healthmonitor-associate
HEALTHMONITOR_UUID mylbpool
neutron lbnetwork2-healthmonitor-associate
HEALTHMONITOR_UUID mylbpool
neutron lbnetwork3-healthmonitor-associate
HEALTHMONITOR_UUID mylbpool
```

Creates a virtual IP (VIP) address that, when accessed through the load balancer, directs the requests to one of the pool members.
LISTING 4.5: Creating a virtual IP

neutron lbnetwork1-vip-create --name mylbnetwork1-vip \
   --protocol-port 80
   --protocol HTTP --subnet-id SUBNET_UUID mylbpool
neutron lbnetwork2-vip-create --name mylbnetwork2-vip \
   --protocol-port 80
   --protocol HTTP --subnet-id SUBNET_UUID mylbpool
neutron lbnetwork3-vip-create --name mylbnetwork3-vip \
   --protocol-port 80
   --protocol HTTP --subnet-id SUBNET_UUID mylbpool
Chapter 5

Load balancing algorithms in Openstack

In Openstack load balancing mechanisms, the load balancer listens to the network ports for incoming services request. When a request from a node arrives, the load balancer uses a scheduling algorithm to assign a requester, the service requested through the load balancer honouring the specific request made. Openstack implement various algorithms to determine the best possible route the server needs to send its traffic and also in distributing workloads between multiple backend systems or services referred as nodes. Amongst the most popular algorithms in Openstack are as follow:

5.1 Least Connection Algorithms

As the name denotes, “chooses the least connected node” the least connections scheduling algorithm methods function best in environments where sets of servers have similar processing capabilities. As in the implementation, all the three servers share similar capabilities. Considering the case where a pool of the two servers LBSERVER1 and LBSERVER2 are used, also considering their differences in CPU usage and processing power for server LBSERVER1 and LBSERVER2. To further explain this algorithm, LBSERVER1 with 85 active connections with a connection limit of 100, whilst server LBSERVER2 with 95 active connections with a much larger connection limit of 200. In the implementation, the Least connections scheduling algorithm method selects LBSERVER1, which is the server with the least number of active connections, even though the server is close to reaching maximum capacity.

The least connection scheduling algorithm handles connections request to the server with the least number of connection established, the scheduling algorithm is a dynamic process in nature because it counts the active connections for all the servers dynamically. There can be instances where two or more servers have the exactly the same requirement and specifications. Due to demand for services provided, there is always a high likelihood that one server can still get overloaded considerably faster than the other.
5.2 Random

As the name implies, the random algorithm matches between nodes randomly using a random number generator on all nodes. When a workload on a node is greater than threshold-load then a random number is generated in a processor, and the load is migrated to a randomly selected neighbour node. The algorithm does not check the state information of a node neither maintains any local load information nor sends any load information to other processors. The Random algorithm is suitable for clusters consisting of nodes with similar CPU power. The algorithm also is known to causing considerable communication overheads due to its nature of the random selection of nodes.

5.3 Weighted-Least-Connection

The Weighted Least Connections algorithm introduces a “weight” component respectively to the capacities of each server on the network. The load balancer implements the weighted least connections algorithm on the basis taking into consideration two major components: the “weights” and the “capacities” of each server also considering the active number of clients connected to each server at that specific time. Unlike the random algorithms, the weighted least algorithm works best in environments where the cluster servers have different processing capacities.

5.4 Weighted-Round-Robin

The Weighted round-robin scheduling algorithm is designed to better handle servers with different processing capabilities, unlike the round-robin algorithms. Each of the server on the network is assigned with a value that indicates their processing capacity. The servers with the higher weights receive a new connection first then followed with the next weights until the last server is served. The servers with higher weights get more connections than those with fewer weights and the in the event some of the servers have equal weight, they receive equal connections. The round-robin aspect of the algorithm allows each queue to be serviced in a set of order, sending a limited amount of data before moving onto the next queue and cycling back to the highest priority queue after the lowest priority queue is serviced. This algorithm performs better than the round-robin in an environment where the processing capacity of the servers is different.

