



Deliverable No. 5.3

Techniques to build the cloud infrastructure available to the community

Grant Agreement No.: 600841
Deliverable No.: D5.3
Deliverable Name: Techniques to build the cloud infrastructure available to the community
Contractual Submission Date: 31/03/2015
Actual Submission Date: 31/03/2015

Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	



COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	CHIC
Project Full Name:	Computational Horizons In Cancer (CHIC): Developing Meta- and Hyper-Multiscale Models and Repositories for In Silico Oncology
Deliverable No.:	D5.3
Document name:	Techniques to build the cloud infrastructure available to the community
Nature (R, P, D, O) ¹	R
Dissemination Level (PU, PP, RE, CO) ²	PU
Version:	1.0
Actual Submission Date:	31/03/2015
Editor: Institution: E-Mail:	Manolis Tsiknakis FORTH tsiknaki@ics.forth.gr

ABSTRACT:

This deliverable reports on the technologies, techniques and configuration needed to install, configure, maintain and run a private cloud infrastructure for productive usage.

KEYWORD LIST:

Cloud infrastructure, OpenStack, Eucalyptus, CloudStack, VMware vSphere, virtualization, computation, storage, security, architecture.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 600841.

The author is solely responsible for its content, it does not represent the opinion of the European Community and the Community is not responsible for any use that might be made of data appearing therein.

¹ R=Report, P=Prototype, D=Demonstrator, O=Other

² PU=Public, PP=Restricted to other programme participants (including the Commission Services), RE=Restricted to a group specified by the consortium (including the Commission Services), CO=Confidential, only for members of the consortium (including the Commission Services)

MODIFICATION CONTROL			
Version	Date	Status	Author
0.1	30/10/2014	Draft	Dimitris Counalakis
0.2	12/01/2015	Draft	Giorgos Zacharioudakis
0.3	10/03/2015	Draft	Giorgos Zacharioudakis
0.4	24/03/2015	Draft	Giorgos Zacharioudakis
0.5	27/03/2015	Draft	Xia Zhao
0.6	30/03/2015	Pre-final	Giorgos Zacharioudakis
1.0	31/03/2015	Final	Giorgos Zacharioudakis

List of contributors

- Dimitris-Nektarios Counalakis, TEIC
- Manolis Tsiknakis, TEIC & FORTH
- Giorgos Zacharioudakis, FORTH
- Stelios Sfakianakis, FORTH
- Kostas Marias, FORTH
- Xia Zhao, BED
- Feng Dong, BED
- Elias Neri, CUSTODIX
- Georgios Stamatakis, ICCS

Contents

1	EXECUTIVE SUMMARY.....	5
1.1	PURPOSE OF THIS DOCUMENT	6
1.2	OVERVIEW OF THE DOCUMENT	7
2	EVALUATION AND COMPARISON OF CLOUD INFRASTRUCTURE TECHNOLOGIES	8
2.1	CLOUD TECHNOLOGIES EVALUATION	8
2.2	OPENSTACK.....	9
2.3	EUCALYPTUS	13
2.4	CLOUDSTACK	16
2.5	VMWARE VSPHERE	18
2.6	COMPARISON OF CLOUD TECHNOLOGIES.....	22
3	OPENSTACK ARCHITECTURE.....	26
3.1	INTRODUCTION	26
3.2	OBJECT STORAGE.....	28
3.3	OPENSTACK OBJECT STORE (SWIFT).....	29
3.4	SWIFT: MODEL OF OPERATION.....	30
3.5	DISCUSSION	31
4	IMPLEMENTATION OF A PRIVATE CLOUD INFRASTRUCTURE	32
4.1	INTRODUCTION	32
4.2	SOFTWARE	32
4.3	NETWORK	33
4.4	RINGS.....	35
4.5	DISCUSSION.....	37
5	SECURITY	39
5.1	INTRODUCTION	39
5.2	IDENTITY MANAGEMENT	39
5.3	AUTHORIZATION AND ACCESS CONTROL.....	40
5.4	DATA ENCRYPTION.....	41
5.5	INTEGRATION WITH CHIC SECURITY FRAMEWORK	41
6	OPENSTACK QUICK INSTALLATION GUIDE	43
7	REFERENCES	59
	APPENDIX 1 – ABBREVIATIONS AND ACRONYMS	66
	APPENDIX 2 – LIST OF TABLES AND FIGURES	69
	LIST OF FIGURES	69
	LIST OF TABLES.....	69
	APPENDIX 3 – OPENSTACK GUI OVERVIEW	70

1 Executive Summary

The CHIC project aims at developing cutting edge ICT tools, services and secure infrastructure to foster the development of elaborate and reusable integrative models (hypermodels) in the field of cancer diagnosis and treatment, as well as larger repositories so as to demonstrate benefits of having both the multiscale data and the corresponding models readily available in the VPH domain. In the course of developing these tools, both retrospective and prospective patient data will be used to test these models as well as validate them, which brings into focus the legal and ethical requirements for the processing of sensitive health data.

