PROJECT REPORT

ON

OPERATIONAL EFFICIENCY IN CLOUD: GREEN DEPLOYMENTS

Submitted by

HEMANI KATYAL
POOJA MAHAJAN
KAVITA KUSHARE
BASAWASHRI BIRADAR

in partial fulfillment for the award of the degree

of

Bachelor of Engineering
of
University of Pune

IN

INFORMATION TECHNOLOGY

CUMMINS COLLEGE OF ENGINEERING FOR WOMEN PUNE
2014-15

PROJECT REPORT

ON

OPERATIONAL EFFICIENCY IN CLOUD: GREEN DEPLOYMENTS

Submitted By

HEMANI KATYAL
POOJA MAHAJAN
KAVITA KUSHARE
BASAWASHRI BIRADAR

Guided by

Prof. SURAJ CHAVAN
(CUMMINS COLLEGE OF ENGINEERING FOR WOMEN)

AND

Mr. RAHUL NEMA
(IBM MENTOR)

INFORMATION TECHNOLOGY

CUMMINS COLLEGE OF ENGINEERING FOR WOMEN PUNE

UNIVERSITY OF PUNE

2014- 2015

INFORMATION TECHNOLOGY

Certificate

This is to certify that,

B80208525 :- Hemani Katyal
B80208537 :- Pooja Mahajan
B80208535 :- Kavita Kushare
B80208510 :- Basawashri Biradar

have successfully completed this project report entitled "**OPERATIONAL EFFICIENCY IN CLOUD: GREEN DEPLOYMENTS**", under my guidance in partial fulfillment of the requirements for the degree of Bachelor of Engineering in Department of Information Technology of University of Pune during the academic year 2014-15.

Date: -
Place: - Pune

Prof. Suraj Chavan
Guide

Prof. Madhura Tokekar
HOD

Dr. Madhuri Khambete
Principal

ACKNOWLEDGEMENT

We take this opportunity to thank our project guides Prof. Suraj Chavan and Mr Rahul Nema and Head of the Department Prof. Madhura Tokekar for their valuable guidance and for providing all the necessary facilities, which were indispensable in the completion of this project.

We are also thankful to all the staff members of the Department of Information Technology of Cummins College of Engineering, Pune for their valuable time, support, comments, suggestions and persuasion.

We would further like to thank the institute for providing the required facilities, Internet access and important books.

Last but not the least we would take an opportunity to thank all the open source communities and forums for their services, which were indispensable in the completion of the project.

Hemani Katyal
Pooja Mahajan
Kavita Kushare
Basawashri Biradar

ABSTRACT

Cloud computing is gaining importance day-by day. Large number of enterprises and individuals are opting for cloud computing services. Thousands of servers have been employed worldwide to cater to the needs of customers for computing services by big organizations like Amazon, Microsoft, IBM and Google.

Cloud promises to bring a lot of Operational Efficiency by consolidation, virtualization and standardization. But most of the solutions today even lag putting energy efficient solution out to the end users. Energy consumed by servers and in cooling data centers of cloud system is a costly affair. Even today if there is a single VM on the host and it is up and running for long duration, it wastes lot of energy, making it more difficult to manage then it was when we only had physical servers.

The idea here is to use mix of analytics, cloud and monitoring tools (energy monitoring) which will decide the placements of the VM's depending on their utilization and impact on energy. Then start, stop and migrate them as per usage pattern and energy optimization needs.

Through this project we show a small scale simulation of Energy Optimization Model of the Cloud comprising of a Controller node and Compute nodes and how it can be extrapolated to fulfil the energy optimization needs of large datacenters with multiple servers and clients.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PART I

REQUIREMENT GATHERING AND ANALYSIS

The objective of this phase is to conduct a preliminary analysis, define project goals and objectives, gather and interpret facts, scrutinize the existing system to identify pros and cons of the current system in-place, diagnose problems and recommend improvements to the system. Analyze end-user information needs and describe benefits.

In the subsequent chapters in this part, the requirements of the proposed system are formulated. The aim and the objectives of the system are described and the need for the system is explored.

A chapter showing the Literature Survey done is included to highlight work done by different people in this domain.

Also the latter chapters introduce the technologies and the tools that have been used to develop the system.

CHAPTER 1

INTRODUCTION

This chapter introduces the concept of Data Centers and Cloud Computing. It talks about the vision and potential of cloud computing, and the challenges of using this technology. The aim and objectives of the project are stated and towards the end of this chapter the question "Who the System is for ? " is also answered in the section End Users of the System.

1.1 DATA CENTERS AND CLOUD COMPUTING: VISION AND POTENTIAL

The model referred to as utility computing, or recently as Cloud computing is one where users access services based on their requirements without regard to where the services are hosted. The latter term denotes the infrastructure as a "Cloud" from which businesses and users can access applications as services from anywhere in the world on demand.

Cloud computing provides users with access to a shared collection of computing resources: networks for transfer, servers for storage, and applications or services for completing tasks.

The compelling features of cloud are:

- On-demand self-service: Users can automatically provision needed computing capabilities, such as server time and network storage, without requiring human interaction with each service provider.

- Network access: Many computing capabilities are available over the network. Many different devices are allowed access through standardized mechanisms.

- Resource pooling: Multiple users can access clouds that serve other consumers according to demand.

- Elasticity: Provisioning is rapid and scales out or is based on need.

- Metered or measured service: Cloud systems can optimize and control resource use at the level that is appropriate for the service. Services include storage, processing, bandwidth, and active user accounts. Monitoring and reporting of resource usage provides transparency for both the provider and consumer of the utilized service.

Hence, Cloud computing can be classified as a new paradigm for the dynamic provisioning of computing services supported by state-of-the-art data centers that usually employ Virtual Machine (VM) technologies for consolidation and environment isolation purposes. Many computing service providers including Google, Microsoft, Yahoo, and

IBM are rapidly deploying data centers in various locations around the world to deliver cloud computing services.
Cloud computing delivers infrastructure, platform, and software (applications) as services, which are made available to consumers as subscription-based services under the pay-as-you-go model. In industry these services are referred to as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) respectively.

- Software-as-a-Service (SaaS): Provides the consumer the ability to use the software in a cloud environment, such as web-based email for example.

- Platform-as-a-Service (PaaS): Provides the consumer the ability to deploy applications through a programming language or tools supported by the cloud platform provider. An example of Platform-as-a-service is an Eclipse/Java programming platform provided with no downloads required.

- Infrastructure-as-a-Service (IaaS): Provides infrastructure such as computer instances, network connections, and storage so that people can run any software or operating system.

Terms such as public cloud or private cloud refer to the deployment model for the cloud. A private cloud operates for a single organization, but can be managed on-premise or off-premise. A public cloud has an infrastructure that is available to the general public or a large industry group and is likely owned by a cloud services company.

Clouds can also be described as hybrid. A hybrid cloud can be a deployment model, as a composition of both public and private clouds, or a hybrid model for cloud computing may involve both virtual and physical servers.

Cloud computing can help with large-scale computing needs or can lead to consolidation efforts by virtualizing servers to make more use of existing hardware and potentially release old hardware from service. Cloud computing is also used for collaboration because of its high availability through networked computers. Productivity suites for word processing, number crunching, and email communications, and more are also available through cloud computing. Cloud computing also avails additional storage to the cloud user, avoiding the need for additional hard drives on each user's desktop and enabling access to huge data storage capacity online in the cloud.

Clouds aim to drive the design of the next generation data centers by architecting them as networks of virtual services (hardware, database, user-interface, application logic) so that users can access and deploy applications from anywhere in the world on demand at competitive costs depending on their QoS (Quality of Service) requirements. Developers with innovative ideas for new Internet services no longer require large capital outlays in hardware to deploy their service or human expense to operate it.

Cloud computing offers significant benefits to IT companies by freeing them from the low-level task of setting up basic hardware and software infrastructures and thus enabling focus on innovation and creating business value for their services.

1.2 CHALLENGES AND REQUIREMENTS

Modern data centers, operating under the Cloud computing model are hosting a variety of applications ranging from those that run for a few seconds (e.g. serving requests of web applications such as e-commerce and social network portals with transient workloads) to those that run for longer periods of time (e.g. simulations or large data set processing) on shared hardware platforms [1].

The need to manage multiple applications in a data center creates the challenge of on-demand resource provisioning and allocation in response to time-varying workloads. Normally, data center resources are statically allocated to applications, based on peak load characteristics, in order to maintain isolation and provide performance guarantees.

Until recently, high performance has been the sole concern in data center deployments and this demand has been fulfilled without paying much attention to energy consumption.

The average data center consumes as much energy as 25,000 households. As energy costs are increasing while availability dwindles, there is a need to shift focus from optimizing data center resource management for pure performance to optimizing for energy efficiency.

Data centers are not only expensive to maintain, but also unfriendly to the environment. High energy costs and huge carbon footprints are incurred due to massive amounts of electricity needed to power and cool numerous servers hosted in these data centers.

Cloud service providers need to adopt measures to ensure that their profit margin is not dramatically reduced due to high energy costs. For instance, Google, Microsoft, and Yahoo are building large data centers in barren desert land surrounding the Columbia River, USA to exploit cheap and reliable hydroelectric power.

Lowering the energy usage of data centers is a challenging and complex issue because computing applications and data are growing so quickly that increasingly larger servers and disks are needed to process them fast enough within the required time period.

Green Cloud computing is envisioned to achieve not only efficient processing and utilization of computing infrastructure, but also minimize energy consumption. This is essential for ensuring that the future growth of Cloud computing is sustainable. Otherwise, Cloud computing with increasingly pervasive front-end client devices interacting with back-end data centers will cause an enormous escalation of energy usage. To address this problem, data center resources need to be managed in an energy-efficient manner to drive Green Cloud computing.

1.3 AIM AND OBJECTIVE

The current state-of-the-art in Cloud infrastructure has no or limited consideration for supporting energy-aware service allocation that meets QoS needs of consumers and minimizes energy costs.

The main objective of this project is to initiate research and development of energy-aware resource allocation mechanisms for data centers so that Cloud computing can be a more sustainable eco-friendly mainstream technology to drive commercial, scientific, and technological advancement for future generations.

Specifically, the aim of project is to:

- Design a Cloud Infrastructure for the purpose of study and research.
- Define an architectural framework and principles for energy-efficient Cloud computing in the above Infrastructure.
- Monitor the workload on the VM’s.
- Implement algorithms for energy-efficient mixing and mapping of VMs to suitable Cloud resources through dynamic consolidation of VM resource partitions during migrations.

1.4 END USERS OF THE SYSTEM

The proposed system would be of use for companies like IBM that provide Cloud services to the consumers.

The end user of the system would be a Cloud Administrator or Systems Integrator, who wants to improve the utilization of the data center’s resources and minimize the price of the service provided to the consumers by reducing the operating costs through the reduced energy consumption. And in turn also decrease the carbon dioxide emissions into the environment by reducing energy consumption by the data center’s resources.

CHAPTER 2

SOFTWARE DEVELOPMEMT LIFE CYCLE

Software follows a Development Lifecycle Model for smooth and effective development process. This chapter is dedicated to the understanding of the different phases software goes through. A special mention of the Agile Development Model is made as it is used in the development of the system described.

2.1 INTRODUCTION TO SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

The software development life cycle (SDLC) is a framework defining tasks performed at each step in the software development process. SDLC is a structure followed by a development team within the software organization. It consists of a detailed plan describing how to develop, maintain and replace specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

There are various software development approaches defined and designed which are used/employed during development process of software, these approaches are also referred as “Software Development Process Models” (e.g. Waterfall model, incremental model, V-model, iterative model, etc.). Each process model follows a particular life cycle in order to ensure success in process of software development.

Software life cycle models describe phases of the software life cycle and the order in which these phases are executed. Each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced according to the design which is called development phase. After coding and development the testing verifies the deliverable of the implementation phase against requirements.

There are following six phases in Software Development Life Cycle:

1. Requirement gathering and analysis
2. Design
3. Implementation or coding
4. Testing
5. Deployment
6. Maintenance

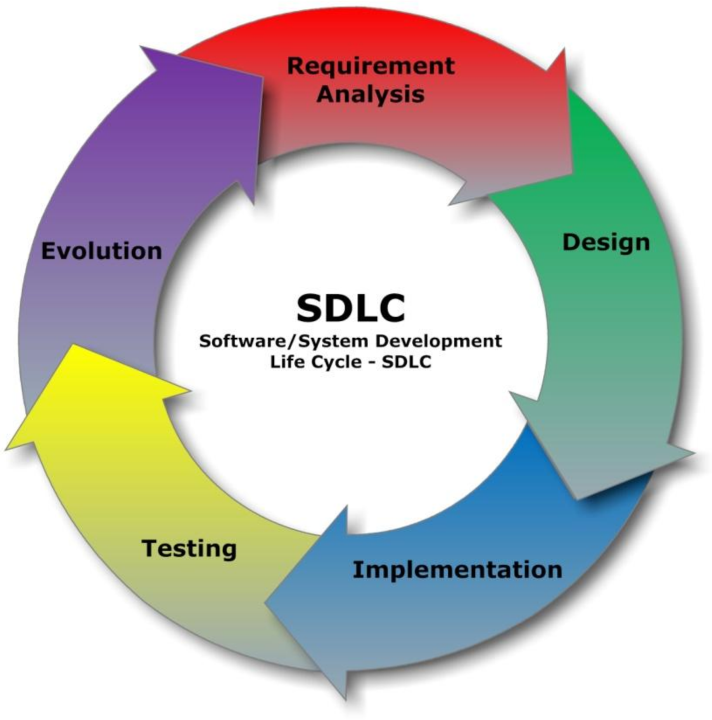


Figure 2.1 Software Development Life Cycle

1) **Requirement gathering and analysis:** Business requirements are gathered in this phase. This phase is the main focus of the project managers and stake holders. Meetings with managers, stake holders and users are held in order to determine the requirements like; Who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system? These are general questions that get answered during a requirements gathering phase. After requirement gathering these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be developed is also studied.

   Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model.

2) **Design:** In this phase the system and software design is prepared from the requirement specifications which were studied in the first phase. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

3) **Implementation / Coding:** On receiving system design documents, the work is divided in modules/units and actual coding is started. Since, in this phase the code is produced so it is the main focus for the developer. This is the longest phase of the software development life cycle.

4) **Testing:** After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During this phase unit testing, integration testing, system testing, acceptance testing are done.

5) **Deployment:** After successful testing the product is delivered / deployed to the customer for their use.

6) **Maintenance:** Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance.

2.2 AGILE SOFTWARE DEVELOPMENT LIFECYCLE

Agile software development is a group of software development methods in which requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development, early delivery, continuous improvement and encourages rapid and flexible response to change.

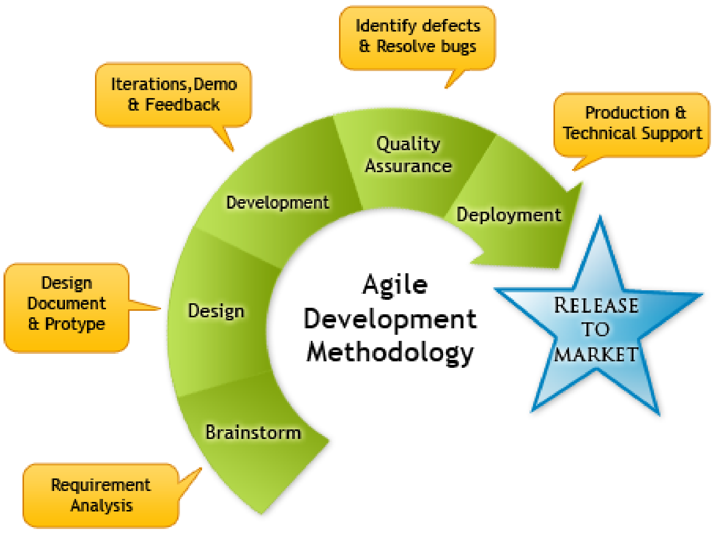

Identify defects & Resolve bugs
Iterations,Demo & Feedback
Production & Technical Support
Quality Assurance
Development
Deployment
Design Document & Protype
Design
Agile Development Methodology
RELEASE TO MARKET
Brainstorm
Requirement Analysis

Figure 2.2 Agile Development Model

2.2.1 FEATURES OF AGILE MODEL

Following are the compelling features of the Agile Development Model which make it stand out among the other models and the reason for its use in this project:

- **Iterative:** Entire application is distributed in incremental units called as iteration. Development time of each iteration is small (couple of weeks), fixed and strictly adhered to. Every Iteration is a mini increment of the functionality and is build on top of previous iteration.

- **Active Customer Involvement:** There is lot of client involvement and face-to-face interaction. Every iteration is tested and approved by client. The feedback obtained is implemented in subsequent iterations; thus minimizing risk and ensuring higher client satisfaction.

- **Feature Driven:** More emphasis is on providing the required features in the application. 80/20 principle is applied to decide the 20 percent features which would be used 80 percent of the time.

- **Fixed Time:** Each iteration has a fixed time span in which it is delivered.

- **Priority Based Delivery:** Features are prioritized depending on customer need, development risk etc. High priority features are developed first. After every iteration, the project priorities are re-evaluated.

- **Adaptive:** The methodology in general is very adaptive, so that the application that is developed can cater to inflow of new requirements throughout its development. Goal is not to remove the uncertainty in the very beginning, but is to adapt to the changing needs.

- **Empowered Teams:** The project teams are generally small and have lot of interaction and communication. Since entire team is actively involved, team is empowered to take decisions. No separate team to manage project.

- **People Centric:** More emphasis is on using the adequately skilled people to do the development than on following the processes. The documentation and other non-development activities are minimized and more time is devoted to development and testing.

- **Rapid Development:** Generally the development is done rapidly using light weight development technologies.

- **More Disciplined:** Being rapid, everything has to be delivered correctly first time. So the process involves lot of team and self discipline Thus, it requires highly skilled and organized team members.

- **Simplicity:** Emphasis is on keeping things as simple as possible and being open to change.

CHAPTER 3

LITERATURE SURVEY

This chapter highlights the work done in the field of Energy Efficient Cloud Computing. It explains why so far such energy efficient system could not be deployed. It further explores the research done and contributions made by different researchers and authors in this domain.

3.1 EXISTING SYSTEMS

Current state-of-the-art Cloud infrastructure such as Amazon EC2 neither support Energy efficient resource allocation that considers consumer preference for energy saving schemes, nor utilize sophisticated economic models to set the right incentives for consumers to reveal information about their service demand accurately. Hence, providers are not able to foster essential information exchange with consumers and therefore cannot attain efficient service allocation, which meets consumer needs and expectations with regards to their energy saving preference for Green Cloud computing.

Market-based resource management has been proposed by researchers to manage allocations of computing resources since it is effectively utilized in the field of economics to regulate supply and demand of limited goods. With Cloud computing emphasizing on a pay-per-use economic model, there is a high potential to apply market-based resource management techniques that justify the monetary return and opportunity cost of resource allocation according to consumer QoS expectations and baseline energy costs. There are many proposed systems utilization market-based resource management for various computing areas, but none of these systems focus on the problem of energy efficiency in addition to maximizing profit. Thus, these systems are not able to support Green Cloud computing. They do not offer economic incentives that can encourage, discourage, or vary quality expectations of service requests from consumers with respect to energy saving schemes.