5.5 Round Robin

The round-robin scheduling algorithm is widely used in the implementation of load balancing and routing internet traffic. The round-robin is simple in nature and easy to understand and implement. Per say, between the three servers as implemented, (LBSERVER1, LBSERVER2 and LBSERVER3). When
the first request arrives the load balancer will forward the request to LBSERVER1. When request 2 arrives it will go to LBSERVER2, and so on. Because LBSERVER3 is the last server on the queue if a fourth request comes the package is forwarded to LBSERVER1. A new connection request is passed to the next available server on the queue, eventually distributing connections evenly across the servers the balance load. Round Robin scheduling algorithms treat all servers are equals regardless of the number of incoming requests, therefore, handling all servers without priority. Amongst the advantages of “Round Robin” algorithm is that it is a simple and easy to implement in Openstack and also regarded fair in the sense, that every process gets an equal share of the CPU power. If the number of processes in the network known the number of queues can also be known and the worst-case response time for processes can be calculated. Sometimes two or more processes may have the same priorities and one of the processors might get suspended in the middle of execution, as the algorithm works on first come first serve basis.

5.6 Load Balancing Strategies in Openstack

Load balancer handles the neutron network port which has an IP address assigned from its internal subnet. The “listener” listens to requests made on multiple designated ports such as port 80, 8080, and so on. The “Pool” holds a list of members that serve services through the load balancer. Member in this context are basically servers that serve traffic behind a load balancer. The “Health monitor” monitors the status of the servers, should a server go offline, its the job of the health monitors to divert traffic away from those members that are not responding.

Openstack implies this strategy to load balance among nodes in different ways which are collectively called the lb_method and the most common forms of algorithms are as follows:

- **ROUND_ROBIN**: The load balancer will select a node for workload handling on a round-robin basis. Each node gets an equal pressure to handle workloads.

- **LEAST_CONNECTIONS**: The load balancer will choose a node based on the number of established connections from a client. The node will the lowest number of connections will be chosen.

- **SOURCE_IP**: The load balancer will compute hash values based on the IP addresses of the clients and the server and then use the hash value for routing. This ensures the requests from the same client always go to the same server even in the face of broken connections.

The above is achieved through the pool.admin_state_up and the pool.session_persistence properties:
• The `pool.admin_state_up` property is useful only when you want to debug the details of a load balancer.

• The `pool.session_persistence` plays a fundamental role and supports three types of session persistence to achieve the load balancing.

• `SOURCE_IP`: With the `source_ip`, the load balancer will attempt to resume a broken connection based on the client’s IP address. Openstack flexibility allows us to not configure the `cookie_name` property in such a case.

• `HTTP_COOKIE`: The load balancer will check a general HTTP cookie using the name specified in the `cookie_name` property and then attempts to resume the connection based on the cookie contents.

• `APP_COOKIE`: Thus allows the load balancer to check the application specific cookie by using the name specified in the `cookie_name` and attempts to resume the connection based on its contents.

### 5.6.1 Health Monitor

The fact that Openstack load balancer sits in front of all nodes and is aware of what is consistently happening in a pool, it also needs to be aware of the health status of all member nodes so as to properly and reliably route client requests to the active nodes for processing should a node fails to respond. Openstack load balancer allows and support four types detection of nodes failure

• `PING`: The load balancer pings every pool members to detect if they are still reachable.

• `TCP`: The load balancer attempts a telnet connection session to member of pool thus determines if a node is still alive.

• `HTTP and HTTPS`: These two protocols, allows the load balancer to checks nodes aliveness by sending HTTP/HTTPS using `health_monitor.http_method` configuration and `health_monitor.url_path` property compares the result code to the expected value configured in the `health_monitor.expected_codes`.

The `health_monitor.expected_codes` accepts a string value as an indicator of node’s aliveness such as:

• **Accepts a single value, such as:** 200;

• **Takes a list of values separated by commas, such as**: 200, 202, 204;

• **Expect a range of values, such as**: 200-250.

The health monitor achieve a reliable failure detection system on Openstack allows the configuration property of the `health_monitor`
• **Timeout**: The maximum time in milliseconds that a monitor waits for a response from a node before it claims the node is inactive.

• **Max_retries**: The number of connection failures attempts before the health monitor marks that node inactive.

• **Delay**: The time in milliseconds between time lapse of sending two consecutive requests to pool members.
Chapter 6

Installation of Openstack and load balancer

As indicated in previous sections, Openstack consists of several key projects that can be installed separately using opensource cloud computing platform. The core platform supports all types of cloud environments depending on the cloud needs. These individual projects include Nova (Compute), Keystone (Identity Service), Neutron (Networking), Glance (Image Service), Cinder (Block Storage), Swift (Object Storage), Ceilometer (Telemetry) and Heat (Orchestration)

As in most computing system, Openstack requires certain core components for its operational functionalities to achieve simple implementation for a massive scalability.

Hardware requirements includes:

- **Controller:** node runs the Keystone, Glance, and a portion of Nova including various Neutron Networking agents and Horizon for the dashboard. The controller supports multiple services such as SQL, MariaDB databases, RabbitMQ messaging queue, and Network Time Protocol (NTP). The controller node, however, requires a minimum of two network interfaces for proper operation.

- **Compute:** node runs the virtualization aspect of Nova that operates instances and also runs a Neutron Networking service agent that connects instances to virtual networks.

- **Block Storage:** node contains the disks that the Block Storage service provisions for instances. However, each node also requires a minimum of one network interface.

- **Object Storage:** the object storage service stores user accounts, containers, network data and so on, each node requires a minimum of one network interface and more than two object storage can be deployed on a node.

- **Networking:** The two networking sets (Public and Private) are requirements are for the Provider and Self-service networks. The provider networks help to deploy the Openstack Neutron service
supporting layer-2 services and VLAN segmentation. Additionally, Provider network supports the DHCP service to instances. The Self-service networks supports the provider networks with layer-3 routing by enabling full self-service networks.

**Other requirements includes:**

- **Security**: Openstack cloud computing services support various security methods including password, policy, and encryption.
- **Host networking**: All nodes require Internet access for administrative purposes and enabling the installation of packages over the internet, security updates, DNS, etc.
- **Network Time Protocol (NTP)**: Help in the synchronization of service among nodes.
- **Openstack packages**: Enable additional repository package distribution
- **SQL database**: uses NoSQL services to store information
- **NoSQL database**: CeilometerTelemetry uses NoSQL services to store and retrieve information such as billing and users reports
- **Message queue**: Openstack supports several message queue service and such includes RabbitMQ, ZeroMQ and uses the message queue to handle operation and status information between services.

### 6.1 Horizon (Dashboard)

In this section, we provide the basic installation of Horizon and other components of Openstack as implemented in the project demonstration. Horizon is the implementation of dashboard on Openstack which provides a web user interface for administration and management purpose.

Horizon installation includes the following requirements

- **Python 2.7 and above**
- **Django 1.8 and above**
- **An accessible keystone endpoint**
- **Cinder**: Block Storage
- **Glance**: Image Management
- **Heat**: Orchestration
- **Neutron**: Networking
- Nova: Compute
- Swift: Object Storage
- Horizon also supports many other services installation via plugins and hence through which the Load-Balancing-as-Service Firewall-as-a-Service and VPN-as-a-Service are installed.

Devstack all in one installation for the Openstack environment for cloud computing and following the use of Devstack provides a single, multiple node(s) installation to provide an Openstack environment for development and testing, we shall demonstrate the following through the following steps.

First through the unix apt-get download package repository: “git” is downloaded and cloned to OpenStack “GitHub” the Openstack resource library