In Task 5.3 of the CHIC project we analysed the state of the art technologies in cloud computing in order to provide a private cloud infrastructure for the support of the CHIC platform. The deployment of this private cloud uses open source technologies with a particular focus on processing biomedical data, with all the legal and security requirements imposed by the latter. The CHIC private cloud infrastructure will support all the technical functionality of CHIC, as described in D5.1.1, including the storage of data and models, the execution of hypermodels and the execution of imaging and visualization functionality.

In this document we describe the analysis and evaluation we conducted for the selection of a cloud technology platform, with a special focus on the special legal and security requirements of CHIC. We describe and analyse the architecture of the chosen platform (Openstack) and we elaborate on the security architecture and mechanisms of Openstack in order to demonstrate how the platform fulfils the required functionality. Finally, we provide a brief installation guide with all the required techniques and steps to install, configure and maintain such as a large scale infrastructure for productive usage.

Introduction

1.1 Purpose of this document

1.1.1 Cloud Computing

The term "Cloud Computing" implies the provisioning of computing resources, both hardware and software, in the form of a service [1]. These services are characterized by certain qualities, such as availability on-demand, self-service, broad-band access, high availability and rapid elasticity. The "cloud" infrastructure technology is the evolution of traditional technologies for sharing and managing large sets of IT resources, such as data networks, data centres and computation clusters, and is usually characterized and distinguished to such technologies by the dynamic allocation.

1.1.2 Service models

In principle, we distinguish between four service models [11] :

- **Infrastructure as a Service (IaaS):** IaaS is the supply of hardware as a service, that is, servers, net technology, storage or computation, as well as basic characteristics such as Operating Systems and virtualization of hardware. Examples of commercial IaaS platforms are EducationERP.net, Amazon S3-EBS-EC2, Eucalyptus, Microsoft, Oracle Coherence, OpenStack, RightScale, 3Tera App Logic, EnStratus, Flexiscale, GoGrid CloudStatus, CampusEA.
- **Platform as a Service (PaaS):** At the PaaS level, the provider supplies more than just infrastructure; i.e. an integrated set of software with all the stuff that a developer needs to build applications, both for the developing and for the execution stages. Examples of commercially available PaaS platforms include Google App Engine, Microsoft Azure Services, Amazon SimpleDB, Microsoft SDS, Oracle Higher Education Constituent Hub, Amazon SQS and more.
- **Software as a Service (SaaS):** In this service model the provider offers software as a service and this was one of the first implementations of Cloud services. Examples of SaaS platforms include the Google App, Microsoft Dynamics CRM online, Microsoft Live@edu, Business Productivity Online Suite, Exchange Hosted Services, Microsoft Office Web Apps, CampusEAI, EducationERP.net, Campus Management, Jaspersoft, Coupa's e-Procurement,
- **Network as a Service (NaaS) [12] :** a category of cloud services where the capability provided to the cloud service user is to use network/transport connectivity services and/or inter-cloud network connectivity services. NaaS involves the optimization of resource allocations by considering network and computing resources as a unified whole.

1.1.3 Deployment models

Currently, there are three deployment models for cloud computing. Based on the location and who manages it, a cloud can be defined as:

- i. **Private cloud:** It is deployed within an organization's infrastructure and the resources are dedicated to the organization itself. Management and resource allocation are also controlled in-house by the organization.
- ii. **Public clouds:** are open to public, but resources and infrastructure if owned by the organization providing the cloud service. Although public cloud providers often ensure client's data security and integrity, data control, especially for sensitive data, can always be an issue.
- iii. **Hybrid clouds:** they combine both public and private clouds using data and application migration techniques.

1.1.4 The CHIC cloud infrastructure

In the context of the CHIC project, due to the legal and ethical restrictions resulting from the nature of managed information, such as clinical, genomic or imaging data of patients or otherwise sensitive and legally restricted data, there are specific requirements to incorporate into our architecture during the design and implementation of the CHIC technical architecture (CHIC deliverable D5.1.1).