Existing energy-efficient resource allocation solutions proposed for various computing systems cannot be implemented for Green Cloud computing. This is because they only focus on minimizing energy consumption or their costs, and do not consider dynamic service requirements of consumers that can be changed on demand in Cloud computing environments. Hence, they do not emphasize autonomic energy-aware resource management mechanisms and policies exploiting VM resource allocation which is the main operating technology in Cloud Computing.

Thus, so far there has been no ideal system for energy efficiency implemented and used anywhere considering the trade off between performance and energy efficiency.

3.2 SIMILAR RESEARCH WORK

Despite the large volume of research published on this topic, there are very few open-source software systems implementing Energy Efficient Cloud Solutions.

Various researchers are working in energy efficient computing and architectural principles for energy efficient management of clouds, energy efficient VMs allocation policies and scheduling algorithms considering QoS expectations and power usage characteristics of the devices.

One of the techniques proposed is called dynamic self ballooning, where a driver runs in the guest VM, continuously reclaiming free and unused memory and giving it back to the hypervisor. This technique during migration reduces the amount of memory that is to be sent of the network since the reclaimed and unused memory is not transferred. [4]

Another technique applied to minimize power consumption is concentrating the workload to the minimum of physical nodes and switching idle nodes off. [5]

Yet another mechanism proposed is to adjust the system voltage based on the CPU utilization, and migrating tasks in a heavy loaded machine to idle machines, so as to improve the resource utilization and reduce the energy consumption. [6]

Anton Beloglazov, a PhD Student of "The University of Melbourne" has made major contributions in this domain.

He has conducted exhaustive research and published numerous papers on this topic within the Cloud Computing and Distributed Systems (CLOUDS) Laboratory at the University of Melbourne.

He has proposed the technique of Dynamic VM consolidation in OpenStack Clouds and presented OpenStack Neat as an extension to OpenStack implementing dynamic consolidation of Virtual Machines (VMs) using live migration. The major objective of dynamic VM consolidation is to improve the utilization of physical resources and reduce energy consumption by re-allocating VMs using live migration according to their real-time resource demand and switching idle hosts to the sleep mode.

For example, assume that two VMs are placed on two different hosts, but the combined resource capacity required by the VMs to serve the current load can be provided by just one of the hosts. Then, one of the VMs can be migrated to the host serving the other VM, and the idle host can be switched to a low power mode to save energy. When the resource demand of either of the VMs increases, they get deconsolidated to avoid performance degradation. This process is dynamically managed by OpenStack Neat.

In general, the problem of dynamic VM consolidation can be split into 4 sub-problems:

1. Deciding when a host is considered to be underloaded, so that all the VMs should be migrated from it, and the host should be switched to a low power mode, such as the sleep mode.

2. Deciding when a host is considered to be overloaded, so that some VMs should be migrated from the host to other hosts to avoid performance degradation.

3. Selecting VMs to migrate from an overloaded host out of the full set of the VMs currently served by the host.

4. Placing VMs selected for migration to other active or re-activated hosts.

The aim of the OpenStack Neat project is to provide an extensible framework for dynamic consolidation of VMs based on the OpenStack platform. The framework provides an infrastructure enabling the interaction of components implementing the 4 decision-making algorithms listed above. The framework allows configuration-driven switching of different implementations of the decision-making algorithms. [2][3]

3.3 CONCLUSION

Recent Research papers reviewed reveal that Live Migration feature of Virtualization Technology has ample scope to minimize energy consumption in data centers.

This thesis tackles the research challenges in relation to energy-efficient distributed dynamic VM consolidation in IaaS environments. In particular, the following research problems have been noticed:

- **How to define workload-independent QoS requirements:** Since multiple applications of various types can coexist in an IaaS environment and share physical resources, it is necessary to derive a workload independent QoS metric that can be used to define system-wide QoS constraints.

- **When to migrate VMs:** Dynamic VM consolidation comprises two basic processes: (1) migrating VMs from overloaded servers to avoid performance degradation; and (2) migrating VMs from underloaded servers to improve the utilization of resources and minimize energy consumption. A crucial decision that must be made in both situations is determining the best time to migrate VMs to minimize energy consumption, while satisfying the defined QoS constraints.

- **Which VMs to migrate:** Once a decision to migrate VMs from a server is made, it is required to select one or more VMs from the full set of VMs allocated to the server, which must be reallocated to other servers. The problem consists in determining the best subset of VMs to migrate that will provide the most beneficial system reconfiguration.

- **Where to migrate the VMs selected for migration**: Determining the best placement of new VMs or the VMs selected for migration to other servers is another essential aspect that influences the quality of VM consolidation and energy consumption by the system.

- **Which physical nodes to switch on/off:** To save energy, VM consolidation should be combined with dynamic switching of the power states of nodes, which addresses the problem of narrow power ranges of servers by eliminating power consumption in the idle state. To optimize energy consumption by the system and avoid violations of the QoS requirements, it is necessary to efficiently determine when and which physical nodes should be deactivated to save energy, or reactivated to handle increases in the demand for resources.

- **How to design distributed dynamic VM consolidation algorithms:** To provide scalability and eliminate single points of failure, it is necessary to design dynamic VM consolidation algorithms in a distributed manner. The problem is that traditionally global resources management algorithms are centralized. Therefore, a new approach is required to make the dynamic VM consolidation system distributed.

Hence, it is clearly evident that at present there is no such system for Energy Efficient Resource allocation available and large volumes of research is currently in progress in this area. Also observations show that one of the most common techniques of achieving energy efficiency and operational efficiency in cloud is through VM Migration and Consolidation. Concerted efforts are being taken to come up with an energy efficient cloud solution.

CHAPTER 4

CLOUD COMPUTNG TECHNOLOGIES

This chapter is devoted to the understanding of different technologies associated with Cloud Computing such as Virtualization, Load Balancing and Live Migration which will be required in designing of the proposed system.

4.1 VIRTUALIZATION

Virtualization is a general and ambiguous term that typically means to run multiple instances of something that was intended to run as a single instance. At its simplest level, virtualization allows you, virtually and cost effectively, to have two or more virtual computing environments, running different operating system and application on one piece of hardware.

For example, with virtualization, you can have both a Linux virtual machine and a Microsoft Windows virtual machine on one system.

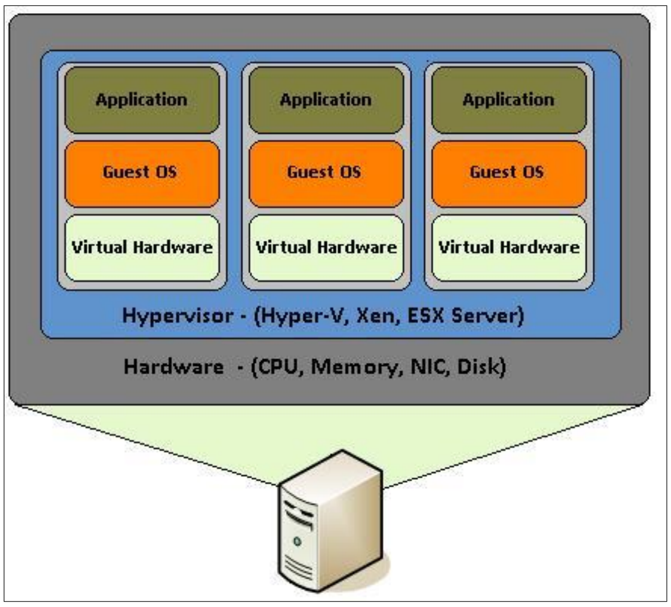


Figure 4.1 Hardware Virtualization

The concept can be shown in Figure 4.1 above. Here physical hardware is the bottom most part which is nothing but the part on which the operating system is based. Then the Hypervisor layer which is the most important part utilized for virtualization.

4.1.1 HYPERVISOR

A hypervisor is a virtualization platform that enables you to run multiple operating systems on a single physical computer called the host computer. The main function of hypervisor is to provide isolated execution environment for each virtual machines and the underlined hardware resources on the physical computer. Hypervisor comes in several different flavours.

They can be categorized, for example, by type that is, by whether they run directly on the physical hardware or within an operating system environment.

Type-1 Hypervisor

Type 1 hypervisors run directly on the host's hardware to control the hardware and to manage guest operating systems. A guest operating-system thus runs on another level above the hypervisor. Since, Type-1 hypervisor run directly on bare metal instead of within an operating system environment, they can generally provide the best performance, availability, and security of any form of hypervisor.

Some examples are VMware ESX, Xen, and Microsoft Hyper-V.

Type-2 Hypervisor

Type 2 (or hosted) hypervisors run within a conventional operating-system environment. With the hypervisor layer as a distinct second software level, guest operating-systems run at the third level above the hardware. This type of virtualization is typically referred as hosted virtualization. The operating system running in virtual machines on Type-2 hypervisor platform are one level separated from underlined physical hardware that guest operating system on Type-1 hypervisor platform. This extra level of separation between virtual machine and hardware results in a performance hit being incurred on Type-2 hypervisor platform, and the effect of this added overhead limits the virtual machines you can realistically run on Type-2 platform.

Some examples are VMware Workstation, VMware Fusion, MED-V, Windows Virtual PC, VirtualBox, Parallels, MokaFive, KVM etc.

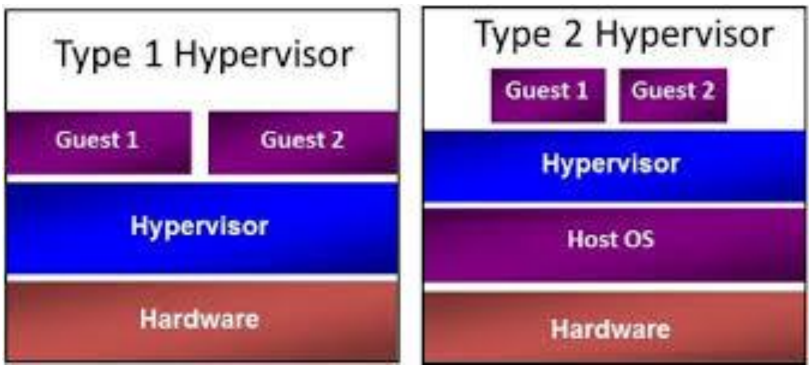


Figure 4.2 Types of Hypervisors

4.1.2 ADVANTAGES OF VIRTUALIZATION

Listed below are some of the advantages of Virtualization:

1. We can test new operating system and/or software without altering our current system in a safe and separated machine.

2. We can run different operating system at a time on same computer.

3. On servers it allows you to provide very different services with a single machine and save money.

4. Reduce the number of physical servers.

5. Reduce administrative overhead because servers can be administered from a single console.

4.2 LOAD BALANCING

Load Balancing is an essential part of any distributed computing system. It is a technique used to distribute workload evenly across two or more computers, network links, CPUs or other resources in order to get optimal resource utilization, maximize throughput, minimize response time and avoid overload. The load distribution among servers in a network has to be performed in a manner such that no server spends idle CPU time while some other server on the network is loaded heavily.

In a cloud computing environment where user tasks are carried out on virtual machines as opposed to physical servers (in case of a normal distributed computing system), the system load has to be efficiently distributed among the virtual machines as well as maintaining efficient distribution of these virtual machines on various physical servers in the cloud. The key idea of load distribution here is to effectively and efficiently utilize the infrastructure available in the cloud for processing user's requests.

An efficient load balancer greatly improves the efficiency and resource availability of the cloud. The load balancer is also responsible for handling issues like Application scalability, making appropriate database entries corresponding to client's requests which will be useful while billing the client. Load Balancing also plays a key role in the design and implementation of Energy Efficient Systems.

The load balancing problem is to handle efficient distribution of virtual machines on physical servers in the cloud. The Load Balancer is configured with the network information of all the servers in the cloud. Each host server is identified by its unique IP address assigned to it in the network. Each host server is assigned Weight which signifies the Load Balancer's current estimate of the System Load at that host server. Thus, the host server with the least Weight should be assigned the job at hand.

4.3 LIVE MIGRATION

Live Virtual machine Migration is a technique that migrates the entire OS and its associated application from one physical machine to another. The Virtual machines are migrated lively without disrupting the application running on it. The benefits of virtual machine migration include conservation of physical server energy, load balancing among the physical servers and failure tolerance in case of sudden failure. The different virtual machine migration techniques are as follows.

1. **Fault Tolerant Migration Techniques**

   Fault tolerance allows the virtual machines to continue its job even if any part of system fails. This technique migrates the virtual machine from one physical server to another physical server based upon the prediction of the failure occurred, fault tolerant migration technique is to improve the availability of physical server and avoids performance degradation of applications.

2. **Load Balancing Migration Techniques**

   The Load balancing migration technique aims to distribute load across the physical servers to improve the scalability of physical servers in cloud environment. Load balancing aids in minimizing the resource consumption, implementation of fail-over, enhancing scalability, avoiding bottlenecks and over provisioning of resources.

3. **Energy Efficient Migration Techniques**

   The power consumption of Data center is mainly based on the utilization of the servers and their cooling systems. The servers typically need up to 70 percentage of their maximum power consumption even at their low utilization level. Therefore, there is a need for migration techniques that conserves the energy of servers by optimum resource utilization.

CHAPTER 5

OPENSTACK

In order to design the Cloud Infrastructure, the project makes use of OpenStack as a tool. This chapter introduces the basics of OpenStack. The different components of Openstack along with the Conceptual Architecture of OpenStack is elaborated here.

5.1 OPENSTACK OVERVIEW

OpenStack is a global collaboration of developers and cloud computing technologists, producing the ubiquitous open source cloud computing platform for public and private clouds. The project aims for simple implementation, massive scalability, and a rich set of features. Cloud computing experts from around the world contribute to the project.

OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data center, all managed through a dashboard that gives administrators control while empowering users to provision resources through a web interface.

5.2 COMPONENTS OF OPENSTACK

OpenStack has a modular architecture with various code names for its components. OpenStack has several shared services that span the three pillars of compute, storage and networking, making it easier to implement and operate your cloud. These services including identity, image management and a web interface integrate the OpenStack components with each other as well as external systems to provide a unified experience for users as they interact with different cloud resources.

- **Compute (Nova)**

The OpenStack cloud operating system enables enterprises and service providers to offer on-demand computing resources, by provisioning and managing large networks of virtual machines. Compute resources are accessible via APIs for developers building cloud applications and via web interfaces for administrators and users. The compute architecture is designed to scale horizontally on standard hardware.

OpenStack Compute (Nova) is a cloud computing fabric controller (the main part of an IaaS system). It is written in Python and uses many external libraries such as Eventlet (for concurrent programming), Kombu (for AMQP communication), and SQLAlchemy (for database access). Nova's architecture is designed to scale horizontally on standard hardware with no proprietary hardware or software requirements and provide the ability to integrate with legacy systems and third party technologies. It is designed to manage and automate pools of computer resources and can work with widely available virtualization technologies, as well as bare metal and high-performance computing (HPC) configurations. KVM and XenServer are available choices for hypervisor technology, together with Hyper-V and Linux container technology such as LXC. In addition to different hypervisors, OpenStack runs on ARM.

- **Object Storage (Swift)**

In addition to traditional enterprise-class storage technology, many organizations now have a variety of storage needs with varying performance and price requirements. OpenStack has support for both Object Storage and Block Storage, with many deployment options for each depending on the use case.

OpenStack Object Storage (Swift) is a scalable redundant storage system. Objects and files are written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster. Storage clusters scale horizontally simply by adding new servers. Should a server or hard drive fail, OpenStack replicates its content from other active nodes to new locations in the cluster. Because OpenStack uses software logic to ensure data replication and distribution across different devices, inexpensive commodity hard drives and servers can be used.

Object Storage is ideal for cost effective, scale-out storage. It provides a fully distributed, API-accessible storage platform that can be integrated directly into applications or used for backup, archiving and data retention. Block Storage allows block devices to be exposed and connected to compute instances for expanded storage, better performance and integration with enterprise storage platforms, such as NetApp, Nexenta and SolidFire.

A few details on OpenStack’s Object Storage

- OpenStack provides redundant, scalable object storage using clusters of standardized servers capable of storing petabytes of data.

- Object Storage is not a traditional file system, but rather a distributed storage system for static data such as virtual machine images, photo storage, email storage, backups and archives. Having no central "brain" or master point of control provides greater scalability, redundancy and durability.

- Objects and files are written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster.

- Storage clusters scale horizontally simply by adding new servers. Should a server or hard drive fail, OpenStack replicates its content from other active nodes to new locations in the cluster. Because OpenStack uses software logic to ensure data replication and distribution across different devices, inexpensive commodity hard drives and servers can be used in lieu of more expensive equipment.

- **Block Storage (Cinder)**

OpenStack Block Storage (Cinder) provides persistent block level storage devices for use with OpenStack compute instances. The block storage system manages the creation, attaching and detaching of the block devices to servers. Block storage volumes are fully integrated into OpenStack Compute and the Dashboard allowing for cloud users to manage their own storage needs. In addition to local Linux server storage, it can use storage platforms including Ceph, CloudByte, Coraid, EMC (VMAX and VNX), GlusterFS, IBM Storage (Storwize family, SAN Volume Controller, and XIV Storage System), Linux LIO, NetApp, Nexenta, Scality, SolidFire and HP (Store Virtual and StoreServ 3Par families). Block storage is appropriate for performance sensitive scenarios such as database storage, expandable file systems, or providing a server with access to raw block level storage. Snapshot management provides powerful functionality for backing up data stored on block storage volumes. Snapshots can be restored or used to create a new block storage volume.

A few points on OpenStack Block Storage:

- OpenStack provides persistent block level storage devices for use with OpenStack compute instances.

- The block storage system manages the creation, attaching and detaching of the block devices to servers. Block storage volumes are fully integrated into OpenStack Compute and the Dashboard allowing for cloud users to manage their own storage needs.

- In addition to using simple Linux server storage, it has unified storage support for numerous storage plat- forms including Ceph, NetApp, Nexenta, SolidFire, and Zadara.

- Block storage is appropriate for performance sensitive scenarios such as database storage, expandable file systems, or providing a server with access to raw block level storage.

- Snapshot management provides powerful functionality for backing up data stored on block storage vol- umes. Snapshots can be restored or used to create a new block storage volume.

- **Networking (Neutron)**

Today's data center networks contain more devices than ever before. From servers, network equipment, storage systems and security appliances, many of which are further divided into virtual machines and virtual networks. The number of IP addresses, routing configurations and security rules can quickly grow into the millions. Traditional network management techniques fall short of providing a truly scalable, automated approach to managing these next-generation networks. At the same time, users expect more control and flexibility with quicker provisioning.

OpenStack Networking is a pluggable, scalable and API-driven system for managing networks and IP addresses. Like other aspects of the cloud operating system, it can be used by administrators and users to increase the value of existing data center assets. OpenStack Networking ensures the network will not be the bottleneck or limiting factor in a cloud deployment and gives users real self-service, even over their network configurations.

OpenStack Networking (Neutron, formerly Quantum) is a system for managing networks and IP addresses. Like other aspects of the cloud operating system, it can be used by administrators and users to increase the value of existing data center assets. OpenStack Networking ensures the network will not be the bottleneck or limiting factor in a cloud deployment and gives users real self-service, even over their network configurations.

OpenStack Neutron provides networking models for different applications or user groups. Standard models include flat networks or VLANs for separation of servers and traffic. OpenStack Networking manages IP ad- dresses, allowing for dedicated static IPs or DHCP. Floating IPs allow traffic to be dynamically re routed to any of your compute resources, which allows you to redirect traffic during maintenance or in the case of failure. Users can create their own networks, control traffic and connect servers and devices to one or more networks. Administrators can take advantage of software-defined networking (SDN) technology like OpenFlow to allow for high levels of multi-tenancy and massive scale. OpenStack Networking has an extension framework allow- ing additional network services, such as intrusion detection systems (IDS), load balancing, firewalls and virtual private networks (VPN) to be deployed and managed.