```
1. sudo apt-get install -y python-setuptools python-virtualenv python-dev gettext
   git gcc libpq-dev python-pip python-tox libffi-dev
2. sudo apt-get install -y git
3. git clone https://git.openstack.org/openstack-dev/devstack
```

Upon successfully installing and cloning devstack

The navigation into devstack which contains a script that installs Openstack and templates for configuration and enables the customization of localrc and local.sh script

```
# Misc
ADMIN_PASSWORD=mypassword
DATABASE_PASSWORD=mypassword
RABBIT_PASSWORD=mypassword
SERVICE_PASSWORD=mypassword
SERVICE_TOKEN=mypassword

# Target Path
DEST=/opt/stack

# Enable Logging
LOGFILE=$DEST/logs/stack.sh.log
VERBOSE=True
LOG_COLOR=True
SCREEN_LOGDIR=$DEST/logs

# Nova
enable_service n--novnc n--cauth
nova--api, nova--cert, nova--conductor,
nova--consoleauth, nova--novncproxy,
nova--scheduler, python--novaclient
```
# Neutron
disable_service n-net
ENABLED_SERVICES+=,q-svc,q-agt,q-dhcp,q-l3,q-meta,neutron
ENABLED_SERVICES+=,q-lbaas,q-vpn,q-fwaas

# Swift
enable_service s-proxy s-object s-container s-accounts
SWIFT_HASH=66a3d6b56c1f479c8b4e70ab5c2000f5

# Ceilometer for the component
enable_service ceilometer–acompute ceilometer–acentral
ceilometer–anotification ceilometer–collector ceilometer–api
enable_service ceilometer–alarm–notifier ceilometer–alarm–evaluator

# Heat
enable_service heat h-api h-api–cfn h-api–cw h-eng

# Sahara
enable_service sahara

HOST_IP=192.168.137.100
FLOATING_RANGE=192.168.137.1/24
PUBLIC_NETWORK_GATEWAY=192.168.137.1
Q_FLOATING_ALLOCATION_POOL= start=192.168.137.10, end=192.168.37.200

Once the above configuration is done, we execute the script in the devstack directory using:
./stack.sh command
The above image is a successful indication of the Openstack installation on the System and now giving access to the Dashboard through the HOST-IP.

After we successfully logged in, the dashboard defaults to the “Admin tab”. From here, we can navigate one set of information graphically to another. As presented in the following screenshot, the System Information panel provides the user with information about the environment, including Services and Compute Services.
The API Access section describe operations that are common to all OpenStack APIs and their service endpoint.

Server instances shows all active servers as listed.
The screenshot shows at this point in the installation, the Openstack Identity, Image, Horizon Dashboard, and Compute services have been successfully deployed.

6.1.1 Creating the Load Balancer

The basic reason that motivates Openstack load balancing can be classified in two folds. First, the ability of the cloud to scale applications and secondly to provide resiliency and redundancy to the applications in the cloud. Openstack LBaaSv1 implemented but with a single point of failure which was the Load balancer itself. An improvement was made in the LBaaSv2, instead of having a single process, the function of the load balancer has now been moved into separate instances with a controller. Each time a request for a new load balancer is made the controller spawn a new set. Setting the controller also as a single point of failure.

The introduction of the HAproxy into Openstack load balancer overcome all the problems of a single point of failure. Neutron LBaaS traditionally uses a standard Linux tool called HAproxy, which is a standard for Linux load balancing. It is used as the underlying load balancing infrastructure in the production of workloads for spreading the software load between instances.

LBaaSv2 introduces several new concepts:

**Load balancer** handles the IP address and the port which is assigned from the internal subnet.

**Listener** can listen for requests on multiple designated ports.

**Pool** holds a list of members that serve services through the load balancer.

**Member** are basically servers that serve traffic behind a load balancer.
Health monitor monitors the health status of the servers, should in case a server goes offline, the health monitors divert incoming traffic away from members that are not responding to members that are active and running.

The most common Openstack load balancer implementations use either an agent or the Octavia services. The following methods demonstrate how we configured LBaaSv2 using Octavia.

1. We first edit service-plugins configuration directive in 
   /etc/neutron/neutron.conf.
   service-plugins = [existing service plugins]
   neutron-lbaas.services.loadbalancer.plugin.LoadBalancerPluginv2

2. We then add the Octavia LBaaSv2 service provider to the 
   service-provider configuration directive in /etc/neutron/neutron.conf
   service-provider = LOADBALANCERV2:Haproxy:neutron-lbaas.drivers.haproxy.plugin-driver.HaproxyOnHostPluginDriver:default

3. Ensuring all neutron agents are stopped before restart the network service to activate the new configuration.

4. On completing successfully, we can view the load balancer status and IP address with the “lbaas-loadbalancer-show-command”

The neutron-lbaas-agent service runs on the network node. Openstack load balancer server is a server-client base, various forms of errors may exist and
such as the Error Response Code: 401 (Unauthorized access), 404 (Page Not Found), 409 (Conflict Request indicating that it can’t complete the request), 413 (Over limit), 500 (Internal server error) etc. Once the request is validated and progress has started on the provisioning process, a response object will be returned containing a unique identifier and the status of provisioning the Load Balancer. Such responses can have the values as ACTIVE, PENDING-CREATE or ERROR. When the status of the load balancer indicates ACTIVE, then the load balancer has been successfully provisioned.