From the technical implementation point of view, there is the need for a cloud infrastructure which will provide the computation and storage resources for storing and processing data, as well as models and hypermodels. These requirements call for a service model such as the Infrastructure as a Service (IaaS), for sharing and allocating resources for deploying CHIC software modules and services.

From the legal adherence point of view, there are security requirements which impose that the deployment model that can best serve CHIC and meet those requirements is a *private cloud*, deployed and managed within the CHIC consortium, limiting and securing access to data as well as providing audit and security mechanisms only to authorized CHIC partners.

1.2 Overview of the document

The rest of this document reports on the methodology and the techniques we have employed to build the CHIC cloud infrastructure. In chapter 2 we document the methodology for evaluating and selecting the cloud infrastructure technology platform to be selected amongst the most prominent candidates. The outcome of this evaluation was to select the OpenStack technology platform. In chapter 3 we outline the OpenStack architecture, and especially its functionality and capabilities for data storage, since the defining parameter of selecting a private cloud deployment model was the security requirements regarding data. In chapter 4 we present the implementation details and high level components of building the cloud infrastructure based on the OpenStack technology and in chapter 5 we focus on the security aspects of the cloud platform. Finally in chapter 6 we provide a quick installation guide which can serve as manual for the deployment of the cloud. In an Appendix of this report we provide also a brief overview of the graphical environment (Horizon) for the managing of common tasks within OpenStack.

2 Evaluation and comparison of cloud infrastructure technologies

2.1 Cloud technologies evaluation

Up to now, many cloud platform offerings exist, either deployed as public or private clouds. Both models have advantages and disadvantages and can be more or less suitable for a specific field, depending on the application domain to be deployed. In CHIC, the selection of the most suitable cloud deployment model had to be made in respect to the biomedical domain and the data it holds.

There were several factors that had to be taken under consideration in order to determine which cloud deployment model satisfies best the requirements of handling biomedical data. The most critical characteristic of these data emerges directly from their sensitive nature and the legal framework that governs them. Since the data in question usually derive from actual clinical data it is critical that they be under the total control and responsibility of the cloud maintainers. Although most public cloud providers offer attractive Service Agreements to their customers, there is no real guarantee that their data are not subject of processing by unauthorized third-parties. Moreover, users are not aware of the background actions performed over their data even for maintenance purposes and again, in the case of biomedical data, cloud users usually have the obligation to keep their data localized to specific geographic locations and comply with certain legal suits.

As a result, the most obvious selection is the one of a private cloud over the public cloud offerings. By deploying a private cloud infrastructure, cloud maintainers can control data access, locality and integrity and can provide guarantees for any cloud wise operation on the data.

Consequently, we opt for a cloud platform among a number of enterprise scale, open source private cloud platform candidates. The two preliminary prerequisites, enterprise and open source, are tightly coupled to the biomedical domain: data are growing rapidly being of the order of several hundreds of gigabytes or terabytes and the demand for computing power is also growing respectively to process them. On the other hand, the cost of a commercial cloud platform rises accordingly to the size of the infrastructure and more importantly, it is limited to the design guidelines of the production company. Since the cloud must serve the needs of a research facility and it will probably need heavy customization in order to comply with special requirements and workflows, the requirement for an open source solution seemed crucial.

With respect to the above, the selection was made on the basis of cloud technology adoption and usage level with preference for open source solutions, packaging and support options that each platform provides for various Linux distributions (native packages, updates and availability of software repositories), the diversity of each platform and its maturity level. Furthermore, the cloud platform must yield at least the basic service models and provide a flexible API. Licensing was another significant parameter, since the source code of the cloud platform may be altered by the maintainers or the application developers.

Considering the limited hardware resources available, it was not possible to deploy several cloud platforms simultaneously in order to provide a comparative evaluation and analysis. As a result, by comparing the packaging options and the functionality of several open source cloud platforms and the support that each vendor provides for several Linux distributions, we ended up with the four most promising platforms, namely OpenStack, Eucalyptus, CloudStack and VMware vSphere. Notably, by testing vSphere we can see the performance of open source technologies vs non-open source.

All four cloud operating systems can manage multiple cluster environments. These platforms are enterprise scale but besides the similarities between them and their common functionality, there also are some important differences and each platform has its own unique characteristics, strengths and weaknesses. For example, OpenStack provides an API for applications built on top of the cloud to

access a specialized “Object Storage”, which will be analysed later, Eucalyptus is designed with Amazon Cloud interoperability in mind, CloudStack offers a robust structure for Datacenter organization whereas vSphere provides fast installation via USB drives.