Networking Capabilities

- OpenStack provides flexible networking models to suit the needs of different applications or user groups. Standard models include flat networks or VLANs for separation of servers and traffic.

- OpenStack Networking manages IP addresses, allowing for dedicated static IPs or DHCP. Floating IPs allow traffic to be dynamically re-routed to any of your compute resources, which allows you to redirect traffic during maintenance or in the case of failure.

- Users can create their own networks, control traffic and connect servers and devices to one or more networks.

- The pluggable backend architecture lets users take advantage of commodity gear or advanced networking services from supported vendors.

- Administrators can take advantage of software-defined networking (SDN) technology like OpenFlow to allow for high levels of multi-tenancy and massive scale.

- OpenStack Networking has an extension framework allowing additional network services, such as intrusion detection systems (IDS), load balancing, firewalls and virtual private networks (VPN) to be deployed and managed.

- **Dashboard(Horizon)**

OpenStack Dashboard (Horizon) provides administrators and users a graphical interface to access, provision and automate cloud-based resources. The design allows for third party products and services, such as billing, monitoring and additional management tools. Service providers and other commercial vendors can customize the dashboard with their own brand.

The dashboard is just one way to interact with OpenStack resources. Developers can automate access or build tools to manage their resources using the native OpenStack API or the EC2 compatibility API.

- **Identity Service (Keystone)**

OpenStack Identity (Keystone) provides a central directory of users mapped to the OpenStack services they can access. It acts as a common authentication system across the cloud operating system and can integrate with existing backend directory services like LDAP. It supports multiple forms of authentication including stan- dard username and password credentials, token-based systems, and Amazon Web Services log in credentials such as those used for EC2.

Additionally, the catalog provides a query-able list of all of the services deployed in an OpenStack cloud in a single registry. Users and third-party tools can programmatically determine which resources they can access.

The OpenStack Identity Service enables administrators to:

- Configure centralized policies across users and systems

- Create users and tenants and define permissions for compute, storage, and networking resources by using role-based access control (RBAC) features
- Integrate with an existing directory, like LDAP, to provide a single source of authentication across the enterprise

The OpenStack Identity Service enables users to:

- List the services to which they have access
- Make API requests
- Log into the web dashboard to create resources owned by their account

- **Image Service (Glance)**

OpenStack Image Service (Glance) provides discovery, registration and delivery services for disk and server images. Stored images can be used as a template. They can also be used to store and catalog an unlimited number of backups. The Image Service can store disk and server images in a variety of back-ends, including OpenStack Object Storage. The Image Service API provides a standard REST interface for querying information about disk images and lets clients stream the images to new servers.

Capabilities of the Image Service include:

- Administrators can create base templates from which their users can start new compute instances.

- Users can choose from available images, or create their own from existing servers

- Snapshots can also be stored in the Image Service so that virtual machines can be backed up quickly

A multi-format image registry, the image service allows uploads of private and public images in a variety of formats, including:

- Raw
- Machine (kernel/ramdisk outside of image, also known as AMI)
- VHD (Hyper-V)
- VDI (VirtualBox)
- qcow2 (Qemu/KVM)
- VMDK (VMware)
- OVF (VMware, others)

The table below summarizes OpenStack services described above.

SERVICE	PROJECT NAME	DESCRPTION
Dashboard	Horizon	Provides a web-based self-service portal to intereact with underlying OpenStack Services, such as launching an instance, assigning IP addresses and configuring access control.
Compute	Nova	Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decommisioning of virtual machines on demand.
Networking	Neutron	Enables Network-Connectivity-as-a-Service for other OpenStack services such as OpenStack Compute. Provides an API for users to define networks an the attachments into them. Has a pluggable architecture that supports many popular networking vendors and technologies.
Storage		
Object Storage	Swift	Stores and retrieves arbitrary unstructured data objects via a RESTful, HTTP based API. It is highly fault tolerant with its data replication and scale out architecture. Its implementation is not like a file server with mountable directories.
Block Storage	Cinder	Provides persistent block storage to running instances. Its pluggable driver architecture facilitates the creation and management of block storage devices.
Shared Services		
Identity Service	Keystone	Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.
Image Service	Glance	Stores and retrieves virtual machine disk images. OpenStack Compute makes use of this during instance provisioning.
Telemetry	Ceilometer	Monitors and meters the OpenStack cloud for billing, benchmarking, scalability and statistical purposes.
Higher-level Services		
Orchestration	Heat	Orchestrates multiple composite cloud applications by using either the native HOT template format or the AWS CloudFormation template format, through both OpenStack-native REST API and CloudFormation-compatible Query API.
Database Service	Trove	Provides scalable and reliable Cloud Database-as-a-Service functionality for both relational and non-relational database engines.

Table 5.1 OpenStack Services

5.3 CONCEPTUAL ARCHITECTURE

The OpenStack project as a whole is designed to deliver a massively scalable cloud operating system. To achieve this, each of the constituent services are designed to work together to provide a complete Infrastructure-as-a-Service (IaaS). This integration is facilitated through public application programming interfaces (APIs) that each service offers (and in turn can consume). While these APIs allow each of the services to use another service, it also allows an implementer to switch out any service as long as they maintain the API. These are (mostly) the same APIs that are available to end users of the cloud.

Conceptually, you can picture the relationships between the services as so:

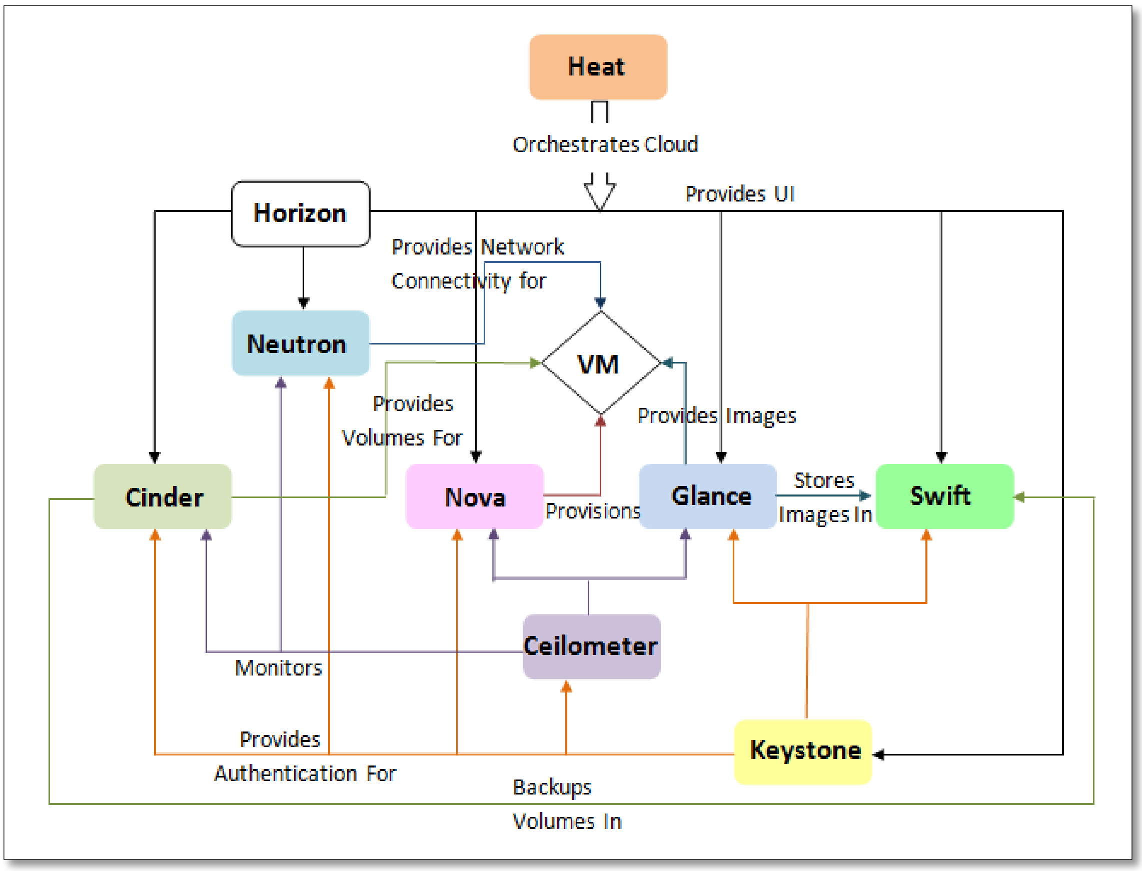


Figure 5.1 Conceptual Architecture of OpenStack

- Dashboard ("Horizon") provides a web front end to the other OpenStack services.
- Compute ("Nova") stores and retrieves virtual disks ("images") and associate metadata in Image ("Glance").
- Network ("Quantum") provides virtual networking for Compute.
- Block Storage ("Cinder") provides storage volumes for Compute.
- Image ("Glance") can store the actual virtual disk files in the Object Store("Swift").
- All the services authenticate with Identity ("Keystone").

CHAPTER 6

NAGIOS

This chapter describes Nagios, the Energy Monitoring Tool used in this project. It describes the working, architecture and features of Nagios.

6.1 NAGIOS OVERVIEW

Nagios is one of the most popular computer networks monitoring software application. It is developed by Ethan Galstad, as an open source, Unix-based enterprise monitoring package with a web-based front-end or console. It provides monitoring of network services (SMTP, POP3, HTTP, FTP, SNMP, SSH) and host resources (processor load, disk usage, system logs) and essentially any device or service that have address and can be contacted via TCP/IP. It can monitor host running Microsoft Windows, Unix/Linux, Novell Netware, and other operating system.

With Nagios, own service check can be created depending on needs by developing simple plug-ins by using tools of choice (shell scripts, C++, Perl, Ruby, Python, PHP, C#, etc.). In the event of service or host problems, Nagios has contact notification in its configuration to handle and resolve such events either via email, pager, or user-defined method.

Considering the aspect of scanning the monitored devices, Nagios uses four states to describe status: OK, WARNING, CRITICAL, UNKNOWN rather than monitoring value or graphs that may be ignored when it needed a quick attention. Also, Nagios gives report of number of services that are up and running in both warning state and critical state with aid of its friendly GUI for service status display. This presents a good overview of infrastructure status.

6.2 NAGIOS ARCHITECTURE

In order to get clearer picture of how Nagios works, it is necessary to look into its architecture. Its architecture is based or built on a server/client model. This architecture can be seen in the figure below. This depicts a Nagios server running on a host, and plug-ins running on the server and all other remote hosts to be monitored. This plug-ins sends information to the server, which in turn displays them on GUI.

In addition, Nagios can be said to be composed of three parts namely; A scheduler, a GUI, and the plug-ins. These are described as below:

1) **A Scheduler:** is a server part of Nagios that checks plug-ins at regular interval and do some actions according to the results from the checked plug-ins.

2) **A GUI:** is the interface of Nagios that is displayed in webpage generated by the Common Gateway Interface (CGI). The interface can display configurations, alerts, state buttons (green, OK/red, Error), MRTG graphs etc.

3) **The plug-ins:** They are configurable by the user. They check a service and return a result to the Nagios server in order to take corresponding actions.

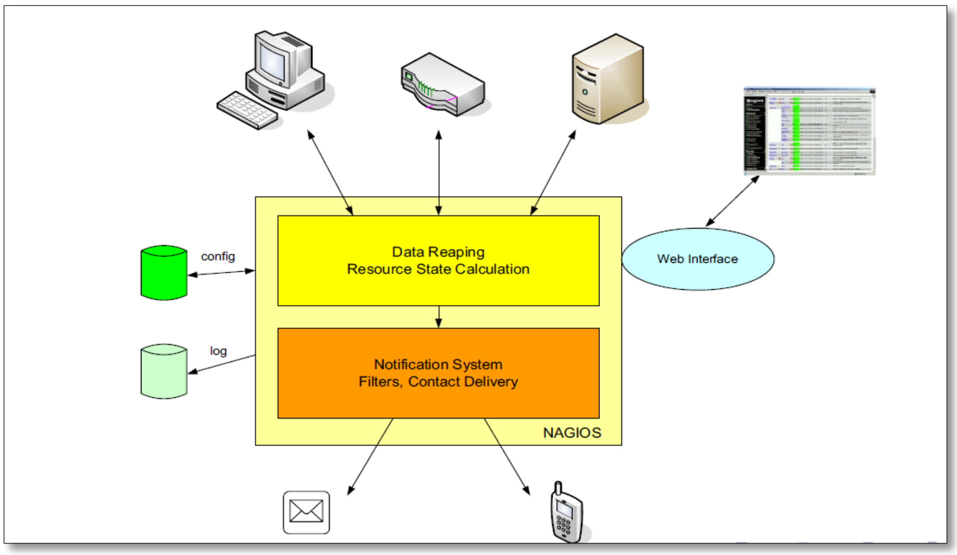


Figure 6.1 Nagios Architecture (Adapted from TEINS Training, 2005)

6.3 FEATURES OF NAGIOS

Nagios provides the following features:

1. **Comprehensive Monitoring:** Capabilities to monitor applications, services, operating systems, network protocols, system metrics and infrastructure components with a single tool. Powerful script APIs allow easy monitoring of in-house and custom applications, services, and systems.

2. **Visibility:** Fast detection of infrastructure outages. Alerts can be delivered to technical staff via email or SMS. Escalation capabilities ensure alert notifications reach the right people.

3. **Awareness:** Fast detection of infrastructure outages. Alerts can be delivered to technical staff via email or SMS. Escalation capabilities ensure alert notifications reach the right people.

4. **Problem Remediation:** Alert acknowledgments provide communication on known issues and problem response. Event handlers allow automatic restart of failed applications and services.

5. **Proactive Planning:** Trending and capacity planning add-ons ensure you're aware

of aging infrastructure. Scheduled downtime allows for alert suppression during infrastructure upgrades.

6. **Reporting:** Availability reports ensure SLAs are being met. Historical reports provide record of alerts, notifications, outages, and alert response.

7. **Multi-Tenant Capabilities:** Multi-user access to web interface allows stake holders to view infrastructure status. User-specific views ensures clients see only their infrastructure components.

8. **Extendible Architecture:** Integration with in-house and third-party applications is easy with multiple APIs. Hundreds of community-developed addons extend core Nagios functionality.

6.4 NAGIOS AGENTS

Nagios makes use of the following agents to perform its task:

1. **NRPE**

   Nagios Remote Plugin Executor (NRPE) is a Nagios agent that allows remote system monitoring using scripts that are hosted on the remote systems. It allows for monitoring of resources such as disk usage, system load or the number of users currently logged in. Nagios periodically polls the agent on remote system using the check_nrpe plugin.

   NRPE allows you to remotely execute Nagios plugins on other Linux/Unix machines. This allows you to monitor remote machine metrics (disk usage, CPU load, etc.). NRPE can also communicate with some of the Windows agent addons, so you can execute scripts and check metrics on remote Windows machines as well.

2. **NRDP**

   Nagios Remote Data Processor (NRDP) is a Nagios agent with a flexible data transport mechanism and processor. It is designed with an architecture that allows it to be easily extended and customized. NRDP uses standard ports and protocols (HTTP(S) and XML) and can be implemented as a replacement for NSCA.

3. **NSClient++**

   This program is mainly used to monitor Windows machines. Being installed on a remote system NSClient++ listens to port TCP 12489. Nagios plugin that is used to collect information from this addon is called check_nt. As NRPE, NSClient++ allows to monitor the so-called 'private services' (memory usage, CPU load, disk usage, running processes, etc.

PART II

DESIGN PHASE

The Design Phase takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced.

Design elements describe the desired system features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudo-code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the system in sufficient detail, such that skilled developers and engineers may develop and deliver the system with minimal additional input design.

Subsequent Chapters in this part of the report include the basic system architecture and its layout. Various UML diagrams used to represent the system are also covered in this part of the report.

CHAPTER 7

DESIGN OVERVIEW

This chapter elaborates and explains the aim of our project statement through various Architecture Diagrams. High Level System Architecture gives a glance on the working of the system followed by a Block Diagram which shows how the components are arranged in the actual implementation. Towards the end of the chapter the OpenStack architecture used for implementation is described as well.

7.1 PROPOSED HIGH LEVEL SYSTEM ARCHITECTURE

The aim of this project is to addresses the problem of enabling energy-efficient resource allocation, hence leading to Green Cloud computing data centers, to satisfy competing applications' demand for computing services and save energy.

Figure 7.1 shows the high-level architecture for supporting energy-efficient service allocation in Green Cloud computing infrastructure.

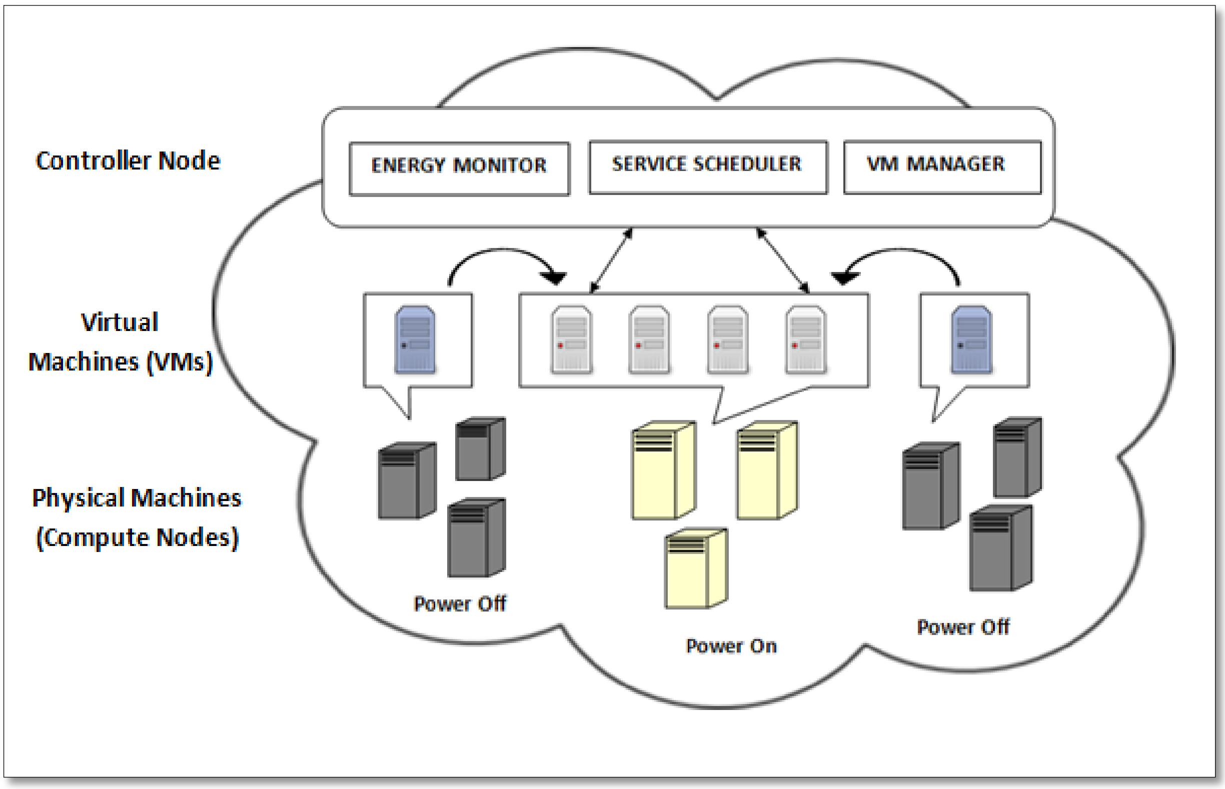


Figure 7.1 High-level System Architectural Framework

Following are the main entities involved:

a) **Controller Node:** Cloud consumers submit service requests from anywhere in the world to the Cloud. Controller acts as the interface between the Cloud infrastructure and consumers. It requires the interaction of the following components to support energy-efficient resource management:

   - Energy Monitor: Observes and determines which physical machines to power on/off.