LBaaSv2 supports implementations via different service plug-ins. Both implementations use and support the LBaaSv2 API and the implementation to achieve as follows.

6.2 Configuration and Creation of LB in Openstack

Creating Load balancer: neutron lbaas-loadbalancer-create [–name lbnetwork1] [–vip-address 192.168.137.20] 192.168.137.0


This object will contain a unique identifier and the requester of the operation must specify at least the following attributes of the Listener: tenant-id: this is only required if the requester or caller has an admin role and wants to create a Listener for another tenant. loadbalancer-id: The load balancer ID the listener will be provisioned on. protocol-port: The port in which the frontend will be listening. There is a range of available ports between 1-65535 mostly recommended for port 80 8080. admin-state-up: The default value for this attribute should always be set to true. name: The name attributed to this value is a string type and can be empty. description: A string type attribute and value can be an empty string. connection-limit: Setting the default value for connection limit to -1, indicating an infinite limit.


This operation creates a new Pool based on the configuration defined in the load balancer server object. Once the request is validated, a response object will be returned and the object will contain a unique identifier.

This operation must specify at least the following attributes of the Pool:

- tenant-id: to create a “POOL” for another tenant.

- protocol: The available protocols members will be listening eg. TCP, HTTP, or HTTPS
• **lb-algorithm:** The load balancing algorithm to distribute traffic to the pool’s members. Must be one of ROUND-ROBIN, LEAST-CONNECTIONS, or SOURCE-IP.

• **listener-id:** The listener in which this pool will become the default pool.

• **admin-state-up:** The default value for this attribute is true.

• **name:** The name attributed to this value is a string type and can be empty.

• **description:** A string type attribute and value can be an empty string.

• **session-persistence:** The default value for this is an empty dictionary.

**Creating member:** `neutron lbaas-member-create [–weight WEIGHT] [–name lbnetwork1] –subnet SUBNET –address ADDRESS –protocol-port PROTOCOL-PORT POOL`. This operation provisions a new Member to the Pool based on the configuration defined and once the request is validated, a response object will be returned containing a unique id number.

This operation must specify at least some of the following properties:

• **tenant-id:** to create a Pool for another tenant.

• **address:** The range of IP Addresses member of the pool to receive traffic from the load balancer.

• **protocol-port:** The designated port numbers that the member of the pool is listening to receive traffic from.

• **subnet-id:** Defines the list of subnets accessing Pool members.

• **admin-state-up:** Attributes the status of the pool with a default value as TRUE.

• **weight:** The default attribute value is equal to 1, a zero indicates a non-running instance.


Health monitor helps to determine whether or not the back-end members of the virtual IPs pool are available for processing a certain request. OpenStack LBaaS supports different types of health monitors such as PING, TCP, HTTP, HTTPS. A PING monitor to pings the members to ensure that the member is alive.
The following properties must be put into consideration to enable load balancer to successfully monitor member of pool status:

- delay: The minimum delay in seconds pinging members of the pool.
- timeout: The maximum time designated for the monitor to wait for a ping reply before it classifies it as times out.
- max-retries: number of times permissible for a ping failure before changing the member’s status to INACTIVE.

The health monitor can assume one of the following values as status: ACTIVE, PENDING-CREATE or ERROR.

LBaaSv2 Openstack Neutron offers a number of commands as we just displayed and that can be used to create and manage virtual IPs, pools, pool members, and health monitors for load balancing purposes.

- In the creation of the pool, the “–lb-method attribute” is used to specify the load balancing algorithm which is used to distribute traffic among the pool members. Possible options include the three algorithms popularly used “ROUND-ROBIN, LEAST-CONNECTIONS, or SOURCE-IP.”

- The “–name attribute” is used to specify a name for the pool.

- The “–protocol attribute” is used to specify the type of traffic that the pool will balance and example of such protocols are TCP for TCP traffics

- The “–tenant-id attribute” is optional and allows the associate the pool member with the specific tenant.

- The “–weight attribute” allows you to associate a weight with the pool member.

- The “–address attribute” is required and used to specify the IP address of the pool member

- The “–max-retries attribute” is used to specify the maximum number of failures before a pool member is marked as DOWN. A mostly adopted value is 3 retries

- The “–timeout attribute” is used to specify the number of seconds required for a monitor to wait for a connection to be established.

To create a load balancer operation the following attribute must be specified:

- tenant-id: To create a load balancer for another tenant.

- vip-subnet-id: The network on which the load balancer’s virtual ip address is allocated from.
• admin-state-up: The default value for this attribute is true, false is will represent the failure of an operation.

• name: Attribute of the name value is usually an empty string.

• description: Value takes a string name attribute, allows empty string for description.

In a plain text output of the script looks as follow