In order to designate the qualities of OpenStack, Cloudstack, Eucalyptus and VMware vSphere, a comparison of these products in terms of Storage, Virtualization, Networking, Management, Security and Community/Support criteria is needed:

- *Storage criteria:* platforms must provide connectivity to either network block devices, or via direct (object) storage. Fault tolerance and HA (high availability) configuration requirements must be met.
- *Virtualization criteria:* with this term, criteria concerning VM provisioning, resource management, migration, Hypervisor compatibility etc. are established.
- *Network criteria:* network connectivity is essential to cloud computing. A cloud platform must be capable of either flat or VLAN networking.
- *Management criteria:* a cloud platform should grant for effective management facilities in terms of logging, reporting, recovery mechanisms, high availability and hypervisors and guest OSs management and deployment.
- *Security criteria:* due to the scale and the sensitivity of data, a cloud platform should provide security mechanisms like user and key management, data encryption, auditing and reporting.
- *Maturity and support criteria:* a cloud platform can be developed in either a community based environment or inside a company. Active development and support are essential to its viability and especially for open source software, a large community of users along with the developers are critical for the welfare of the platform.
- *Performance criteria:* it is crucial that any cloud platform is evaluated for its performance, especially for storage, since it can often prove to be a bottleneck for the whole infrastructure.

In the next sections we provide a brief description of the basic concepts and building blocks of each software platform. We focus in the storage and public cloud interoperability options for each one. For public cloud interoperability, we use as metric the compatibility level they provide to one of the world’s market leader in public cloud platforms, Amazon EC2 and Amazon S3. Amazon’s EC2, is a web service that provides resizable compute capacity in the cloud [45] and Amazon S3, provide web services interface that can be used to store and retrieve data [46]. Both services are part of the Amazon WEB services.

2.2 OpenStack

2.2.1 Introduction

OpenStack is a cloud operating system which, in principle, provides IaaS. It is designed to manage data centres and has no proprietary hardware or software requirements.

OpenStack was created by NASA and Rackspace Hosting [47] back in 2010. Nowadays, a lot of hardware and software vendors have joined the project, among them Cisco, Dell, IBM, Intel and Red Hat [48].

OpenStack is available under the Apache License 2.0 [49] and there is a six-month release cycle from the OpenStack community. Before each release, the community organizes the OpenStack Design Summit [50], where developers around the globe gather to discuss the requirements for the next release and implementation details.

2.2.2 Architecture

OpenStack components are presented at **Figure 1**:

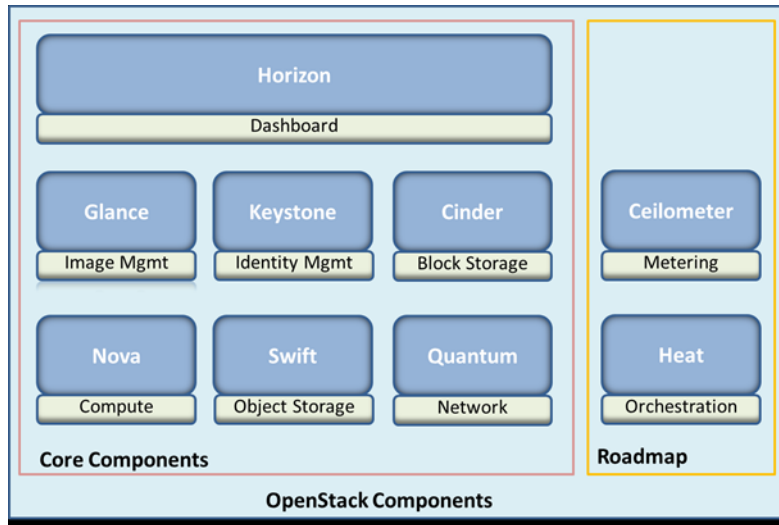


Figure 1 OpenStack components³

OpenStack is built upon nine different services, as shown in Table 1.

Service	Codename	Description
Dashboard	Horizon	The default WEB GUI for OpenStack
Compute	Nova	Handles Virtual Machine resource
Identity	Keystone	A framework for authentication and authorization
Network	Neutron	It provides “ <i>network as a service</i> ” between interface devices (e.g., vNICs) managed by other OpenStack services (e.g., nova).
Image Service	Glance	Catalogue and repository service for virtual disk images
Block Storage	Cinder	Provides block storage services to VMs (iSCSI, FC etc.)
Object Storage	Swift	Provides Object storage as a distributed storage system that can be integrated directly into applications.
Monitoring	Ceilometer	Monitors and meters the OpenStack cloud for billing, benchmarking, scalability, and statistics purposes.
Orchestration	Heat	Orchestrates multiple composite cloud applications by using the CloudFormation [51] template format, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.