   - Service Scheduler: Assigns requests to VMs and determines resource entitlements for allocated VMs. It also decides when VMs are to be added or removed to meet demand.

   - VM Manager: Keeps track of the availability of VMs and their resource entitlements. It is also in charge of migrating VMs across physical machines.

b) **Virtual Machines:** Multiple VMs can be dynamically started and stopped on a single physical machine to meet accepted requests, hence providing maximum flexibility to configure various partitions of resources on the same physical machine to different specific requirements of service requests.

   Multiple VMs can also concurrently run applications based on different operating system environments on a single physical machine. In addition, by dynamically migrating VMs across physical machines, workloads can be consolidated and unused resources can be put on a low-power state, turned off or configured to operate at low-performance levels (e.g., using DVFS) in order to save energy.

c) **Physical Machines (Compute Nodes):** The underlying physical computing servers provide hardware infrastructure for creating Virtualized resources to meet service demands.

7.2 BASIC BLOCK DESIGN

The actual system designed to follow up the above proposed high level system architecture is represented as shown in Figure 7.2.

The system consists of a Controller Node and a pair of Compute Nodes connected through Nova Network. All the three physical nodes are located in the same Subnet.

The three nodes make use of SSH and REST API’s to communicate with each other.

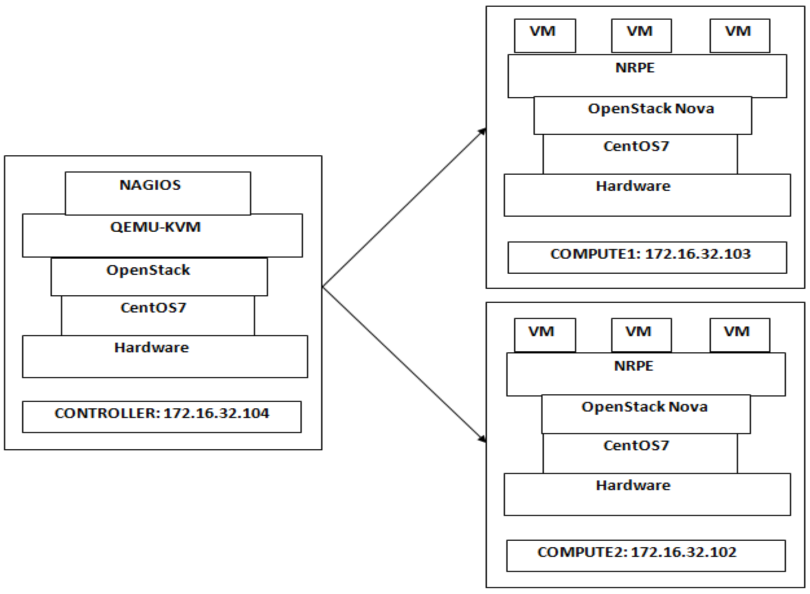


Figure 7.1 Basic Architectural Block Diagram of the System

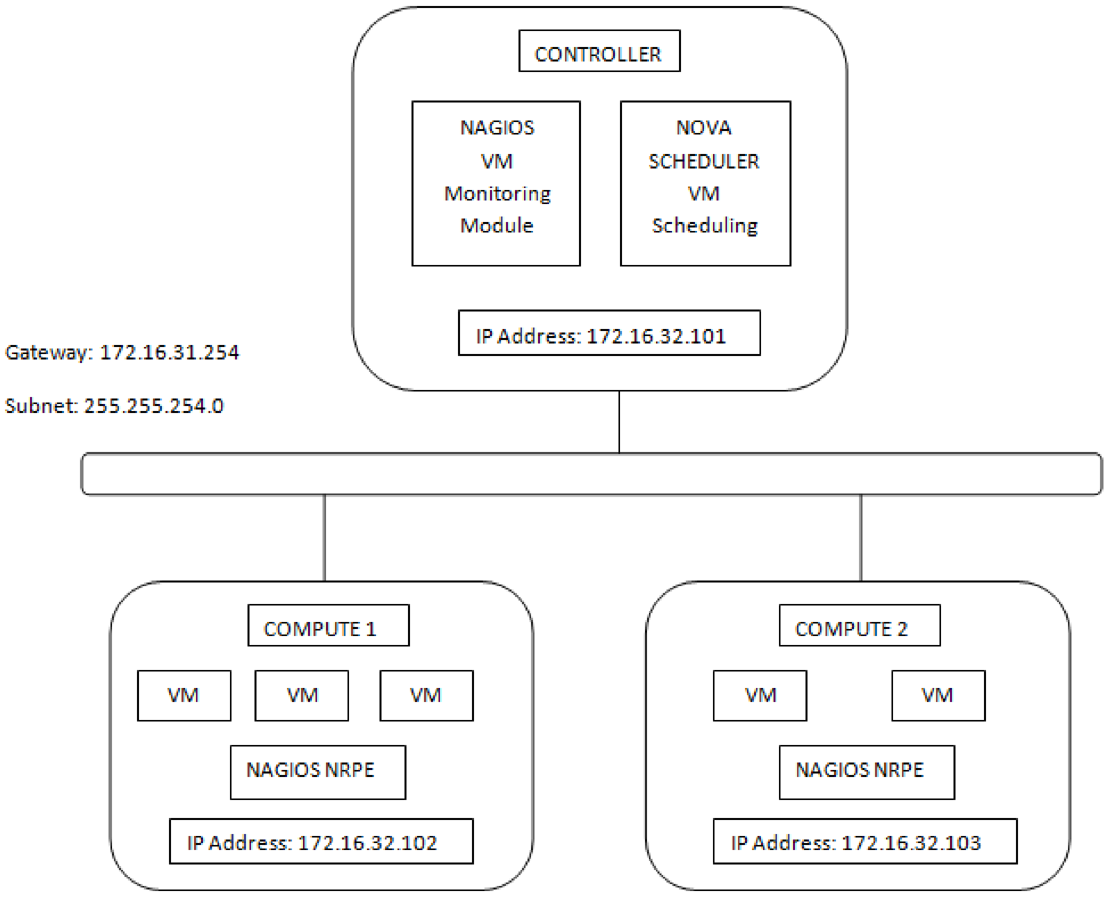

CONTROLLER
NAGIOS VM Monitoring Module
NOVA SCHEDULER VM Scheduling
IP Address: 172.16.32.101
Gateway: 172.16.31.254
Subnet: 255.255.254.0
COMPUTE 1
VM
VM
VM
NAGIOS NRPE
IP Address: 172.16.32.102
COMPUTE 2
VM
VM
NAGIOS NRPE
IP Address: 172.16.32.103

Figure 7.2 Basic Architectural Block Diagram of the System

Controller Node is the main management block in the architecture. It handles all the management level work. It consists of two main modules needed in the project.

1) Monitoring Module: This makes use of Nagios to monitor the workload on the compute nodes.

2) Scheduler Module: This makes the decision regarding the placement and migration of the VMs among different host machines.

Compute Nodes house several Virtual Machines in it. It further has Nagios NRPE agent in it that assists the Nagios Server located on the controller in monitoring the workload and the resource utilization of the physical machines.

All the three nodes make use of the OpenStack Architecture described in the following section to provide a private cloud infrastructure.

7.3 OPENSTACK TWO-NODE ARCHITECTURE WITH LEGACY NETWORKING

OpenStack is highly configurable to meet different needs with various compute, networking, and storage options. We can choose our own OpenStack adventure using a combination of core and optional services. Here, we use the following architecture.

Two-node architecture with legacy networking (nova-network) and optional nodes for Block Storage and Object Storage services.

- The controller node runs the Identity service, Image Service, management portion of Compute, and the dashboard. It also includes supporting services such as a SQL database, message queue, and Network Time Protocol (NTP).

  Optionally, the controller node runs portions of Block Storage, Object Storage, Orchestration, Telemetry, Database, and Data Processing services. These components provide additional features for your environment.

- The compute node runs the hypervisor portion of Compute that operates tenant virtual machines or instances. By default, Compute uses KVM as the hypervisor. Compute also provisions tenant networks and provides firewalling (security groups) services. You can run more than one compute node.

  Optionally, the compute node runs a Telemetry agent to collect metrics. Also, it can contain a third network interface on a separate storage network to improve performance of storage services.

- The optional Block Storage node contains the disks that the Block Storage service provisions for tenant virtual machine instances. You can run more than one of these nodes.

Optionally, the Block Storage node runs a Telemetry agent to collect metrics. Also, it can contain a second network interface on a separate storage network to improve performance of storage services.

- The optional Object Storage nodes contain the disks that the Object Storage service uses for storing accounts, containers, and objects. You can run more than two of these nodes. However, the minimal architecture example requires two nodes.

  Optionally, these nodes can contain a second network interface on a separate storage network to improve performance of storage services.

Figure 7.3 shows the basic minimal Hardware requirements of the Compute and the Controller nodes in the above architecture.

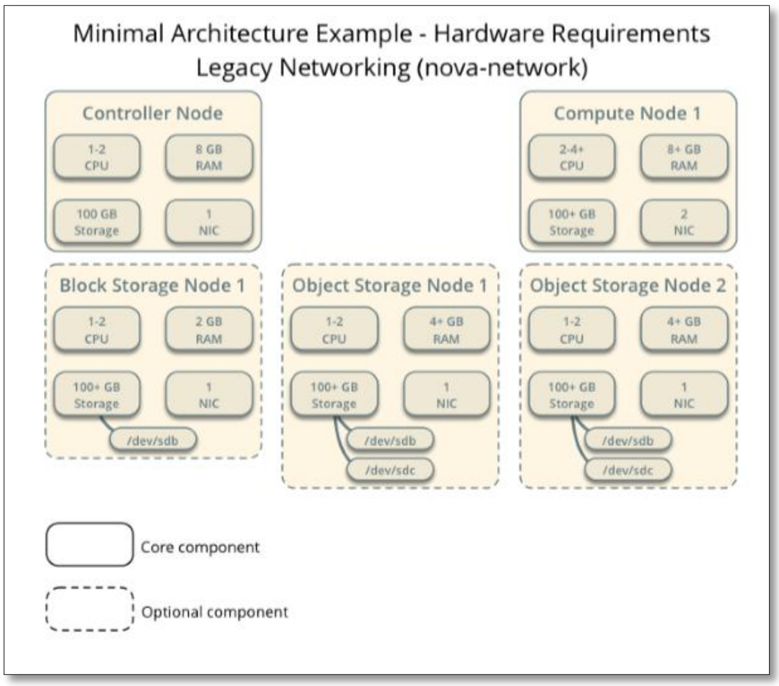


Figure 7.3 Minimal architecture example with legacy networking (nova-network)—Hardware requirements

The basic Network Layout of the Compute and the Controller nodes in the above architecture is described in Figure 7.4.

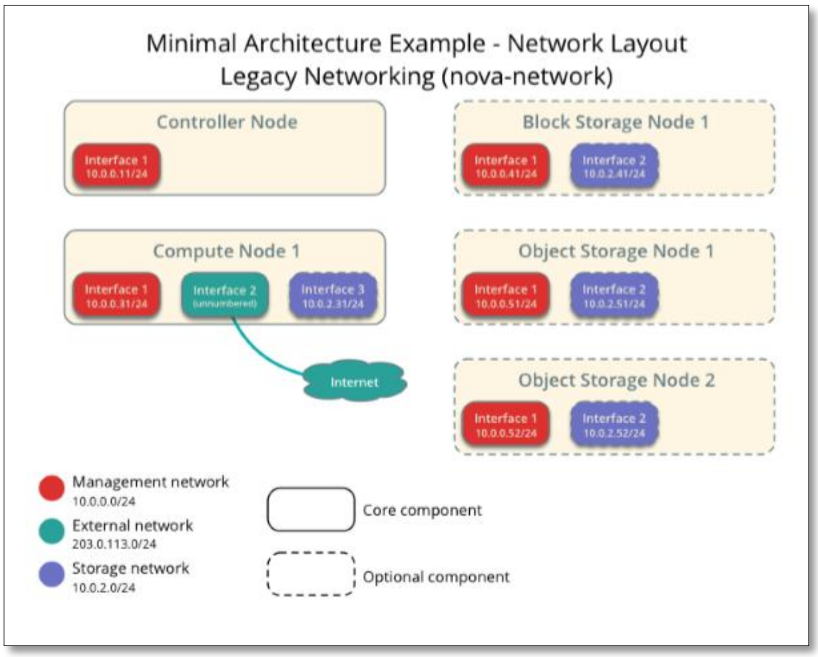


Figure 7.4 Minimal architecture example with legacy networking (nova-network)—Network layout

The design of the OpenStack Service fabric for the Controller and the Compute nodes for the above described architecture is shown in Figure 7.5.

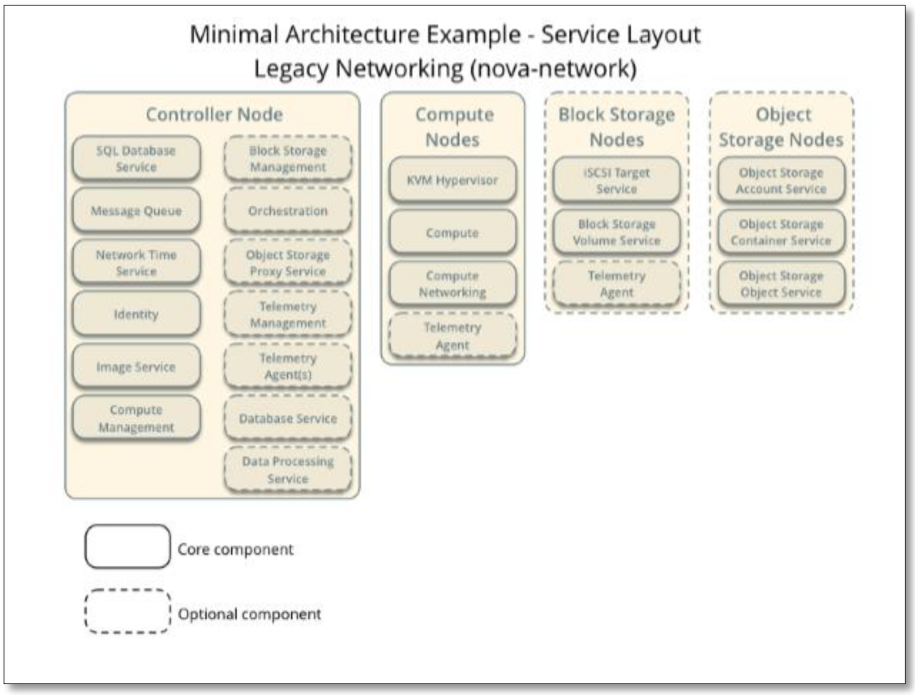


Figure 7.5 Minimal architecture example with legacy networking (nova-network)—Service layout

CHAPTER 8

SCHEDULER DESIGN

This chapter describes the internals of the Scheduler. It further elaborates the process of launching an instance and VM migration in OpenStack Nova. The working of the Scheduler along with the details of the filtering and weighing functions are also included in this chapter.

8.1 OVERVIEW

Launching a new instance involves multiple components inside OpenStack Nova:

- **API server:** handles requests from the user and relays them to the cloud controller.
- **Cloud controller:** handles the communication between the compute nodes, the networking controllers, the API server and the scheduler.
- **Scheduler:** selects a host to run a command.
- **Compute worker:** manages computing instances: launch/terminate instance, attach/detach volumes.
- **Network controller:** manages networking resources: allocate fixed IP addresses, configure VLANs.

The flow of launching an instance goes like this:

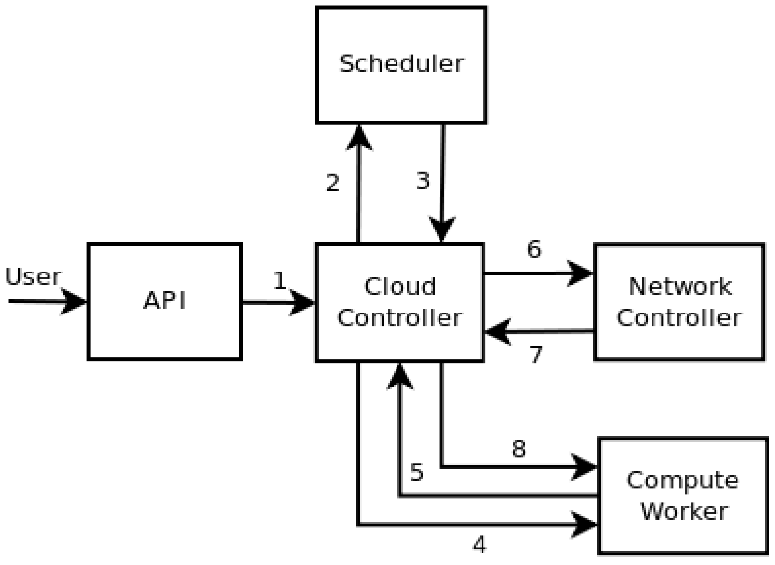


Figure 8.1 Launching of Instance

(1) The API server receives a run_instances command from the user. The API server relays the message to the cloud controller.

(2) Authentication is performed to make sure this user has the required permissions. The cloud controller sends the message to the scheduler.

(3) The scheduler casts the message to a random host and asks him to start a new instance.

(4) The compute worker on the host grabs the message.

(5,6,7,8) The compute worker needs a fixed IP to launch a new instance so it sends a message to the network controller. The compute worker continues with spawning a new instance.

8.2 OPENSTACK NOVA-SCHEDULER

Nova Compute uses the nova-scheduler service to determine which compute node should be used when a VM instance is to be launched. The nova-scheduler makes this automatic initial placement decision using a number of metric and policy considerations, such as available compute resources and VM affinity rules. The nova-scheduler does not do periodic load-balancing or power-management of VMs after the initial placement.