```json
{
  "loadbalancer": {
    "description": "lbnetwork",
    "admin_state_up": true,
    "tenant_id": "c8x1a69e88bf4b12a1855f877alf0054",
    "provisioning_status": "ACTIVE",
    "listeners": [],
    "vip_address": "192.168.137.20",
    "vip_subnet_id": "005d5930−93b6−63c4−13f6−d721086ae0x7",
    "id": "f97c30d0−06e9−95ec−33ac−28a95532xf2w",
    "operating_status": "ONLINE",
    "name": "lbnetwork2"
  }
}
```

This operation returns the successful creation of the load balancer object.

### 6.3 Updating the Load Balancer

Upon the successful validation of the request, the service will return an “accepted response” code. If the load balancer provisioning status is “PENDING-UPDATE” then we wait for the changes to be applied and the provisioning status changed to ACTIVE. The updating load balancer operation allows changes of the following Load Balancer attributes:

• name: Name attribute of this value will be an empty string.

• description: The description, attributed with a string and supports an empty string.

• admin-state-up: The default value for this attribute is true

In a plain text output of the script looks as follow

```json
{
  "loadbalancer": {
    "description": "lbnetwork",
    "admin_state_up": true,
    "tenant_id": "c8x1a69e88bf4b12a1855f877alf0054",
    "provisioning_status": "PENDING_UPDATE",
    "listeners": [],
    "vip_address": "192.168.137.20",
```
"vip_subnet_id": "005d5930-93b6-63c4-13f6-d721086ae0x7",
"id": "f97c30d0-06e9-95ec-33ac-28a95532xf2w",
"operating_status": "ONLINE",
"name": "lbnetwork2"
}

This operation returns the updated Load Balancer object.

### 6.3.1 Remove a Load Balancer

The remove operation removes the specified load balancer and its associated configuration files from the tenant account. The operation unlike creating and updating the load balancer does not require a request or a response body. Currently, Openstack doesn’t support data recoverable once purged.

### 6.3.2 List all load balancers

This operation does not require a request body and returns the list of all properties of the load balancers associated with the tenant account. Each element in the list is a load balancer can contain the following attributes:

(a) id:

(b) tenant-id:

(c) name:

(d) description:

(e) vip-subnet-id:

(f) vip-address:

(g) admin-state-up:

(h) listeners:

(i) provisioning-status:

(j) operating-status:

### 6.3.3 Load Balancer integration With ProximityScheduler

Openstack also uses the proximity scheduling of hosts which is very critical to achieving high performance in its cloud computing. The scheduler is dynamic and is able to use the neutron topology and location information of each host. When more than one compute instances is requested and the proximity scheduler requirement is set, the scheduler allocates a set of hosts that are close each other. It involves modification of filter-scheduler so that it can be able to consider the available hosts at the same time and returns the best-chosen hosts instead of returning one by one.
Figure 6.2: Load balancer properties.

```bash
root@lbnetwork1:~$ neutron lbaas-pool-show ac99b221-f144-4860-b490-18b7906f79b5
neutron CLI is deprecated and will be removed in the future. Use openstack CLI instead.
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin_state_up</td>
<td>True</td>
</tr>
<tr>
<td>description</td>
<td>Alt Traffic</td>
</tr>
<tr>
<td>healthmonitor_id</td>
<td>faa49c06-0307-45c7-a37c-d37a2a2d2104</td>
</tr>
<tr>
<td>id</td>
<td>ac99b221-f144-4860-b490-18b7906f79b5</td>
</tr>
<tr>
<td>lb_algorithm</td>
<td>SOURCE IP</td>
</tr>
<tr>
<td>listeners</td>
<td>{&quot;id&quot;: &quot;8b702870-a55b-4bae-a2f6-14374f327282&quot;}</td>
</tr>
<tr>
<td>loadbalancers</td>
<td>{&quot;id&quot;: &quot;5016e11f-44c3-44cd-a22b-a203a1878161&quot;}</td>
</tr>
<tr>
<td>members</td>
<td>3074d6be-7c6b-4e3b-a773-c98b606010b0f</td>
</tr>
<tr>
<td>name</td>
<td>c666367a-d32a-4f46-b24c-df6b2b5ef89f</td>
</tr>
<tr>
<td>protocol</td>
<td>Pool 1</td>
</tr>
<tr>
<td>session_persistence</td>
<td>HTTP</td>
</tr>
<tr>
<td>tenant_id</td>
<td>a399b68b6b11435eb7a0234d9528631</td>
</tr>
</tbody>
</table>
Chapter 7