Table 1 OpenStack components

³ Image copied from <http://applycloud.blogspot.gr/2013/05/OpenStack-components.html>

Below we provide an overview of three basic components of OpenStack: Storage, Networking and Compute.

2.2.2.1 Storage

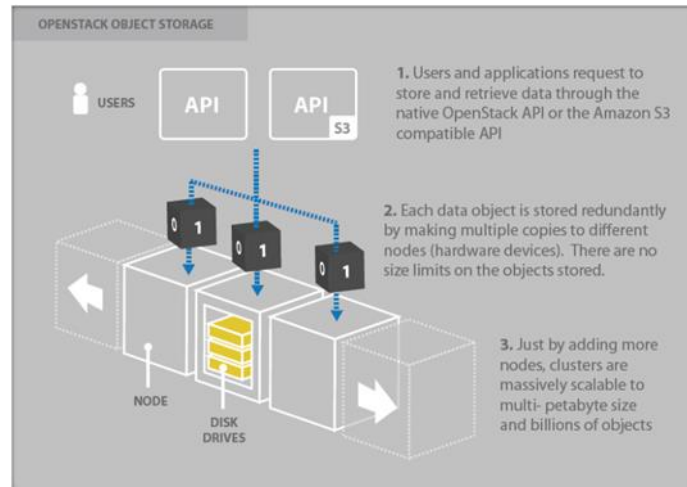


Figure 2 OpenStack Storage Model⁴

The storage options of OpenStack, are of great significance. Its object storage (namely Swift) is exposed via RESTful⁵ API, so that it can be integrated directly into applications. The OpenStack object storage, has several capabilities:

- It is provided in a distributed manner, using clusters of storage servers with characteristics of redundancy and scalability.
- It may be used as storage facility for static data, such as virtual machine images, photo storage, email storage, backups and archives.
- Objects and files are written to multiple disk drives with multiple replicas, spread throughout several servers.
- Storage clusters scale horizontally simply by adding new servers.
- In case of a server or hard drive failure, OpenStack replicates its content from other active nodes to new locations in the cluster.

On the other hand, the block storage option, provides storage over iSCSI⁶, FC or FCoE⁷ protocols. In brief:

- The block storage system manages the creation, attaching and detaching of the block devices to servers.
- It has unified storage support for various storage platforms including Ceph, NetApp, Nexenta, SolidFire, and Zadara [52].

⁴ Image copied from <http://rogerluethy.wordpress.com/2011/11/16/OpenStack-object-storage-an-overview/>

⁵ Representational State Transfer (REST) defines a set of architectural principles by which Web services are designed with focus on a system's resources, including how resource states are addressed and transferred, usually, over HTTP [139][140]

⁶ Internet Small Computer Systems Interface (iSCSI) as defined in RFC3720, "Internet Small Computer Systems Interface (iSCSI)" [141]

⁷ Fiber Channel (FC) and Fiber Channel Over Ethernet (FCoE) as referenced in "Fibre Channel: Backbone - 5 revision 2.00" [142]

- Block storage is appropriate for performance sensitive scenarios such as databases, 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.

2.2.2.2 Networking

OpenStack networking of VMs can be deployed with either flat or VLAN networking. To summarize:

- OpenStack Networking manages static or dynamic IP addresses. Traffic can be dynamically rerouted to any compute resource, which provides a level of redundancy or failover in case of failure.
- Users are able to create their own networks and thus, control traffic
- 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.

2.2.2.3 Compute

OpenStack is designed to manage and automate pools of compute resources. It takes advantage of the popular KVM [53] or Xen [54] hypervisors and it supports Intel (x86 and x86_64) [55][56], ARM [57] and alternative hardware architectures [58].

2.2.3 Amazon EC2 and Amazon S3 support

OpenStack provides API for both EC2 and S3. In Table 2 we present a view of these APIs and their compatibility level with Amazon services [59]:

EC2		S3	
Feature	Supported	Feature	Supported
EC2 Query API	Y	List Bucket Objects	Y
EC2 Soap API	N	Bucket ACLs	Y
OpenStack API / Rackspace API	Y	Bucket Lifecycle	N
SSL Between Components	N	Bucket Policy/Location	N/?
Horizontal Component Scalability	Y	Bucket Logging/Notification	N / N
Web-based UI	Y	Bucket Object Versions	Y
Command line interface	Y	Bucket Versioning	?