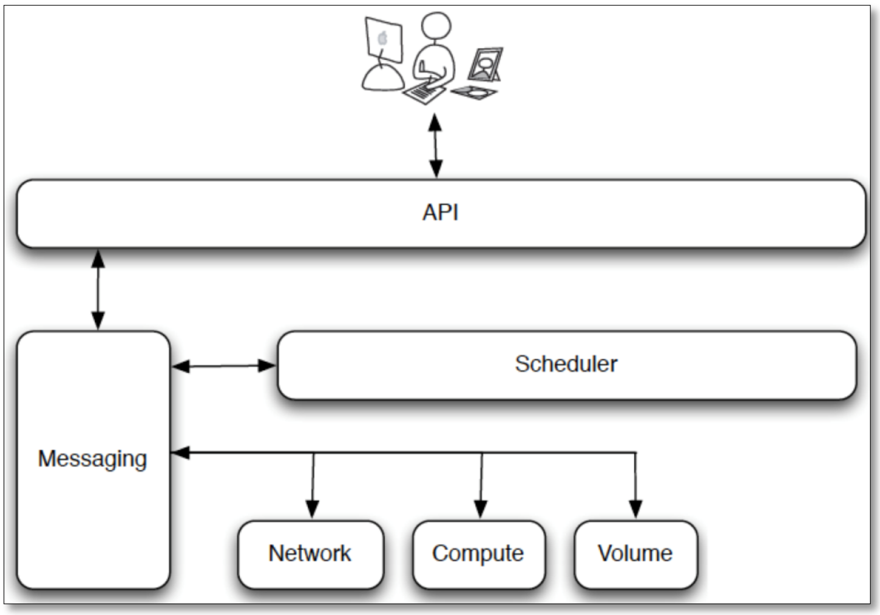


Figure 8.2 Openstack-Nova-Scheduler

The scheduler has a number of configurable options that can be accessed and modified in the nova.conf file, the primary file used to configure nova-compute services in an OpenStack cloud.

```
scheduler_driver=nova.scheduler.multi.MultiScheduler

compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler

scheduler_available_filters=nova.scheduler.filters.all_filters

--> scheduler_default_filters=AvailabilityZoneFilter,RamFilter,ComputeFilter -->
least_cost_functions=nova.scheduler.least_cost.compute_fill_first_cost_fn
compute_fill_first_cost_fn_weight=-1.0
```

The two variables “schedule_default_filters” and “Least_cost_functions” represent two algorithmic processes used by the Filter Scheduler to determine initial placement of VMs (There are two other schedulers, the Chance Scheduler and the Multi Scheduler, that can be used in place of the Filter Scheduler; however, the Filter Scheduler is the default and should be used in most cases). These two process work together to balance workloads across all existing compute nodes at VM launch.

8.2.1 FILTERING

Filter scheduler

The filter scheduler (nova.scheduler.filter_scheduler.FilterScheduler) is the default scheduler for scheduling virtual machine instances. It supports filtering and weighting to make informed decisions on where a new instance should be created.

Filtering (schedule_default_filters)

When a request is made to launch a new VM, the nova-compute service contacts the nova-scheduler to request placement of the new instance. The nova-scheduler uses the scheduler, by default the Filter Scheduler, named in the nova.conf file to determine that placement. First, a filtering process is used to determine which hosts are eligible for consideration and an eligible host list is created and then a second algorithm, Costs and Weights (described later), is applied against the list to determine which compute node is optimal for fulfilling the request.

The Filtering process uses the scheduler_available_filters configuration option in nova.conf to determine what filters will be used to filter out ineligible compute nodes and to create the eligible hosts list. By default, there are three filters that are used:

- The **AvailabilityZoneFilter** filters out hosts that do not belong to the Availability Zone specified when a new VM launch request is made via the Horizon dashboard or from the nova CLI client.

- The **RamFilter** ensures that only nodes with sufficient RAM make it on to the eligible host list. If the RamFilter is not used, the nova-scheduler may over-provision a node with insufficient RAM resources. By default, the filter is set to allow overcommitment on RAM by 50 percent, i.e. the scheduler will allow a VM requiring 1.5 GB of RAM to be launched on a node with only 1 GB of available RAM. This setting is configurable by changing the "ram_allocation_ratio=1.5″ setting in nova.conf.

- The **ComputeFilter** filters out any nodes that do not have sufficient capabilities to launch the requested VM as it corresponds to the requested instance type. It also filters out any nodes that cannot support properties defined by the image that is being used to launch the VM. These image properties include architecture, hypervisor and VM mode.

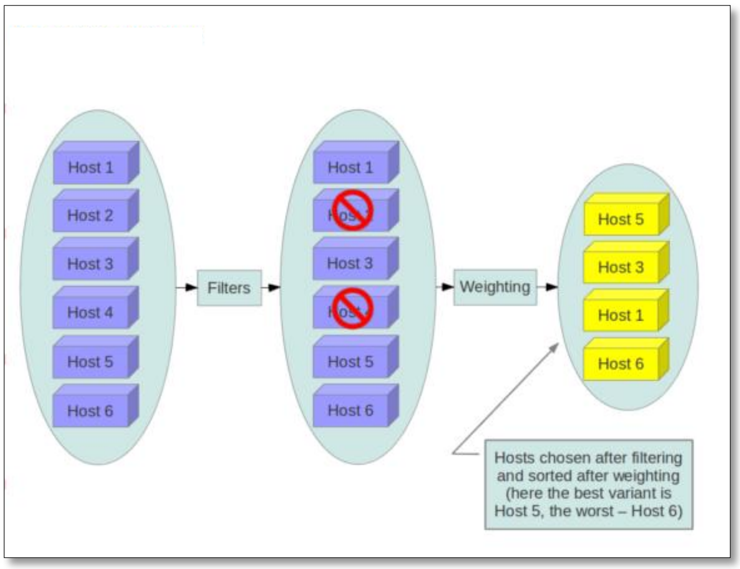

Host 1
Host 2
Host 3
Host 4
Host 5
Host 6
Filters
Host 1
Host 3
Host 5
Host 6
Weighting
Host 5
Host 3
Host 1
Host 6
Hosts chosen after filtering and sorted after weighting (here the best variant is Host 5, the worst – Host 6)

Figure 8.3 Process of Filtering Host

An example of how the Filtering process may work is that host 2 and host 4 above may have been initially filtered out for any combination of the following reasons, assuming the default filters were applied:

1. The requested VM is to be in Availability Zone 1 while nodes 2 and 4 are in Availability Zone 2.

2. The requested VM requires 4 GB of RAM and nodes 2 and 4 each have only 2 GB of available RAM.

3. The requested VM has to run on vSphere and nodes 2 and 4 support KVM.

There are a number of other filters that can be used along with or in place of the default filters; some of them include:

- The **CoreFilter** ensures that only nodes with sufficient CPU cores make it on to the eligible host list. If the CoreFilter is not used, the nova-scheduler may over-provision a node with insufficient physical cores. By default, the filter is set to allow overcommitment based on a ratio of 16 virtual cores to one physical core. This setting is configurable by changing the "cpu_allocation_ratio=16.0″ setting in nova.conf.

- The **DifferentHostFilter** ensures that the VM instance is launched on a different compute node from a given set of instances, as defined in a scheduler hint list.

- The **SameHostFilter** ensures that the VM instance is launched on the same compute node as a given set of instances, as defined in a scheduler hint list.

8.2.2 WEIGHING

Costs and Weights

When resourcing instances, the filter scheduler filters and weights each host in the list of acceptable hosts. Each time the scheduler selects a host, it virtually consumes resources on it, and subsequent selections are adjusted accordingly. This process is useful when the customer asks for the same large amount of instances, because weight is computed for each requested instance.

All weights are normalized before being summed up; the host with the largest weight is given the highest priority.

Next, the Filter Scheduler takes the hosts that remain after the filters have been applied and applies one or more cost function to each host to get numerical scores for each host. Each cost score is multiplied by a weighting constant specified in the nova.conf config file. The weighting constant configuration option is the name of the cost function, with the _weight string appended. Here is an example of specifying a cost function and its corresponding weight:

```
least_cost_functions=nova.scheduler.least_cost.compute_fill_first_cost_fn,nova.sc
heduler.least_cost.noop_cost_fn

compute_fill_first_cost_fn_weight=-1.0

noop_cost_fn_weight=1.0
```

There are three cost functions available; any of the functions can used alone or in any combination with the other functions.

1. The **nova.scheduler.least_cost.compute_fill_first_cost_fn** function calculates the amount of available RAM on the compute nodes and chooses which node is best suited for fulfilling a VM launch request based on the weight assigned to the function. A weight of 1.0 will configure the Scheduler to "fill-up" a node until there is insufficient RAM available. A weight of -1.0 will configure the scheduler to favor the node with the most available RAM for each VM launch request.

2. The **nova.scheduler.least_cost.retry_host_cost_fn** function adds additional cost for retrying a node that was already used for a previous attempt. The intent of this function is to ensure that nodes which consistently encounter failures are used less frequently.

3. The **nova.scheduler.least_cost.noop_cost_fn** function will cause the scheduler not to discriminate between any nodes. In practice this function is never used.

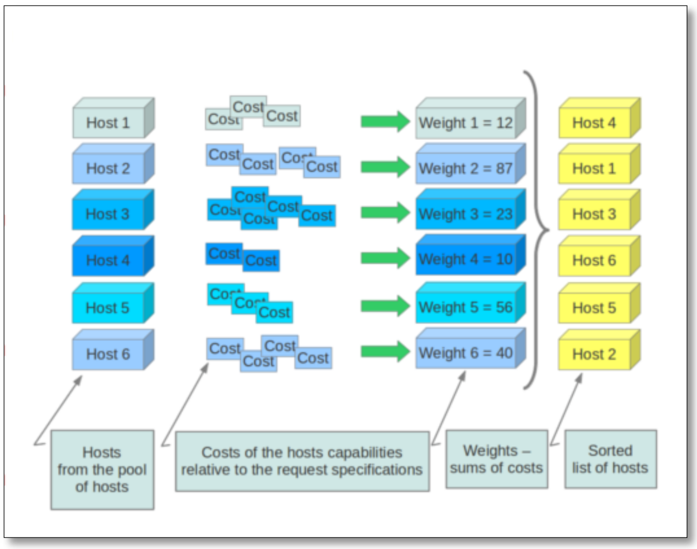


Figure 8.4 Weighing Hosts

In the example above, if we choose to only use the nova.scheduler.least_cost.compute_fill_first_cost_fn function and set the weight to compute_fill_first_cost_fn_weight=1.0, we would expect the following results:

- All nodes would be ranked according to amount of available RAM, starting with host 4.
- The nova-scheduler would favor launching VMs on host 4 until there is insufficient RAM available to launch a new instance.

CHAPTER 9

UML MODELING

The use of UML as a tool for defining the structure of a system is a very useful way to manage large, complex systems. Having a clearly visible structure makes it easy to introduce new people to an existing project. This chapter is dedicated to UML Designing. Here, different UML diagrams used to represent the system are presented.

9.1 INTRODUCTION

The Unified Modeling Language (UML) first appeared in the 1990's as an effort to select the best elements from the many modeling systems proposed at the time, and to combine them into a single coherent notation. It has since become the industry standard for software modeling and design, as well as the modeling of other processes in the scientific and business worlds.

The UML is a tool for specifying software systems. Standardized diagram types to help you describe and visually map a software system's design and structure. Using UML it is possible to model just about any kind of application, both specifically and independently of a target platform.

9.2 CLASS DIAGRAM

The class diagram shows the building blocks of any object-orientated system. Class diagrams depict a static view of the model, or part of the model, describing what attributes and behaviour it has rather than detailing the methods for achieving operations. Class diagrams are most useful in illustrating relationships between classes and interfaces. Generalizations, aggregations, and associations are all valuable in reflecting inheritance, composition or usage, and connections respectively.

The diagram below illustrates the different components like OpenStack, Nova Compute, Nagios, Virtual Machines Dashboard used in the system and how they are related to each other.

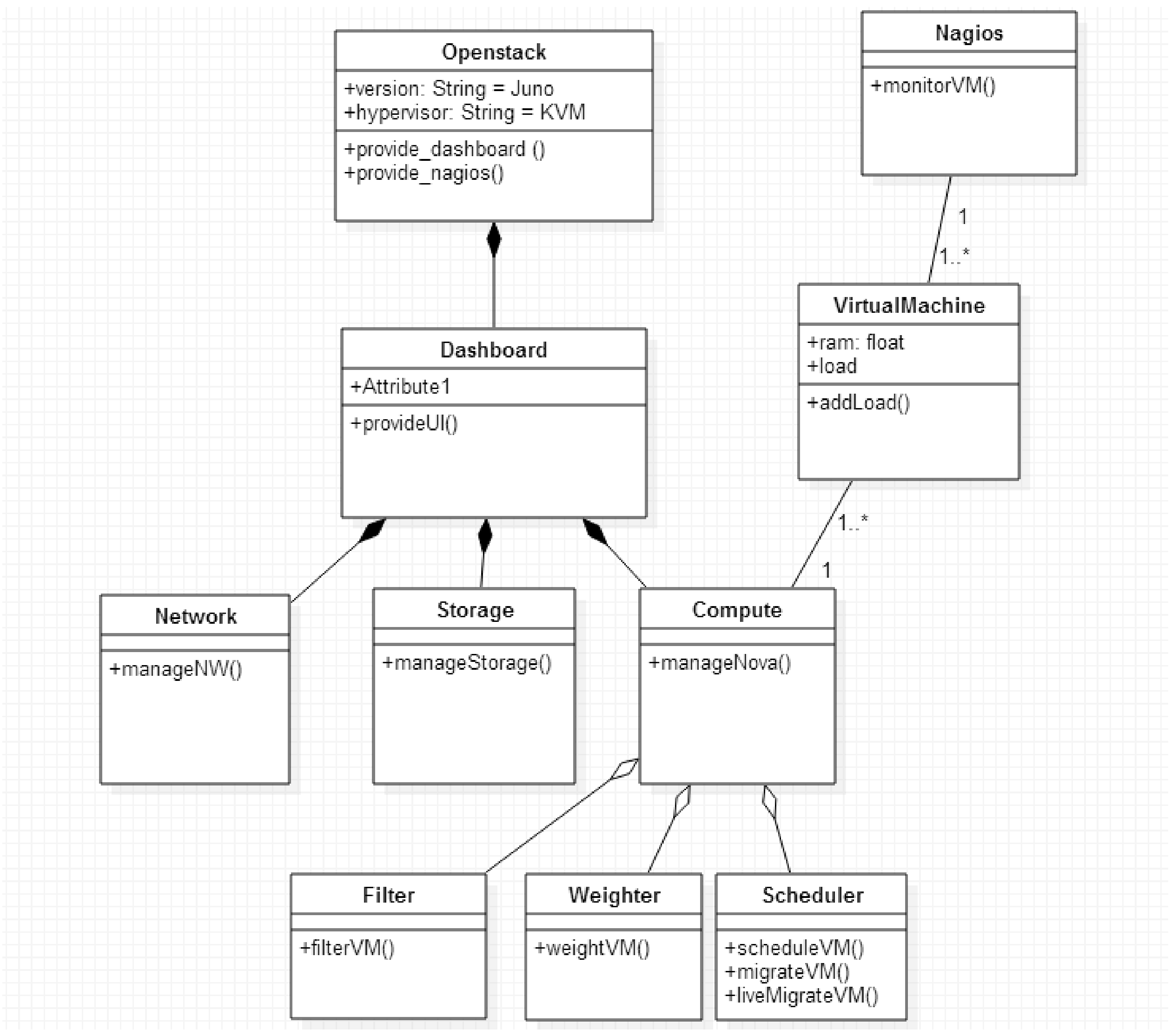


Figure 9.1 Class Diagram

9.3 DEPLOYMENT DIAGRAM

A deployment diagram models the run-time architecture of a system. It shows the configuration of the hardware elements (nodes) and shows how software elements and artifacts are mapped onto those nodes.

The diagram below illustrates the Deployment Diagram for the proposed system.

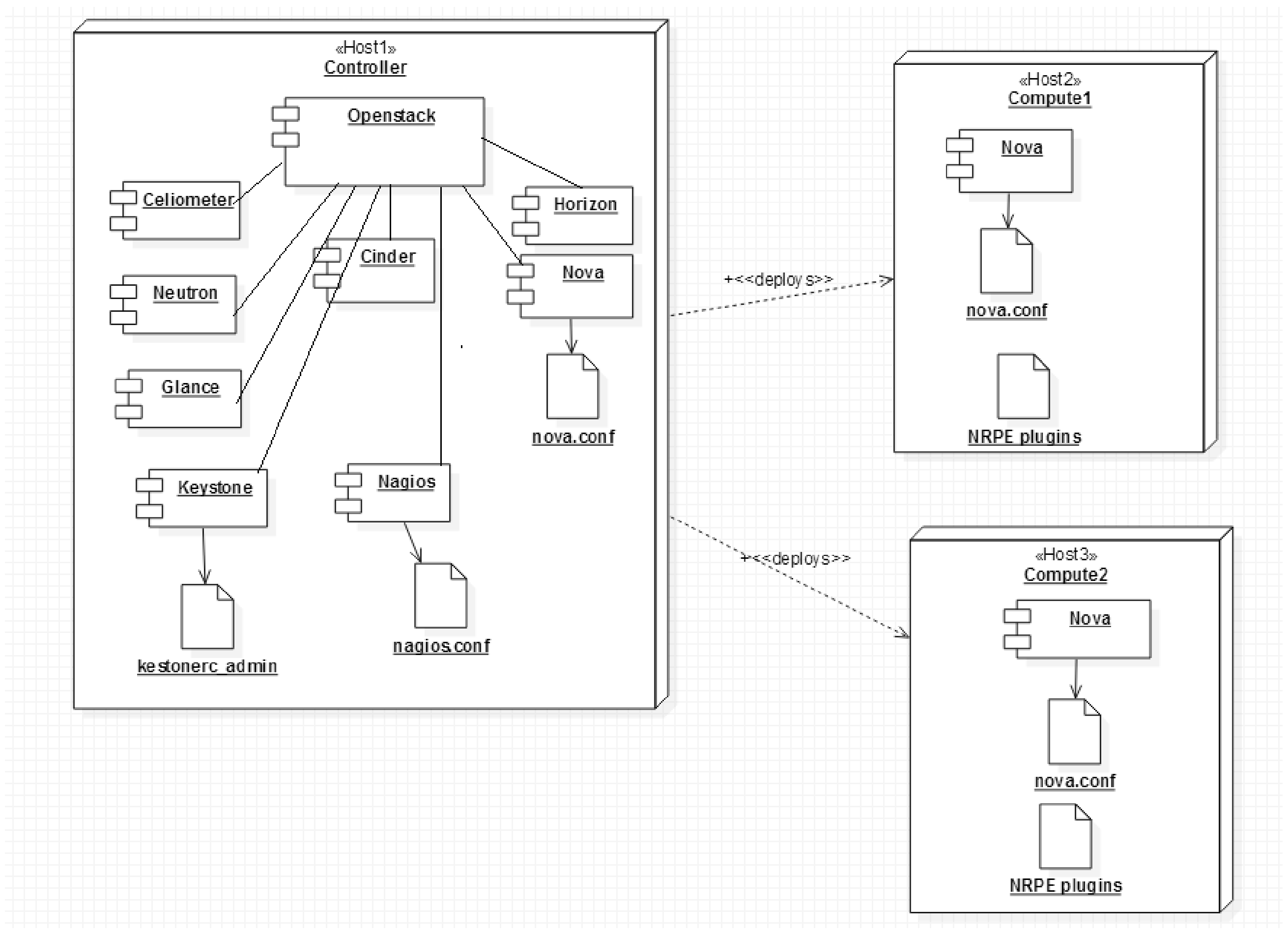


Figure 9.2 Deployment Diagram

9.4 ACTIVITY DIAGRAM

In UML, an activity diagram is used to display the sequence of activities. Activity diagrams show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity. They may be used to detail situations where parallel processing may occur in the execution of some activities. Activity diagrams are useful for business modelling where they are used for detailing the processes involved in business activities.

The diagram below illustrates the task of Launching an Instance and Migrating it when certain conditions are fulfilled.