Openstack High Availability

7.0.1 Characteristics Openstack High Availability

Openstack high availability cloud systems seek to minimize the following:

- System downtime: is said to occur when cloud users are unable to access services over a specified maximum amount of time.

- Data loss: Thus as a result of accidental data deletion or a deliberate destruction of data.

High availability in Openstack protects cloud users against system downtime and data loss. Moreover, prevent the unanticipated system single point of failure and cascading failures. Cascading failure is a result of a single failure escalating into a series of consequential system failures in the network.

Redundancy and failover A crucial aspect of Openstack high availability is its ability to eliminate single points of failure. That is, If an instance of a service fails, the system can then failover to use another instance of another hardware that did not fail. In the event that a component fails then a backup system must take on its load, Openstack high availability systems replaces the failed component to maintain necessary redundancy.

Stateless versus stateful services Openstack classifies stateless and stateful services as follow: When a request is made is and a response is provided and then no further action is required thus is regarded as a “Stateless service”. Openstack stateless service includes nova-conductor, glance-API, nova-API, keystone-API, nova-scheduler and neutron-API. To successfully achieve a stateless services highly availability, we must provide redundant computer instances and load balance between them.

Stateful service Unlike in the former, stateless services requests to another service depends on the results of the first request obtained. A stateful service usually is more challenging to maintain and configure, simply because for every single action may involve more than two requests. Stateful services in Openstack includes such SQL, MariaDB databases, message queue (RabbitMQ) etc.

To achieve a stateful service highly available depend on whether we adopted an active/passive configuration or we adopted the active/active configuration
settings. The active/passive configuration basically maintains a redundant instance that can be brought online when the active server fails whilst the active/active configuration, each service has a backup but manages both the main and redundant systems concurrently in this way, if there is a failure, the user is unlikely to notice. In such, updates to one instance on the database updates all other instances and hence allowing Openstack to load balancer traffic between the instances.

**High availability for instances** Openstack high availability works on shared storage and local storage environments, which means that migrated instances will maintain the same network configuration either using static-IP, floating-IP and so on, even if created from scratch. HA requires three resource agents to work and such as fence-compute to mark a Compute node for evacuation when the node becomes unavailable, Nova-Evacuate: handles the evacuation of instances from a failed node, and runs on one of the Controller nodes, nova-compute-wait: Restarts the Compute services of an instance once evacuated. Openstack uses the command line interface or through the dashboard (Horizon) to server migrate or move an instance from one compute host to another. In such a case, the scheduler chooses the destination of the compute host based on its configuration settings.
Chapter 8

Conclusion

Openstack load balancing consists in aggregating multiple components and services in order to achieve total processing capabilities without any intervention from the end user, diverting traffic from its natural flow and maintain a minimum required level of consistency between all routing decisions within the Haproxy configuration. Openstack is best for the implementation of cloud data center and load balancing in a cloud environment. Because of its ability for making data cluster highly available along with high latency in maintaining zero fault tolerance between applications hosted inside the Virtual Machines in a cloud environment.

8.1 Analysis and classification of the load balancing algorithms used by the plugins

In this experiment, we set up three node clusters with Ubuntu Server 16.04 LTS installed on all the nodes. These nodes are named as LBSERVER1, LBSERVER2 and LBSERVER3 assigned with 192.168.137.10, 192.168.137.20 and 192.168.137.30 as IP addresses respectively. Then a third IP address, 192.168.137.100 used as a Virtual IP address (VIP) mapped to the cluster database using MariaDB.