Table 2 OpenStack EC2 and S3 API

Although OpenStack does not fully support the Amazon WEB (AWS) services API, it provides a basic level of connectivity capable of handling virtual machine instances and storage in the Amazon Cloud.

2.3 Eucalyptus

2.3.1 Introduction

“Eucalyptus is open source private cloud software for building private and hybrid clouds that are compatible with AWS APIs. With AWS-compatibility, the open source software pools together existing virtualized infrastructure to create private or hybrid cloud resources for compute, network, and storage.” [60].

Eucalyptus is targeted into creating private and hybrid cloud environments in the sense of Amazon Web Services (AWS) [61]. It incorporates compute, network and storage resources which can be adopted to the workload.

Eucalyptus initiated as a project mainly at the Rice University and soon spread along other institutions. Recently, in September 2014, was acquired by Hewlett-Packard.

The platform is available under the GNU Public License version 3 (GPLv3) [62] licensing model with Proprietary relicensing.

2.3.2 Architecture

The structure of the Eucalyptus platform is presented at Figure 3:

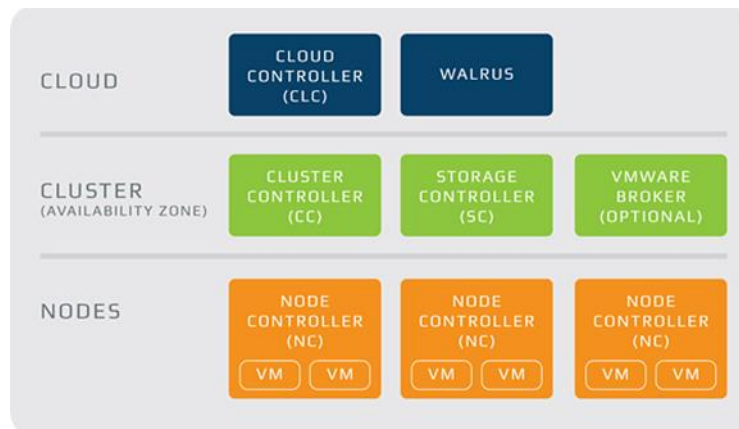


Figure 3 Eucalyptus Components⁸

As one can notice, it has three layers of abstraction, the cloud, the cluster and the nodes. This stratification differentiates the cloud operations and helps the infrastructure at the organization level. The semantics of these layers are explained later in this text.

Eucalyptus consists of the following six components shown in Table 3.

Component	Abbreviation	Description
Cloud Controller	CLC	A web based interface and EC2 interface. It handles incoming requests and provides administrative control and management of the infrastructure, as long as high-level resource scheduling and system accounting.
Walrus	–	Offers persistent storage to VMs

⁸ Picture taken from: <https://www.eucalyptus.com/eucalyptus-cloud/iaas/architecture>

Cluster Controller	CC	It acts as the front end for a cluster within a Eucalyptus cloud and communicates with the Storage Controller and Node Controller. It manages instance (i.e., virtual machines) execution and Service Level Agreements (SLAs) per cluster.
Storage Controller	SC	It communicates with the Cluster Controller and Node Controller and manages Eucalyptus block volumes and snapshots to the instances within its specific cluster.
VMware Broker	–	Provides an AWS-compatible interface for VMware (VMware, n.d.) environments and physically runs on the Cluster Controller
Node Controller	NC	Hosts the virtual machine instances and manages the virtual network endpoints

Table 3 Eucalyptus components

The four basic components of Eucalyptus are *Cloud Controller*, *Walrus*, *Cluster Controller* and *Storage Controller*.

2.3.2.1 Cloud Controller

Cloud Controller offers EC2 compatible SOAP and Query interfaces. It acts as the administrative interface for cloud management and performs high-level resource scheduling and system accounting. CLC, handles Authentication, Accounting, Reporting and Quota Management.

2.3.2.2 Walrus

Is the equivalent to AWS Simple Storage Service (S3). It offers persistent storage to VMs and can be used as a simple HTTP put/get Storage-as-a-Service solution. It does not have any data restrictions.

2.3.2.3 Cluster Controller

Acts as the front end for a cluster within a Eucalyptus cloud and communicates with the Storage Controller (SC) and Node Controller (NC). The CC manages instance (i.e., virtual machines) execution and Service Level Agreements (SLAs) per cluster.

2.3.2.4 Storage Controller

It is the equivalent to AWS Elastic Block Store (EBS). The SC communicates with the Cluster Controller (CC) and Node Controller (NC) and manages Eucalyptus block volumes and snapshots to the instances within its specific cluster.