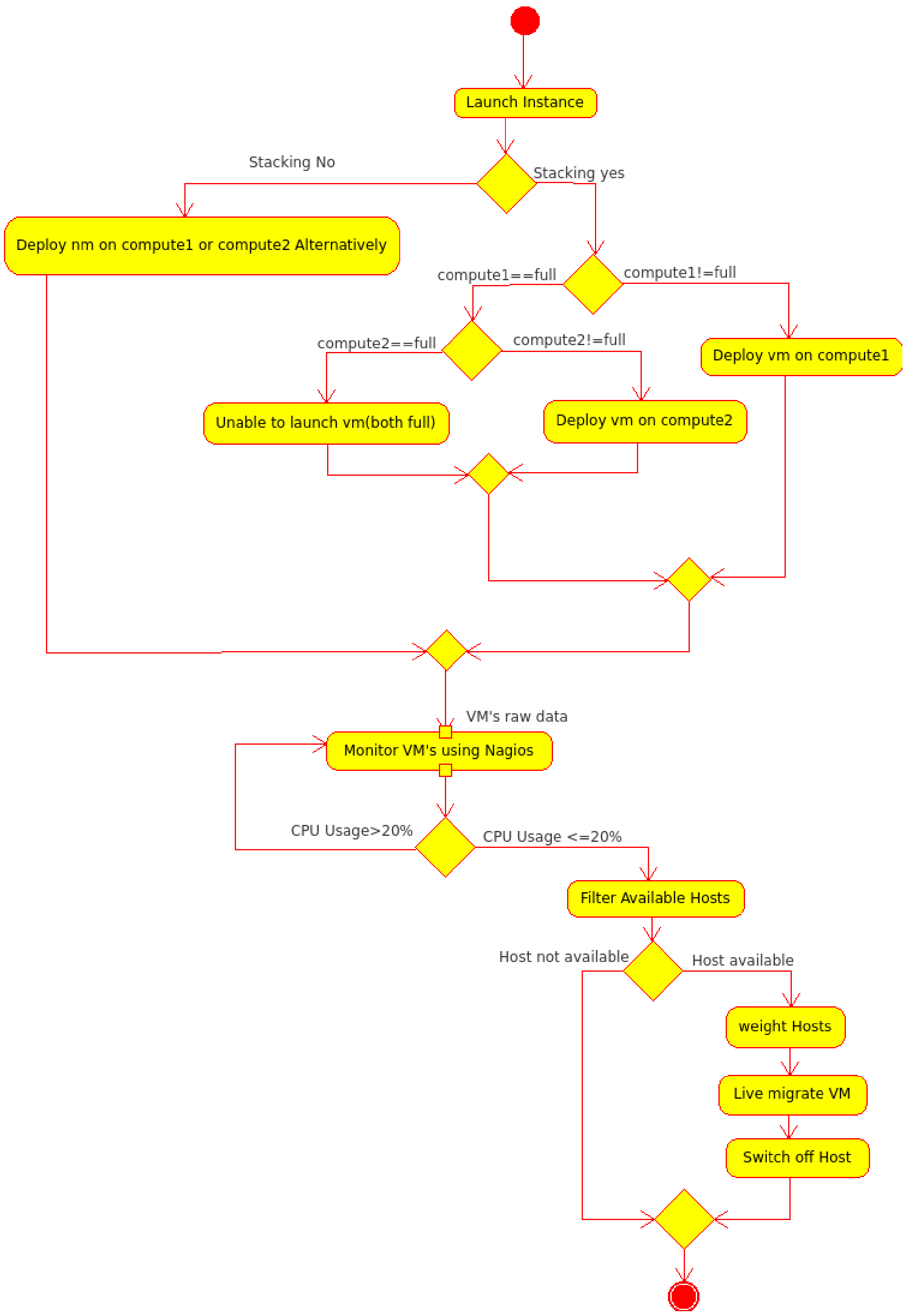


Figure 9.3 Activity Diagram Showing Activity of Launching and Migrating an Instance

9.5 SEQUENCE DIAGRAM

A sequence diagram is a form of interaction diagram which shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline. Sequence diagrams are good at showing which objects communicate with which other objects; and what messages trigger those communications. Sequence diagrams are not intended for showing complex procedural logic.

The diagram below illustrates the sequence of Launching, Monitoring and Migrating an Instance.

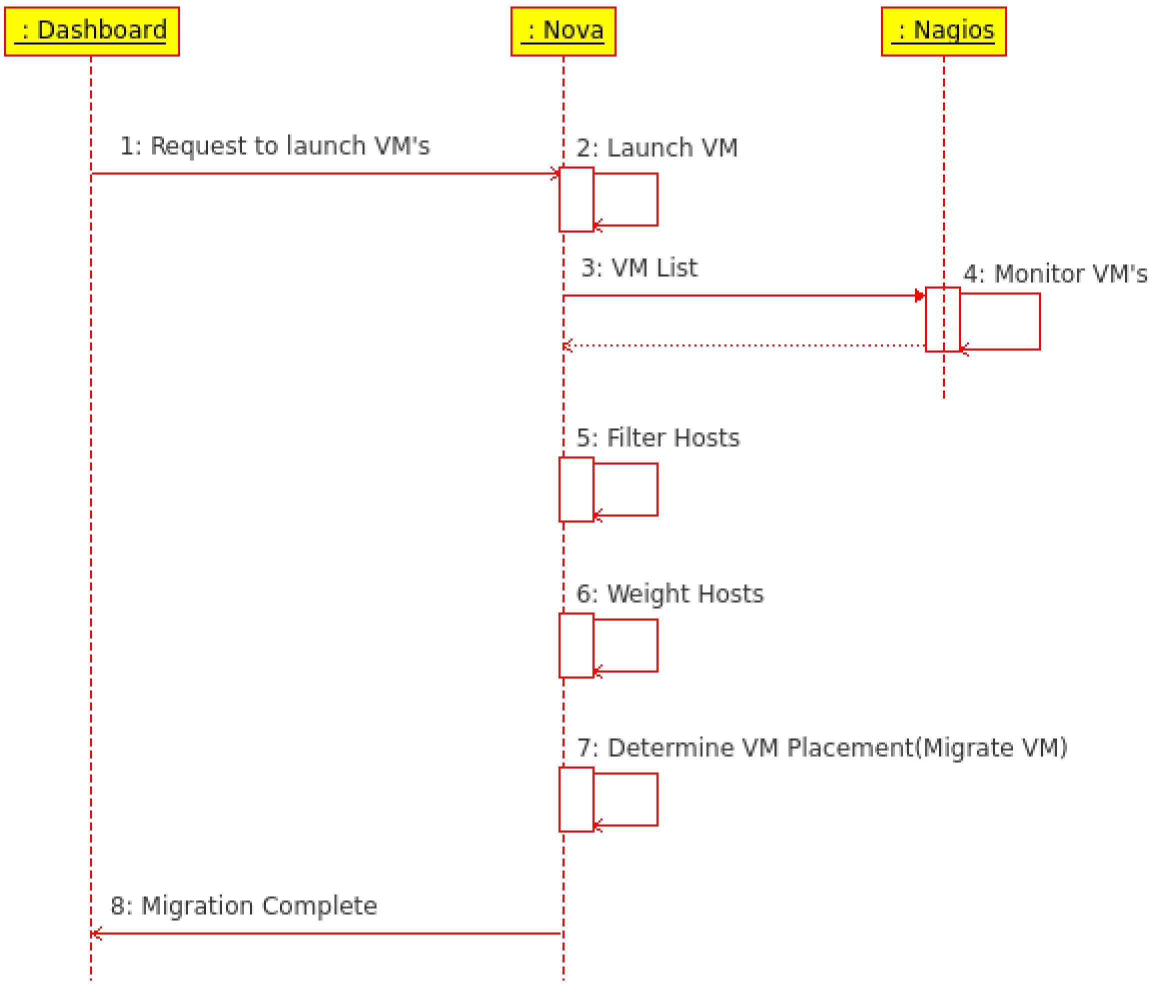


Figure 9.4 Sequence Diagram showing the operations of Launching, Monitoring and Migrating an Instance

PART III

IMPLEMENTATION PHASE

The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently. It includes installing the prerequisite software on user machines and setting up the infrastructural setup.

At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

CHAPTER 10

PROJECT IMPLEMENTATION

This chapter covers the implementation details of the system so proposed. It elaborates on the process of deploying the Cloud Infrastructure and the monitoring module for the purpose of monitoring load across the VM's. Delving further into the ways employed to achieve energy efficiency in the system by means of VM Consolidation.

10.1 INTRODUCTION

The first step in building the system requires setting up the cloud platform. The design requirements and specifications of which have already been proposed earlier.

The steps to install, configure and deploy the Cloud Platform using OpenStack Juno Version for the "Two-node architecture with legacy networking" for a system comprising of a Controller Node and a pair of Compute Nodes have been elaborated in depth in the Appendix section of this report.

Once the system is up and running, the next step is to Launch Virtual Machines on the setup and to add load to them.

Further the system is configured to check for the working of Migration and Live Migration operations of OpenStack.

Since the proposed system demands' monitoring of the VMs, the next step is to start Nagios for the purpose of monitoring.

Nagios is deployed on the Controller Node and Nagios NRPE component of the Nagios is installed on each of the Compute Node for the purpose of monitoring the VM's.

Nagios plugins are altered to provide customised services for the system such as monitoring of the RAM usage, Performance Load and Disk Space Availability of the Compute Nodes.

Having setup the cloud infrastructure for the system and prepared the monitoring module, the next milestone to be achieved is to bring about energy efficiency in the system.

The next few sections in this chapter highlight the ways and means to make the system energy efficient.

10.2 STATIC CONSOLIDATION OF VM'S

One of the ways to achieve Energy Efficiency in the system employs consolidating the VM's statically during the launching of new instances. This ensures that the VM's are allocated to the same node till its capacity is exhausted.

This is achieved by altering the "Ram filter scheduling algorithm" of OpenStack to allow for Stacking of VM's instead of Spreading them equally across Compute Nodes using Load Balancing Techniques.

The major drawback with this approach is that it does not take into consideration the runtime load changes in the system.

```
from oslo.config import cfg

from nova.scheduler import weights

ram_weight_opts = [
cfg.FloatOpt('ram_weight_multiplier',
default=1.0,
help='Multiplier used for weighing ram.  Negative '
'numbers mean to stack vs spread.'),
]

CONF = cfg.CONF
CONF.register_opts(ram_weight_opts)


class RAMWeigher(weights.BaseHostWeigher):
minval = 0

def weight_multiplier(self):
"""Override the weight multiplier."""
return CONF.ram_weight_multiplier

def _weigh_object(self, host_state, weight_properties):
"""Higher weights win.  We want spreading to be the default."""
return host_state.free_ram_mb
```

10.3 DYNAMIC CONSOLIDATION OF VM'S DURING MIGRATION

The study of various research papers on this topic highlighted the technique of Dynamic Consolidation of VM's during Migration to achieve the goal of energy efficiency in cloud.

This approach solves the problem associated with the above method and allows for handing of load changes during runtime.

The algorithm proposed for this takes RAM Usage as input data from Nagios to make decision regarding VM Migration for the purpose of Consolidation.

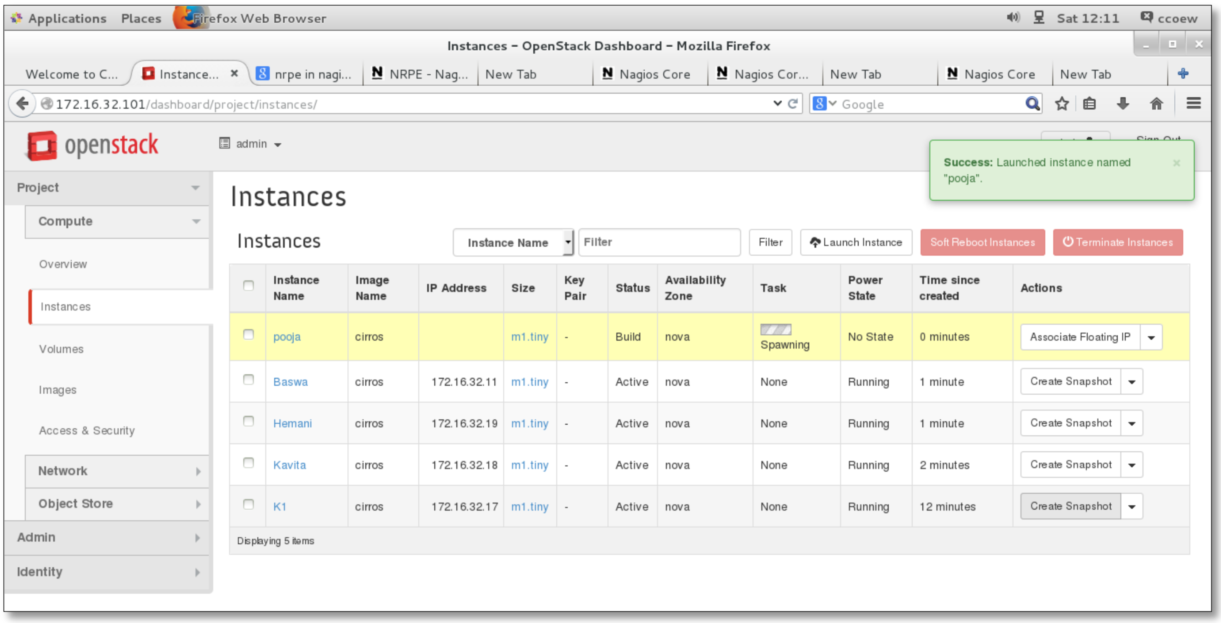
The idea here is to consolidate VM’s when the RAM Usage on either of the Compute Node drops to less than 20% of the total RAM allocation for that node and to shut down the Node if migration is successful.

```
#!/usr/bin/python

import os

from novaclient.openstack.common.gettextutils import _

from novaclient import utils

from nova.compute import utils as compute_utils

from nova import exception

from nova import manager

from nova import objects

class Host(object):

    def __init__(self, total, avail, used, name):

        self._totalSpace = total

        self._availSpace = avail

        self._usedSpace = used

        self._name = name


   def migrateInstance(h1, h2):

       while True:

           if h1.usedSpace < h2.usedSpace:

               if h1._usedSpace <= 0.2 * h1._totalSpace and h2.usedSpace <= h1.availSpace:
instance_uuids = h1.request_spec['instance_uuid']

                   for uuid in self.instance_uuids:

                       try:

                           servers.migrate(server['uuid'])
```

```
                        os.system("shutdown now -h")
                        return "Migration Successful"
                    except Exception as e:
                       error_message = _("Error while migrating instance: %s") % e
                       return "Migration Error"
        else:
             if h2._usedSpace <= 0.2 * h2._totalSpace and h1.usedSpace <= h2.availSpace:
instance_uuids = h2.request_spec['instance_uuid']
                  for uuid in self.instance_uuids
                       try:
                            servers.migrate(server['uuid'])
                            os.system("shutdown now -h")
                            return "Migration Successful"
                       except Exception as e:
                            error_message = _("Error while migrating instance: %s") % e
                            return "Migration Error"


total_1 = 30
avail_1 = 10
used_1  = 20
h1 = Host(total_1, avail_1, used_1, "compute1")
total_1 = 30 avail_1 = 20
used_1  = 10
h2 = Host(total_1, avail_1, used_1, "compute2")
status = Host.migrateInstance(h1, h2)
print "Message:",status
```

10.4 SCREENSHOTS

OpenStack is used to manage the cloud resources such as Instances, Hypervisors, Networks and many more. The figures below show the representation of the same.



Instances – OpenStack Dashboard – Mozilla Firefox

172.16.32.101/dashboard/project/instances/

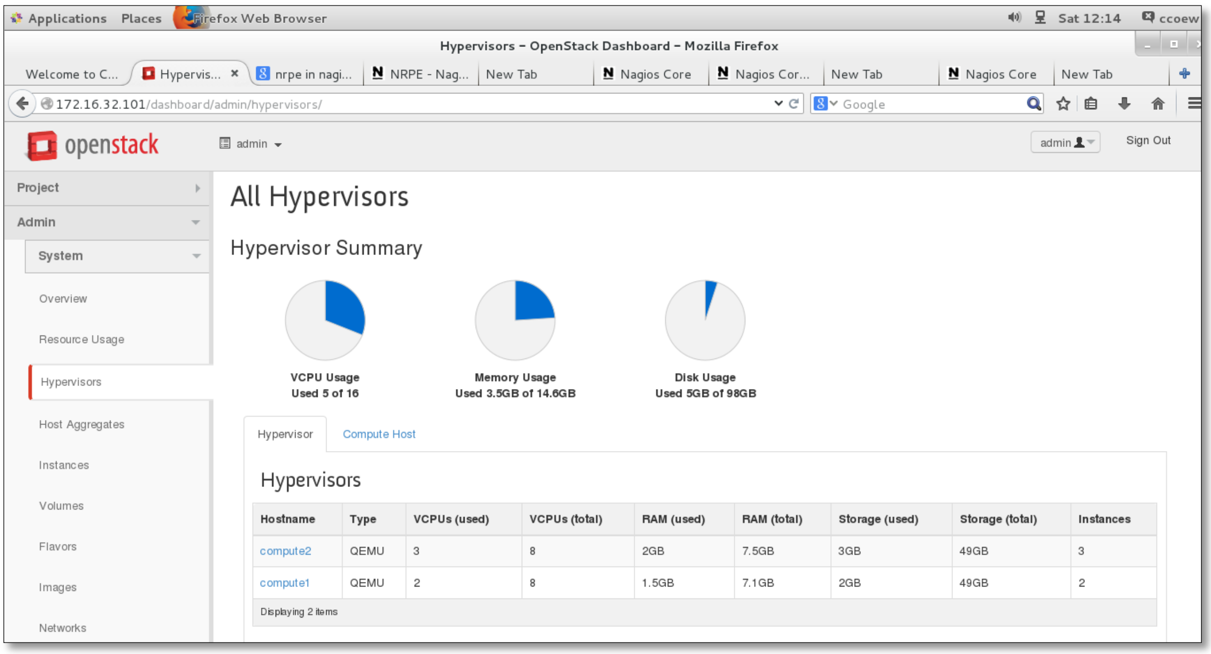
Success: Launched instance named "pooja".