In the implementation of a dynamic load balancing cloud service with Openstack, three distinct algorithms were used in this project. Amongst which includes the Least Connection Algorithms, we realized that under this algorithm with Openstack, the load balancer will choose a node based on the number of established connections from clients, therefore, choosing the node with the lowest number of connections.

The second algorithm implemented was the Round Robin method: The Openstack load balancer will select nodes on a round-robin basis first come first serve. This method, each node gets an equal pressure to handle workloads on the basic and as the job arrives. This algorithm didn’t perform very well compared to the Least Connection using the cookie based-insertion simply because the algorithm doesn’t give special priority nor picking up the fastest server.
Last on the list, Source_IP: Here the load balancer will compute hash values based on the IP addresses of the clients and server and then use the hash value for routing traffics. This makes sure that a request from the node always goes to the same server even if the connection is broken. We found out in the advent, cookies are disabled on a browser there is usually some problem within web component (Horizon) but using the “SOURCE IP” load balancing algorithm instead of the “round robin”. The algorithm makes sure that a given IP address always reaches the given server so long the network topology of the servers remains unchanged. However, the algorithm will fail if there are changes to the network. The least connection algorithm is more resilient amongst the three algorithms used.

8.2 Future Work Unsolved problems

Due to architectural limitation ONLY, five out of the nine components of Openstack was installed to be able to demonstrate and achieve its load balancing as a service component. The architectural configuration for server LBSERVER1, LBSERVER2 and LBSERVER3 were not suitable to support the presence of controllers which required two network interfaces to be properly configured to fulfil the goal of cloud services and migration with Openstack. Also, we needed to install cinder service component of Openstack as a storage cluster. Using either Ceph or GlusterFs, Ceph cluster in Openstack is designed around replication and information distribution whilst Glusterfs uses a hashing algorithm to place data within the storage pool.

Both are very flexible storage systems that perform very well in cloud environments. This is mandatory because for migration of virtual machines to happen between compute nodes we needed a common storage server which is accessible from all compute nodes. Whenever we initiate the creation of a virtual machine, the virtual machine will be created in a compute node along with its interface. If we use native Linux bridge, migration becomes quite impossible due to the generic driver, instead, we need to implement Open vSwitch. Open vSwitch is mandatory in the implementation of Openstack migration services which is the network concept beside Neutron and helps in advanced networking between machines. My future work will include looking more into the aspect of Openstack resource control and load migration that can be extended to support models of low system specification to support "controller provisioning" in which full applications can be developed at minimal requirements.

8.3 Final Remark

Working on the Openstack load balancing cloud was one the best DevOps experience through which I have learned more than enough to be able to duplicate knowledge in a real development environment. Distinguish various
load balancing algorithm and compared their performances amidst all the required specification a full fledge Infrastructure as a Service cloud environment.

A lot of recognition is given to Openstack for strengthening the public and private clouds. Openstack architecture could facilitate hybrid clouds, supporting the unionization of software, infrastructure and platform services so that network service can be accessed in a cloud federation. Keystone most demanded features in such a situation is its abilities to support a single sign-on access between intercloud and intracloud systems.
Chapter 9

References

http://www.dis.uniroma1.it/ciciani/DIDATTICA/ARCHITETTURE/SchedLB.pdf
International Journal of Computer Applications (0975 ? 8887) Volume 132?
No.2, December2015
http://kb.linuxvirtualserver.org/wiki/Weighted Round-Robin Scheduling
http://www.jscape.com/blog/load-balancing-algorithms
https://resources.oreilly.com/examples/9781785287725
https://docs.openstack.org/horizon/latest/install/from-source.html
Openstack Installation: https://docs.openstack.org/install-guide/
LBAAS Installation: https://docs.openstack.org/ocata/networking-guide/config-lbaas.html
HA and keepalived: https://www.digitalocean.com/community/tutorials/how-to-set-up
-highlyavailable-haproxy-servers-with-keepalived-and-floating-ips-on-ubuntu-14-04
Rabbitmq Cluster: http://www.serverlab.ca/tutorials/linux/message-queue/
ubuntu-14-04-rabbitmq-clustering/
Openstack Documentation: docs.openstack.org
Openstack Documentation Wiki: wiki.openstack.org