2.3.3 Amazon EC2 and Amazon S3 support

Eucalyptus implements by design the Amazon specifications for EC2 and S3. It provides REST and SOAP interface compatible with Amazon AWS [64]. Moreover, Eucalyptus is part of the AWS partner network [65]. Eucalyptus provides full connectivity to Amazon EC2 and S3 with native API [66].

In brief, these are the AWS features supported by Eucalyptus:

- Amazon Elastic Compute Cloud (EC2)
- Amazon Elastic Block Storage (EBS)
- Amazon Machine Image (AMI)
- Amazon Simple Storage Service (S3)
- Amazon Identity and Access Management (IAM)
- Auto Scaling

- Elastic Load Balancing
- Amazon CloudWatch

2.3.4 Environment setup for feasibility tests

This section presents the evaluation that has been carried out independently based on a small scale Eucalyptus cloud established at the site of BED. Figure 4 Computer nodes for the deployment of Eucalyptus shows the configuration of the deployment. At the time of the deployment, Ubuntu 12.04 is used as operating system, Xen 4.1 is used as hypervisor on top of Ubuntu for virtualisation, and Eucalyptus 3.1 open source is used as the private cloud middleware.

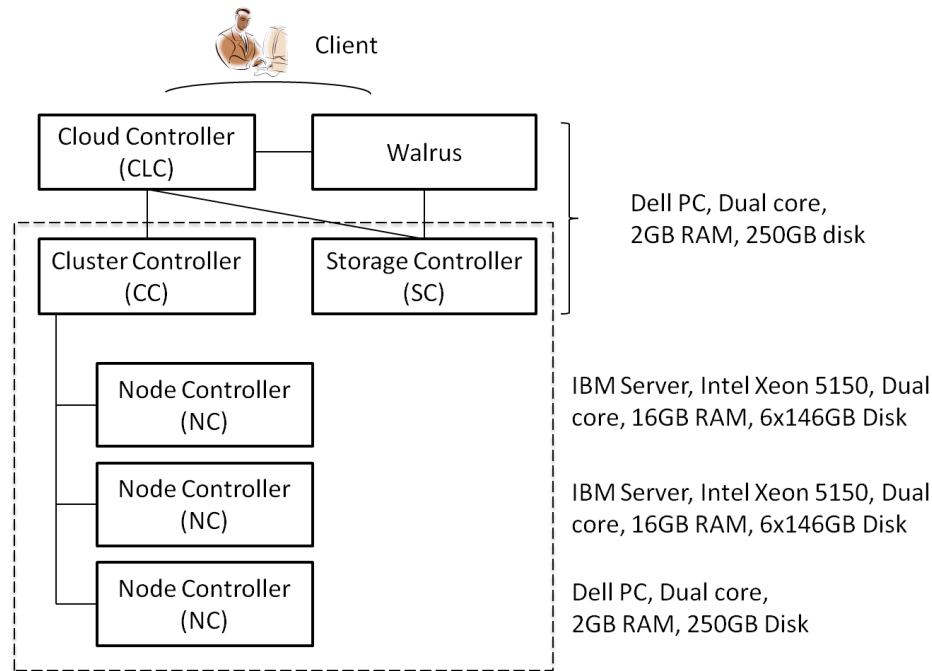


Figure 4 Computer nodes for the deployment of Eucalyptus

We totally used 5 machines:

- 1 machine for Cloud Controller, Cluster Controller, Storage Controller and Walrus;
- 1 machine for front-end client connection; and
- 3 machines for Node Controller.

More Node Controller machines can be added into the configuration if required. For better performance, it is recommended to deploy Cloud Controller, Cluster Controller, Storage Controller and Walrus on separate machines. In the current setting, we have in total 6 cores, 34GB RAM, 2002GB disk on all Node Controllers.

Table 4 lists the maximum virtual machines we can obtain for the current configuration. The *maximum no of instances to offer* column shows merely the static instance configuration, which is more than the actual instances that can be launched. For example, although there are maximum two *c1.xlarge* instances are available in the configuration, only one can be launched.

Instance type	No. of CPU	RAM (GB)	Storage (GB)	Maximum No. of instances to offer
m1.small	1	0.25	5	10

c1.medium	1	0.5	10	10
m1.large	2	0.5	50	5
m1.xlarge	2	4	100	2
c1.xlarge	4	6	100	2

Table 4 Maximum virtual machine instances for the Eucalyptus deployment.