Instances

	Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
☐	pooja	cirros		m1.tiny	-	Build	nova	Spawning	No State	0 minutes	Associate Floating IP
☐	Baswa	cirros	172.16.32.11	m1.tiny	-	Active	nova	None	Running	1 minute	Create Snapshot
☐	Hemani	cirros	172.16.32.19	m1.tiny	-	Active	nova	None	Running	1 minute	Create Snapshot
☐	Kavita	cirros	172.16.32.18	m1.tiny	-	Active	nova	None	Running	2 minutes	Create Snapshot
☐	K1	cirros	172.16.32.17	m1.tiny	-	Active	nova	None	Running	12 minutes	Create Snapshot

Displaying 5 items

Figure 10.1 OpenStack Dashboard Showing Instance Details



Hypervisors – OpenStack Dashboard – Mozilla Firefox

172.16.32.101/dashboard/admin/hypervisors/

All Hypervisors

Hypervisor Summary

VCPU Usage
Used 5 of 16

Memory Usage
Used 3.5GB of 14.6GB

Disk Usage
Used 5GB of 98GB

Hypervisors

Hostname	Type	VCPUs (used)	VCPUs (total)	RAM (used)	RAM (total)	Storage (used)	Storage (total)	Instances
compute2	QEMU	3	8	2GB	7.5GB	3GB	49GB	3
compute1	QEMU	2	8	1.5GB	7.1GB	2GB	49GB	2

Displaying 2 items

Figure 10.2 OpenStack Dashboard Showing Hypervisors Details

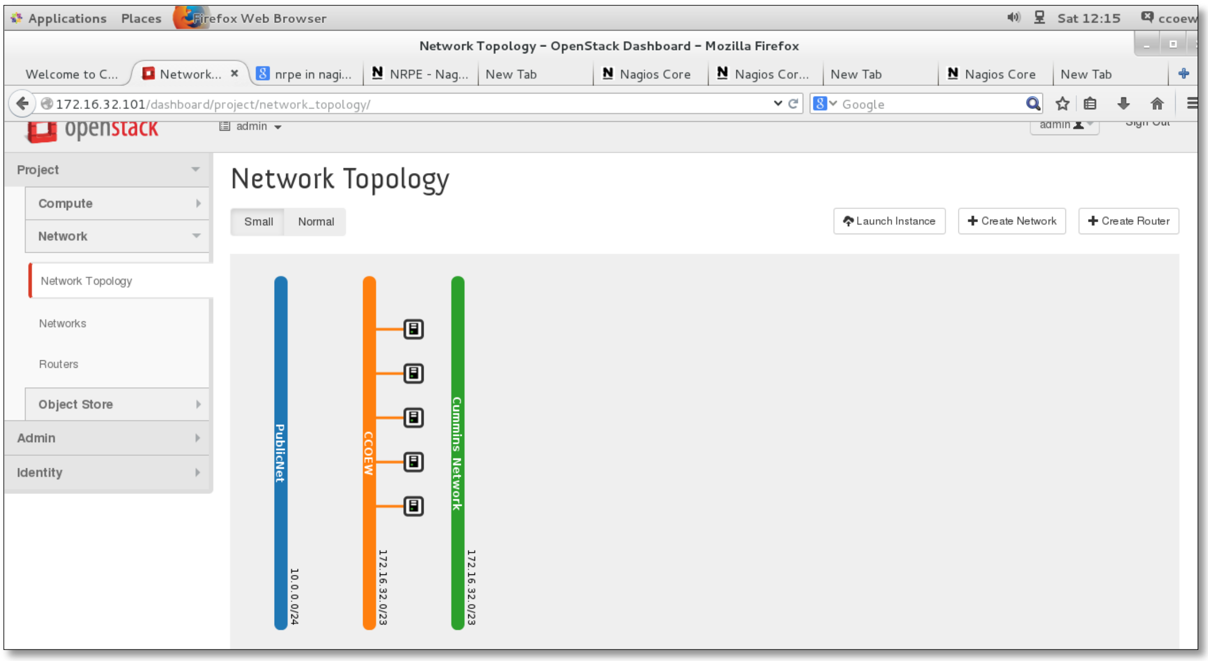
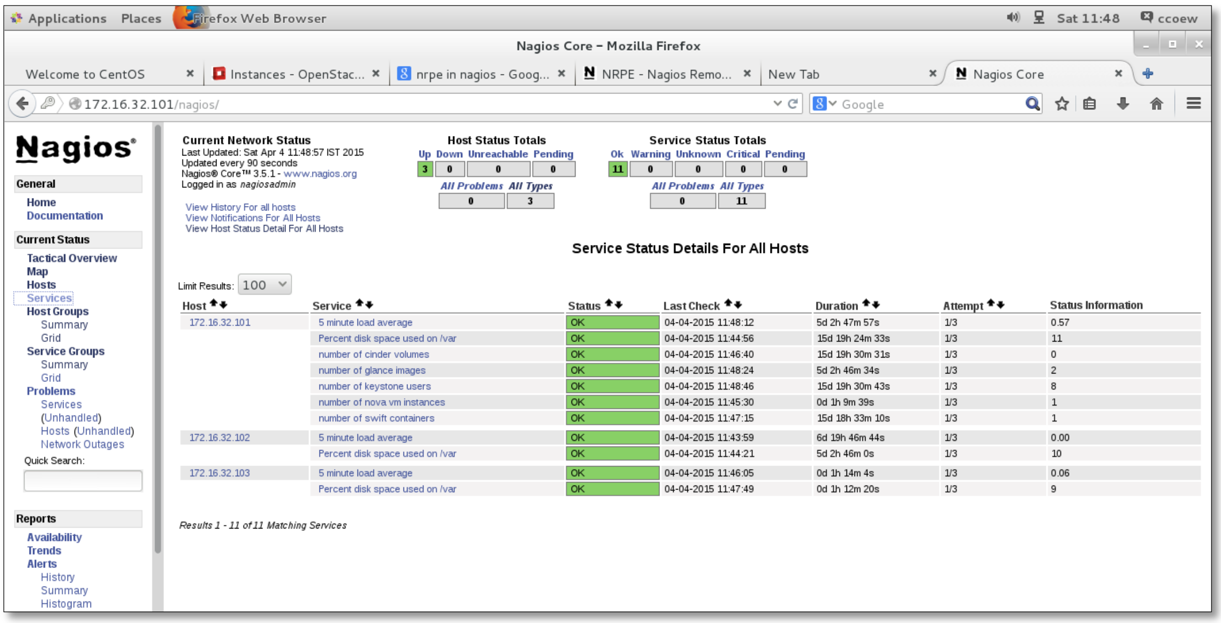


Figure 10.3 OpenStack Dashboard Showing Network Topology



Current Network Status
Last Updated: Sat Apr 4 11:48:57 IST 2015
Updated every 90 seconds
Nagios® Core™ 3.5.1 - www.nagios.org
Logged in as *nagiosadmin*

View History For all hosts
View Notifications For All Hosts
View Host Status Detail For All Hosts

Host Status Totals

Up	Down	Unreachable	Pending
3	0	0	0

All Problems	*All Types*
0	3

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
11	0	0	0	0

All Problems	*All Types*
0	11

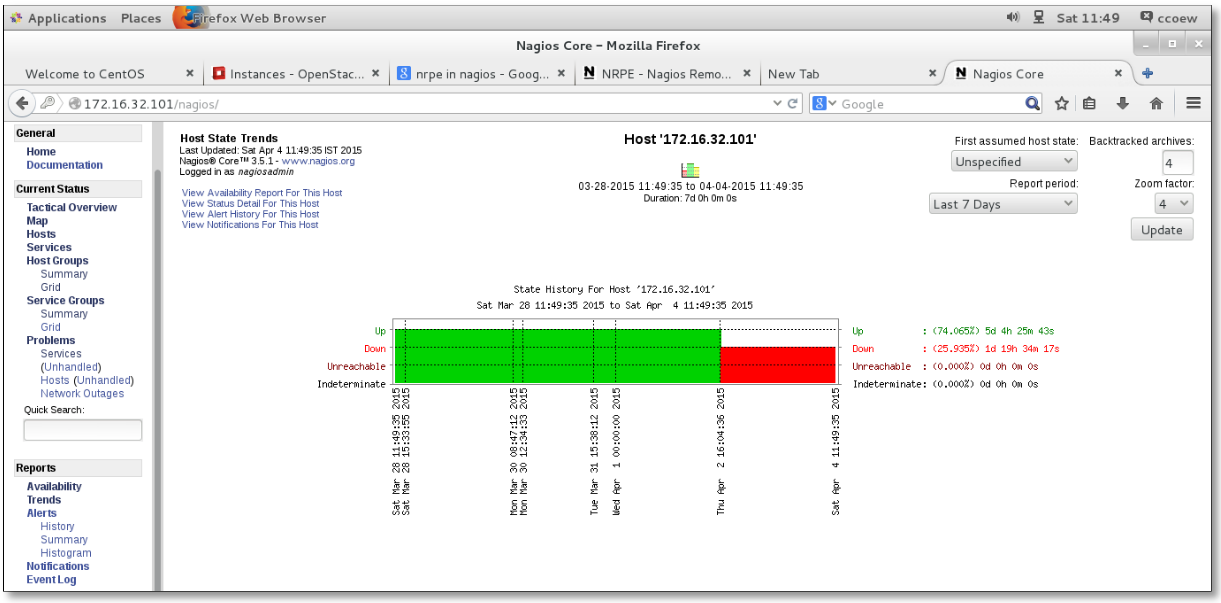
Service Status Details For All Hosts

Limit Results: 100

Host	Service	Status	Last Check	Duration	Attempt	Status Information
172.16.32.101	5 minute load average	OK	04-04-2015 11:48:12	5d 2h 47m 57s	1/3	0.57
	Percent disk space used on /var	OK	04-04-2015 11:44:56	15d 19h 24m 33s	1/3	11
	number of cinder volumes	OK	04-04-2015 11:46:40	15d 19h 30m 31s	1/3	0
	number of glance images	OK	04-04-2015 11:48:24	5d 2h 46m 34s	1/3	2
	number of keystone users	OK	04-04-2015 11:48:46	15d 19h 30m 43s	1/3	8
	number of nova vm instances	OK	04-04-2015 11:45:30	0d 1h 9m 39s	1/3	1
	number of swift containers	OK	04-04-2015 11:47:15	15d 18h 33m 10s	1/3	1
172.16.32.102	5 minute load average	OK	04-04-2015 11:43:59	6d 19h 46m 44s	1/3	0.00
	Percent disk space used on /var	OK	04-04-2015 11:44:21	5d 2h 46m 0s	1/3	10
172.16.32.103	5 minute load average	OK	04-04-2015 11:46:05	0d 1h 14m 4s	1/3	0.06
	Percent disk space used on /var	OK	04-04-2015 11:47:49	0d 1h 12m 20s	1/3	9

Results 1 - 11 of 11 Matching Services

Figure 10.4 Nagios Showing Service Status Details



Host State Trends
Last Updated: Sat Apr 4 11:49:35 IST 2015
Nagios® Core™ 3.5.1 - www.nagios.org
Logged in as *nagiosadmin*

Host '172.16.32.101'
03-28-2015 11:49:35 to 04-04-2015 11:49:35
Duration: 7d 0h 0m 0s

First assumed host state: Unspecified
Backtracked archives: 4
Report period: Last 7 Days
Zoom factor: 4

```
State History For Host '172.16.32.101'
Sat Mar 28 11:49:35 2015 to Sat Apr  4 11:49:35 2015

Up           : (74.065%) 5d 4h 25m 43s
Down         : (25.935%) 1d 19h 34m 17s
Unreachable  : (0.000%) 0d 0h 0m 0s
Indeterminate: (0.000%) 0d 0h 0m 0s
```

Figure 10.5 Nagios Generating Graphs and Reports

This way the proposed system has been implemented to achieve Operational or more specifically Energy Efficiency in Cloud, making way for the Green Computing.

PART IV

TESTING PHASE

Testing is an integral part of any system or project. If a system is implemented without being tested it may lead to an erroneous working and dissatisfaction on part of the customer. It will also prove disastrous to the reputation of the organization or the person who developed the system and lead to loss in business.

As human beings are prone to commit error under different working conditions, a system should be tested keeping in view the different possibilities on part of user.

Chapters in this part of the report highlight different types of testing strategies and test case used to test the system.

Further, the chapter also throws light upon the cost estimation of the Project undertaken.

CHAPTER 11

TESTING AND COST ESTIMATION

In order to make a system full proof, testing is an essential phase in the software life cycle. This chapter throws light on the different strategies and test cases designed to test the system. Further, it includes a section on the Cost Estimation of the project.

11.1 TESTING STRATEGIES

An overall three steps strategy has been used to test a system:

1. **Task Testing**

   This is the first step to test each task independently. That is, white box and black box tests are designed and executed for each task. Task testing uncovers errors in logic and function but not in timing or behavior.

2. **Behavioral Testing**

   It is possible to simulate the behavior of a system and examine its behavior as a consequence of external events. Events are categorized for testing. The behavior of the s/w is examined to detect the behavioral errors.

3. **Inter-task testing**

   Once errors in individual tasks and in system behavior have been isolated, testing shifts to time related errors. Asynchronous that are known to communicate with each other are related with different data rates and processing load to determine if inter-task synchronization errors will occur.

The various objectives of testing are

1. To uncover errors in function logic or implementation for the software.

2. To verify that the software needs the specific requirement.

3. To verify that software has been implemented according to the predefined standards.

4. To ensure customer satisfaction, enhance business and set a good reputation for the software developer.

DIFFERENT TYPES OF TESTING STRATEGIES

11.1.1 WHITE BOX TESTING

A level of white box test coverage is specified that is appropriate for the software being tested. The white box and other testing uses automated tools to instruct the software to measure test coverage. Using White Box Testing methods, the software engineer can derive test cases that: Guarantees that all independent paths in the module have been exercised at least once. Exercise logical decision on their true or false sides. Execute all loops at their boundaries and within their operational bounds. Exercise the internal data structure to ensure their validity.

11.1.2 BLACK BOX TESTING

A Black Box test of integration builds includes functional, interface, error recovery, stress and out-of-bounds input testing. All black box software tests are traced to control environments. In addition to static requirements, a black box of a fully integrated system is tested against sequence of events designed to model field operation. Performance testing for system is integrated as an integral part of black box test process.

11.1.3 FUNCTIONAL TESTING

Functional testing may be defined as the verification and validation of an individual module or 'function' of software. It is the most "micro" scale of testing for testing particular functions of code modules. Function testing may require developing test driver modules or test harnesses. In addition, function testing often requires detailed knowledge of the internal program design.

11.1.4 ALPHA AND BETA TESTING

Before alpha and beta testing, software is completely assembled as a package. Interface errors have been uncovered and corrected, and final series of software test may begin. Alpha testing is testing which is done by the developer at developer's site. Beta testing is testing which is done by the user or clients at their own site.

11.1.5 MISCELLANEOUS TESTING

- Unit Testing

Individual Components are tested independently to ensure their quality. The focus is to uncover errors in Design and Implementation, including

- Data structure in component
- Program logic and Program structure in a component
- Component Interface
- Functions and Operation of a component.

➢ Integration Testing

A group of dependent component are tested together to ensure the quality of their Integration Unit. The focus is to uncover errors in:

- Design and Construction of software Architecture
- Integrated functions or operations at sub-system level
- Interface and interaction and/or environment Integration

➢ Validation Testing

The system software is tested as whole. It verifies that all the elements mesh properly to make sure that all system functions and performance is achieved in the target environment. The focus areas are :

- System function and Performance.
- System Reliability and Recoverability (Recover Test).
- System Behaviors in Special Conditions (Stress and load test).
- System User Operations (acceptance test/ alpha test).
- Hardware and Software Integration Collaboration.
- Integration of external software and the system.

11.2 TEST CASES

Test Case Name	Description	Prerequisites	Steps	Input	Expected Output	Actual Output
Login1.1	Login to openstack	Openstack installed on controller Host	Enter URL in browser Username and password	Correct Username and password	Login successful	Login successful

Login1.2	Login to openstack	Openstack installed on controller Host	Enter URL in browser Username and password	Incorrect Username and password	Login failed	Login Failed
LaunchVM_1.1	Create VM (Spreading VM's across Hosts)	Services of Hosts must be up(set +ve value in RAM.py) Resources must be available to launch	Click on launch instance	Instance name, flavour, boot option	Instance launched (with spreading across Hosts)	Instance launched and scheduled
LaunchVM_1.2	Create VM (Stacking VM's on a single Host)	Services of Hosts must be up(set -ve value in RAM.py) Resources must be available to launch	Click on launch instance	Instance name, flavour, boot option	Instance launched(with stacking on a single Host)	Instance launched and scheduled
Terminate VM	Delete vm from pool of VM's	VM should be created first	Click on terminate vm and confirm termination	Select VM name to terminate	VM should be terminated and removed from pool	Termination successful
Start VM	Start vm when it is in shut down state	VM should be in shut down state	Select and click on start vm	Start vm	VM should started	VM started
Shut Down Vm	Shut down vm when it is in running state	VM must be in running state	Click on shutdown vm	Select and shut down vm	VM should stopped	VM stopped
Migrate VM_1	Migrate vm from one host to other host resources available)	There should be available host to migrate vm	Select vm and migrate	Select and migrate vm	VM should be migrate	VM migrated
Migrate VM_2	Migrate vm from one host to other host resources unavailable)	There should not be available host to migrate vm	Select vm and migrate	Select and migrate vm	VM should not be migrated	VM not migrated
Live Migrate VM_1	Lively Migrate vm from one host to other	There should be available host to migrate vm	Select vm and Live migrate	Select and Live migrate vm	VM should be lively migrated	VM not migrated

	host resources available)			specify host name on which to migrate vm		
Live Migrate VM_2	Lively Migrate vm from one host to other host resources unavailable)	There should not be available host to migrate vm	Select vm and Live migrate	Select and Live migrate vm specify host name on which to migrate vm	VM should not be lively migrated	VM not migrated

Table 11.1 Test Cases

11.3 COST ESTIMATION

Software cost estimation is the process of predicting amount of effort required to build a software system. Size is the primary cost factor in most models and can be measured using lines of codes or function points.

The most commonly used measure of source code program length is the number of lines of code. NCLOC is used to represent non commented line of code. NCLOC is also sometimes referred to as effective lines of code. NCLOC is therefore a measure of the uncommented length.

The commented length is also a valid measure, depending on whether or not line documentation is considered to be part of programming effort. The abbreviation CLOC is used to represent a commented source line of code.

By measuring NCLOC and CLOC separately we can define:

Total length (LOC) = NCLOC + CLOC

KLOC is used to denote thousands of lines of code.

The constructive cost model is an example of regression model used for estimating software cost and effort. These methods use a basic formula with parameters that are determined via a historical database and current project characteristics.

The basic COCOMO model computes effort as a function of program size.

The basic COCOMO equation is

$$\boxed{K = a\ KLOC \wedge b}$$

Where,

E = effort in person /month
a, b = cost drivers
KLOC = Program size

For our project, the values are

A = 2.4
B = 1.05
E = 2.4 * (KLOC ^1.05)

Efforts estimated = 1637 person/month
Project Duration = 8 months (240 days)
Working hours per head per day = 3 hours
Total working hours per head = 240 *3 hours
In terms of person days (240*3)/8=90 person days

Number of team members = 4

Hence,

Total no of person days = 90*4 = 360
In terms of person months = 360/30 person months

Hence,

Actual efforts =12 person months

CHAPTER 12

CONCLUSION

This last chapter summarizes the need for Green Deployments along with the work done to achieve operational energy efficiency in cloud. It also presents our vision for the future of Green Computing.

Cloud computing has made the vision of computing resources as a utility another step closer to the reality. As the technology advances and network access becomes faster and with lower latency, the model of delivering computing power remotely over the Internet will proliferate. Therefore, Cloud data centers are expected to grow and accumulate a larger fraction of the world's computing resources. In this context, energy efficient management of data center resources is a crucial problem in regard to both the operating costs and CO_2 emissions to the environment.

Here the need for an effective energy management system in order to reduce the carbon footprints in the cloud and thereby improve the performance of the cloud based services has been suggested. A system to achieve operational efficiency in virtual machine by migration of virtual machines depending on their utilization pattern has been proposed.

This project work shows the simulation of a small energy efficient system comprising of a controller node and compute nodes. It monitors the usage and utilization pattern of the VM's and sets a standard threshold value. This threshold value is used to filter the hosts which are suitable for VM deployment. Based on the result of the filter the VM scheduler migrates the VM to the host which best fits the criteria.

Various parameters which would affect the decision to migrate VMs from one host to another so as to achieve the best tradeoff between performance and energy efficiency also have been studied.

In future, this work can be extrapolated to develop energy efficient algorithms for VM placement taking into consideration the amalgamation of all the parameters affecting the migration decision. These algorithms can then be used to design and implement the energy efficient cloud systems heralding a major advancement in the trend of Green Computing.

APPENDIX-I

A1. STEPS TO CONFIGURE OPENSTACK

Step 0: Prerequisites

- **Set up ssh**

  Make sure that you can ssh as root to all your hosts from wherever you're running packstack.

  On Controller:

  ```
  # ssh-keygen (Hit Enter to accept all of the defaults)
  # ssh-copy-id -i ~/.ssh/id_rsa.pub  root@compute
  ```

  Result: This makes passwordless login from controller to compute

Step 1: Software repositories

- **Update your current packages:**

  ```
  sudo yum update -y
  ```

- **Setup the RDO repositories:**

  yum install https://repos.fedorapeople.org/repos/openstack/openstack-juno/rdo- release-juno-1.noarch.rpm

Step 2: Configure the Security Policies

CloudStack does various things which can be blocked by security mechanisms like SELinux. These have to be disabled to ensure the Agent has all the required permissions.

Set the SELINUX variable in /etc/selinux/config to "permissive". This ensures that the permissive setting will be maintained after a system reboot.

```
vi /etc/selinux/config
```

Change the following line

```
SELINUX=enforcing
```

to SELINUX=permissive

Step 3: Install Packstack Installer

```
sudo yum install -y openstack-packstack
```

Step 4: Run Packstack to install OpenStack

- **Genreate answer file** :

  ```
  packstack --gen-answer-file packstack-answers.txt
  ```

- **Make changes in the answer file :**

  i. CONFIG_NOVA_COMPUTE_HOSTS=172.16.32.105, 172.16.32.106

  ii.CONFIG_NOVA_NETWORK_PUBIF=eno1
  CONFIG_NOVA_NETWORK_PRIVIF=eno1

  iii. CONFIG_NOVA_COMPUTE_PRIVIF=eno1

  iv. CONFIG_NEUTRON_OVS_TUNNEL_IF=eno1

- **Save The file**
- **Run the file :**

  ```
  packstack ---answer-file=./ packstack-answers.txt
  ```

After these steps the openstack is succesfully installed.

STEPS TO ENABLE MIGRATION OF VMS

- **Create nova ssh between compute nodes**

  ```
  usermod -s /bin/bash nova

  $ su - nova

  ~$ mkdir -p -m 700 .ssh

  ~$ cat > .ssh/config <<EOF
  Host *
  StrictHostKeyChecking no
  UserKnownHostsFile=/dev/null
  ```

```
EOF
:~$ ssh-keygen -f id_rsa -b 1024 -P ""

:~$ scp /var/lib/nova/.ssh/id_rsa.pub
root@compute01:/var/lib/nova/.ssh/authorized_keys

 :~$scp /var/lib/nova/.ssh/id_rsa.pub
root@compute02:/var/lib/nova/.ssh/authorized_keys


:~$ scp /var/lib/nova/.ssh/* root@compute02:/var/lib/nova/.ssh/
```

STEPS TO CONFIGURE OBJECT STORAGE

- Download openstack-nova-2014.2.1-1.el7.src.rpm

```
# wget http://pbrady.fedorapeople.org/copr/openstack-nova-2014.2.1-
1.el7.src.rpm
```

- Then as root:-

```
# rpm -iv openstack-nova-2014.2.1-1.el7.src.rpm

# yum-install rpm-build

# cd; cd rpmbuild/SPEC;

# yum install python-sphinx python-oslo-sphinx python-d2to1 graphviz

# rpmbuild -bb  openstack-nova.spec

# cd ../RPMS/noarch

# yum install openstack-nova-objectstore

# systemctl enable openstack-nova-objectstore

# systemctl start openstack-nova-objectstore
```

STEPS FOR VNC CONSOLE

- iptables -A INPUT -p tcp -m multiport --dports 6080 -m comment --comment "console incoming" -j ACCEPT]
- iptables-save > /etc/sysconfig/iptables

Opening the port 6080 and adding it to iptable as VNC console work with this port.

A2. STEPS TO CONFIGURE MIGRATION IN OPENSTACK USING KVM-LIBVIRT

A2.1 SHARED STORAGE

Prerequisites

- **Hypervisor:** KVM with libvirt
- **Shared Storage:** *NOVA-INST-DIR*/instances/ (for example, /var/lib/nova/ instances) has to be mounted by shared storage. This guide uses NFS.
- **Instances:** Instance can be migrated with iSCSI-based volumes.

Example Compute installation environment

- Prepare at least three servers. In this example, we refer to the servers as HostA, HostB, and HostC:
  - HostA is the *Cloud Controller*, and should run these services: nova-api, nova-scheduler, nova-network, cinder-volume, and nova-objectstore.
  - HostB and HostC are the *compute nodes* that run nova-compute.

  Ensure that *NOVA-INST-DIR* (set with state_path in the nova.conf file) is the same on all hosts.

- In this example, HostA is the NFSv4 server that exports *NOVA-INST-DIR*/instances directory. HostB and HostC are NFSv4 clients that mount HostA.

Configuring your system

1. Configure your DNS or /etc/hosts and ensure it is consistent across all hosts. Make sure that the three hosts can perform name resolution with each other. As a test, use the **ping** command to ping each host from one another.

   ```
   $ ping HostA
   $ ping HostB
   $ ping HostC
   ```

2. Ensure that the UID and GID of your Compute and libvirt users are identical between each of your servers. This ensures that the permissions on the NFS mount works correctly.
3. Export *NOVA-INST-DIR*/instances from HostA, and ensure it is readable and writable by the Compute user on HostB and HostC.

4. Configure the NFS server at HostA by adding the following line to the /etc/exports file:

   ```
   NOVA-INST-DIR/instances
   HostA/255.255.0.0(rw,sync,fsid=0,no_root_squash)
   ```

   Change the subnet mask (255.255.0.0) to the appropriate value to include the IP addresses of HostB and HostC. Then restart the NFS server:

   ```
   # /etc/init.d/nfs-kernel-server restart
   # /etc/init.d/idmapd restart
   ```

5. On both compute nodes, enable the 'execute/search' bit on your shared directory to allow qemu to be able to use the images within the directories. On all hosts, run the following command:

   ```
   $ chmod o+x NOVA-INST-DIR/instances
   ```

6. Configure NFS on HostB and HostC by adding the following line to the /etc/fstab file:

   ```
   HostA:/ /NOVA-INST-DIR/instances nfs4 defaults 0 0
   ```

   Ensure that you can mount the exported directory:

   ```
   $ mount -a -v
   ```

   Check that HostA can see the *NOVA-INST-DIR*/instances/" directory:

   ```
   $ ls -ld NOVA-INST-DIR/instances/
   drwxr-xr-x 2 nova nova 4096 2012-05-19 14:34 nova-install-dir/instances/
   ```

   Perform the same check on HostB and HostC, paying special attention to the permissions (Compute should be able to write):

   ```
   $ ls -ld NOVA-INST-DIR/instances/
   drwxr-xr-x 2 nova nova 4096 2012-05-07 14:34 nova-install-dir/instances/
   $ df -k
   Filesystem     1K-blocks     Used    Available   Use%  Mounted on
   /dev/sda1      921514972   4180880  870523828   1%    /
   none           16498340      1228   16497112    1%    /dev
   none           16502856         0   16502856    0%    /dev/shm
   none           16502856       368   16502488    1%    /var/run
   none           16502856         0   16502856    0%    /var/lock
   none           16502856         0   16502856    0%    /lib/init/rw
   HostA:         921515008 101921792 772783104   12%   /var/lib/nova/instances
   ```

7. Update the libvirt configurations so that the calls can be made securely. These methods enable remote access over TCP and are not documented here.
   - SSH tunnel to libvirtd's UNIX socket
   - libvirtd TCP socket, with GSSAPI/Kerberos for auth+data encryption
   - libvirtd TCP socket, with TLS for encryption and x509 client certs for authentication
   - libvirtd TCP socket, with TLS for encryption and Kerberos for authentication

   Restart libvirt. After you run the command, ensure that libvirt is successfully restarted:

   ```
   # stop libvirt-bin && start libvirt-bin
   $ ps -ef | grep libvirt
   root 1145 1 0 Nov27 ? 00:00:03 /usr/sbin/libvirtd -d -l
   ```

8. Configure your firewall to allow libvirt to communicate between nodes.

   By default, libvirt listens on TCP port 16509, and an ephemeral TCP range from 49152 to 49261 is used for the KVM communications. Based on the secure remote access TCP configuration you choose be careful which ports you open, and always understand who has access.

9. You can now configure options for live migration. In most cases, you will not need to configure any options. The following chart is for advanced users only.

10. By default, the Compute service does not use the libvirt live migration function. To enable this function, add the following line to the nova.conf file:

    ```
    live_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRAT
    E_PEER2PEER,VIR_MIGRATE_LIVE, VIR_MIGRATE_TUNNELLED
    ```

    The Compute service does not use libvirt's live migration by default because there is a risk that the migration process will never end. This can happen if the guest operating system uses blocks on the disk faster than they can be migrated.

A2.2 BLOCK MIGRATION

Configuring KVM for block migration is exactly the same as the above configuration, except that *NOVA-INST-DIR*/instances is local to each host rather than shared. No NFS client or server configuration is required.

REFERENCES

REFERENCE PAPERS

[1] Rajkumar Buyya, Anton Beloglazov and Jemal Abawajy,”Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges ”.

[2] Anton Beloglazov and Rajkumar Buyya, "OpenStack Neat: A Framework for Dynamic and Energy-Efficient Consolidation of Virtual Machines in OpenStack Clouds", Concurrency and Computation: Practice and Experience (CCPE), John Wiley & Sons, Ltd, USA, 2014 (in press, accepted on 19/05/2014).

[3] Anton Beloglazov, "Energy-Efficient Management of Virtual Machines in Data Centers for Cloud Computing", PhD Thesis, The University of Melbourne, 2013.

[4] Michael R. Hines, Kartik Gopalan, “Post-copy based live VM migration using adaptive pre-paging and dynamic selfballooning,” In Proceedings of ACM SIGPLAN/SIGOPS international conference on Virtual execution environments(VEE), pages 51-60, New York, USA, 2009.

[5] E. Pinheiro, R. Bianchini, E. V. Carrera, T. Heath, “Load balancing and unbalancing for power and performance in cluster-based systems,” In Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP),pp. 182–195,2001.

[6] Liang-The Lee, Kang-Yuan Liu and Chia-Ying Tseng, “A dynamic resource management with energy saving Mechanism for storing Cloud Computing,” International Journal of Gridand Distributed Computing,Vol-6 Feb,2013.

[7] Kejiang Ye, Dawei Huang, Xiaohong Jiang, Huajun Chen, Shuang Wu, “Virtual Machine Based Energy-Efficient Data Center Architecture for Cloud Computing: A Performance Perspective” in [International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing], [2010][IEEE]. DOI :10.1109 pp.171-178.

[8] Anton Beloglazov and Rajkumar Buyya, “Energy Efficient Resource Management in Virtualized Cloud Data Centers” in [10th International Conference on Cluster, Cloud and Grid Computing], [2010][IEEE]. DOI :10.1109 pp.826-831.

[9] Zhen Xiao, *Senior Member, IEEE, Weijia Song, and Qi Chen* “Dynamic Resource Allocation using Virtual Machines for Cloud Computing Environment” in [IEEE Transaction on Parallel and Distributed Systems(TPDS), Vol. N, No. N, Month Year] pp.1-11.

[10] R.Giridharan,“Efficient Resources Allocation and Reduce Energy Using Virtual Machines for Cloud Environment” in [Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT’14)],[Vol.2, Special Issue 1, March 2014] [IIJIRCCE] pp. 133-138.

[11] T. Kamalakar Raju, A. Lavanya, Dr. M. Rajanikant h “Virtualization Technology using Virtual Machines for Cloud Computing ”, Mar. 2014 IJMER pp.7-11.

[12] Inderjit Singh Dhanoa, Dr. Sawtantar Singh Khurmi ,“Energy-Efficient Virtual Machine Live Migration in Cloud Data Centers” ,IJCST Vol. 5, SPL - 1, Jan - March 2014.

[13] Pradip D. Patel,Miren Karamta, M. D. Bhavsar, M. B. Potdar, “ Live Virtual Machine Migration Techniques in Cloud Computing: A Survey”, *January 2014*

[14] Gergo L., N.Florian and Hermann de Meer , “PerformanceTradeoffs of Energy-Aware VM Consolidation ,” Journalof Cluster Computing(Springer), Volume 16, Issue 3 , pp 481-496, 2013.

[15] Yichao Jin, Yonggang Wen , “Energy Efficient and Server Virtualization in Data Centers: An Empirical Investigation,” IEEE conference on Computer Communications Workshops (INFOCOM WKSHPS), Orlando, FL,2012.

[16] R. Nathuji and K. Schwan, “VirtualPower: Coordinated power management in virtualized enterprise systems,”ACM SIGOPS Operating Systems Review, vol. 41, no. 6, pp. 265–278, 2007.

[17] Y. Song, H. Wang, Y. Li, B. Feng, and Y. Sun, “Multitiered on-demand resource scheduling for VM-based datacenter,” In Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2009), Shanghai, China, pp. 148–155,2009.

[18] Manasa H.B, B.Anirban, “Energy Aware Resource Allocation in Cloud Data Centers,” International Journal of Engineering and Advanced Technology, Volume-2, Issue-5, June 2013.

[19] Jing SiYuan, “A Novel Energy Efficient Algorithm for Cloud Resource Management,” International Journal of Knowledge and Language Processing, Volume 4, Number 2, 2013.

[20] H. Liu, C.-Z. Xu, H. Jin, J. Gong, X. Liao, “Performance and energy modeling for live migration of Virtual Machines,” In Proceedings of the 20th international symposium on High performance distributed computing, ser. HPDC ’11. New York, NY, USA: ACM, pp. 171–182, 2011.

[21] Kejiang Ye, Xiaohong Jiang, Dawei Huang, Jianhai Chen, Bei Wang,“Live Migration of Multiple Virtual Machines with Resource Reservation in Cloud Computing Environments,” Cloud Computing (CLOUD), 2011 IEEE International Conference, pp.267-274, 4-9 July 2011.

[22] Takahiro Hirofuchi et.al, “Making VM Consolidation More Energy-efficient by Post-Copy Live Migration,”International Conference on Cloud Computing, Grids and Virtulization,2011.

URL REFERENCES:

[1] OpenStack Training Guide, Available: http://docs.openstack.org/icehouse/training-guides/content/operator-getting-started.html, DOA: August 2014

[2] Introduction of Cloud computing, Available: http://www.oracle.com/us/technologies/cloud/oracle-cloud-computing-wp-076373.pdf, DOA: August 2014

[3] Virtualization, Available: http://en.wikipedia.org/wiki/Virtualization, DOA: August 2014

[4] Software Development Life Cycle, Available: http://www.techopedia.com/definition/22193/software-development-life-cycle-sdlc, DOA: August 2014

[5] Software Development Life Cycle Phases, Available: http://istqbexamcertification.com/what-are-the-software-development-life-cycle-sdlc-phases/, DOA: August 2014

[6] OpenStack Introduction, Available: http://opensource.com/resources/what-is-openstack, DOA: September 2014

[7] Standardization of OpenStack cloud computing, Available: http://www.redhat.com/en/insights/openstack, DOA: September 2014

[8] Writing Openstack Automation Scripts with Python Bindings, Available: http://www.ibm.com/developerworks/cloud/library/cl-openstack-pythonapis/, DOA: February 2015

[9] Migrating Instances Between Compute Nodes, Available: https://docs.joyent.com/sdc7/managing-instances/migrating-instances-between-compute-nodes, DOA: October 2014

[10] Compute API v2.1 Available: http://developer.openstack.org/api-ref-compute-v2.1.html, DOA: January 2015

[11] Client Challenges for Infrastructure APIs, Available: https://www.tildedave.com/2014/04/02/client-challenges-for-infrastructure-apis.html, DOA: January 2015

[12] Internap OpenStack API Getting Started Guide, Available: http://downloads.internap.com/api-getting-started-guide.pdf, DOA: January 2015

[13] Openstack VM live migration ,Available: https://kimizhang.wordpress.com/2013/08/26/openstack-vm-live-migration/, DOA: March 2015

[14] Openstack Nova Internals Of Instance Launching, Available : http://www.laurentluce.com/posts/openstack-nova-internals-of-instance-launching/, DOA: Febrauary 2015

[15] OpenStack nova-scheduler and its algorithm, Available: http://williamherry.blogspot.in/2012/05/openstack-nova-scheduler-and-its.html, DOA: January 2015

[16] Nova scheduler concepts, Available: http://www.ovirt.org/images/c/cc/Scheduling_fosdem.pdf, DOA: January 2015

[17] OpenStack Nova-Scheduler And vSphere DRS, Available : http://www.rackspace.com/blog/openstack-nova-scheduler-and-vsphere-drs/, DOA:January 2015

[18] Discover OpenStack: The Compute components Glance and Nova, Available: http://www.ibm.com/developerworks/cloud/library/cl-openstack-nova-glance/,, DOA:January 2015

[19] IBM Cloud manager with OpenStack, Available: https://www.ibm.com/developerworks/servicemanagement/cvm/sce/i/IBMCloudManagerwithOpenStack.pdf , DOA:January 2015

[20] Introduction to OpenStack and its components, Avialable: https://cssoss.wordpress.com/2011/04/28/openstack-beginners-guide-for-ubuntu-11-04introduction-to-openstack-and-its-components/, DOA: January 2015

[21] Coverage on IBM SDE Product Announcement: Platform Resource Scheduler, Available: https://storify.com/chaitisen/ibm-sde-product-announcement-platform-resource-sch, DOA: January 2015

[22] OpenStack End User Guide, Available: http://docs.openstack.org/user-guide/content/sdk_compute_apis.html, DOA: March 2015

[23] OpenStack installation Guide, Available: http://docs.openstack.org/juno/install-guide/install/yum/content/, DOA: December 2014

[24] Compute Node With Packstack And Existing Network, Available:

https://ask.openstack.org/en/question/55009/only-one-compute-node-with-packstack-and-existing-network/, DOA: December 2014

[25] Adding a compute node, Available: https://ask.openstack.org/en/question/53937/add-a-compute-node-in-juno/, DOA: December 2014

[26] RDO Juno Set up Two Real Node (Controller+Compute) Gluster 3.5.2 Cluster ML2&OVS&VXLAN on CentOS 7, Available: http://bderzhavets.blogspot.in/2014/11/rdo-setup-two-real-node.html, DOA: December 2014

[27] Monitoring Common Services using Nagios, Available: http://www.yoyoclouds.com/2014/07/monitoring-common-services-using-nagios.html,DOA:February2015

[28] Nagios Settings, Available: http://www.server-world.info/en/note?os=CentOS_6&p=nagios&f=2, DOA: February 2015

BOOK REFERENCES

[1] Judith Hurwitz and Robin Bloor and Marcia Kaufman and Dr.Fern Halper," *Cloud Computing for Dummies* ",Wiley India Pvt.Ltd.

[2] Barrie Sosinsky,"*Cloud Computing Bible*", Wiley India Pvt.Ltd.

[3] Dr.Kumar Saurabh,"*Cloud Computing Unleashing Next Gen Infrastructure to Application*",third edition.

[4] Anthony T.Velte and Toby J.Velte and Robert Elsenpeter,"*Cloud Computing : A Practical Approach*",Tata McGRAW-HILL edition.

[5] Wesley J.Chun,"*Core Python Programming*",Pearson Education,second edition.

[6] Dusty Philips,"*Python 3: Object Oriented Programming*",[PACKT] open source.