2.4 CloudStack

2.4.1 Introduction

CloudStack architecture is based upon a hierarchical structure which provides IaaS cloud services for either private, public or hybrid configurations. It offers centralized management capabilities over all cloud components (network, storage and nodes) and it also provides on-demand access to infrastructure through a self-service portal. This technology was developed by Cloud.com and in May 2010 it was released as free software under the GPLv3 license. In 2011 the project was acquired by Citrix, changing the licensing from GPLv3 to Apache License version 2 [49].

2.4.2 Architecture

CloudStack is organized in the form of nested components. The primary organization structure of CloudStack is *Zone*. A *Zone*, is an organizational unit, which can be identified for example, as a single Datacenter. Each zone consists of *Pods*. *Pods*, can be a single rack of servers in a Datacenter. Hosts belonging to a *Pod* share the same subnet. *Pods* consist of *Clusters* and *Clusters* consist of *Hosts* and their affiliated primary storage. *Clusters*, are an organizational structure for grouping hosts together. It can be a pool of Xen, KVM or ESXi (VMware) hosts. Each *Cluster*, has its own primary storage.

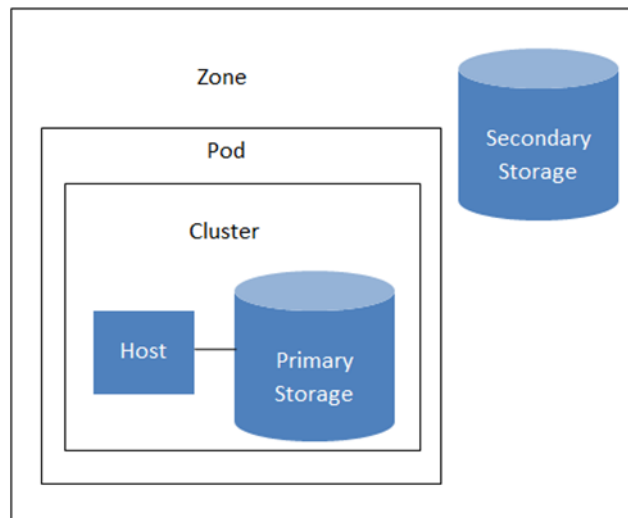


Figure 5 Cloudstack nested organization⁹

There is also another type of storage, *Secondary Storage*, which is shared among by all the components of a Zone. In brief, CloudStack has 9 components as shown in Table 5.

Component	Description
Hosts	Machines used for provisioning services
Cluster	A host group with their storage
Pod	A collection of clusters
Zone	A collection of pods.
Management Server Farm	Management nodes
Primary Storage	VM storage
Secondary Storage	Storage for support stuff, (templates, ISO images etc)
Network	A logical network structure associated with services

Table 5 Cloudstack components

2.4.3 Amazon EC2 and Amazon S3 support

CloudStack includes, as part of its API, AWS services integration. It is compatible with Amazon's EC2 and S3 services and in the current version, the related service is fully integrated in CloudStack.

CloudStack provides a translation service for AWS API calls, which are translated through this service into its own API calls. It supports both EC2 SOAP & Query API and S3 REST API [67].

⁹ Picture taken from: https://cloudstack.apache.org/docs/en-US/Apache_CloudStack/4.0.2/html-single/Admin_Guide/

However, in its current stable version, there are also certain limitations to be mentioned [68]:

- AWS is available only through zones with basic networking
- Elastic IP and Elastic Load Balancing are only available with Citrix NetScale device.

2.5 VMWare vSphere

VMware vSphere (formerly VMware Infrastructure 4) is VMware's server virtualization platform for build cloud infrastructure. VMware vSphere platform is formed by a series products and components as shown in Figure 6, which includes core component - VMware ESXi Server, management – vCenter Server and vSphere Client. The following is a brief introduction to the main component of VMware vSphere.

2.5.1 VMware hypervisor (ESXi)

VMware Hypervisor is a free bare-metal hypervisor, it install/run directly from physical server and does not require a separate operation system (OS) to be installed. It has less overhead compared to other hypervisors which require a hosting OS. VMware vSphere used to support both ESX and ESXi, however from version 5, it only support ESXi as its hypervisor [143].

2.5.2 VMware vCenter server

VMware vCenter server is the centralized platform for configuring and managing VMware vSphere environments. It provides core data centre services including access control, performance monitoring and warning managements.

2.5.3 VMware vSphere client/web client

VMware vSphere client is a windows application which connects to vCenter Server or ESXi user interface, and web client allows user to connect to vCenter server through browsers (with plugin installed).

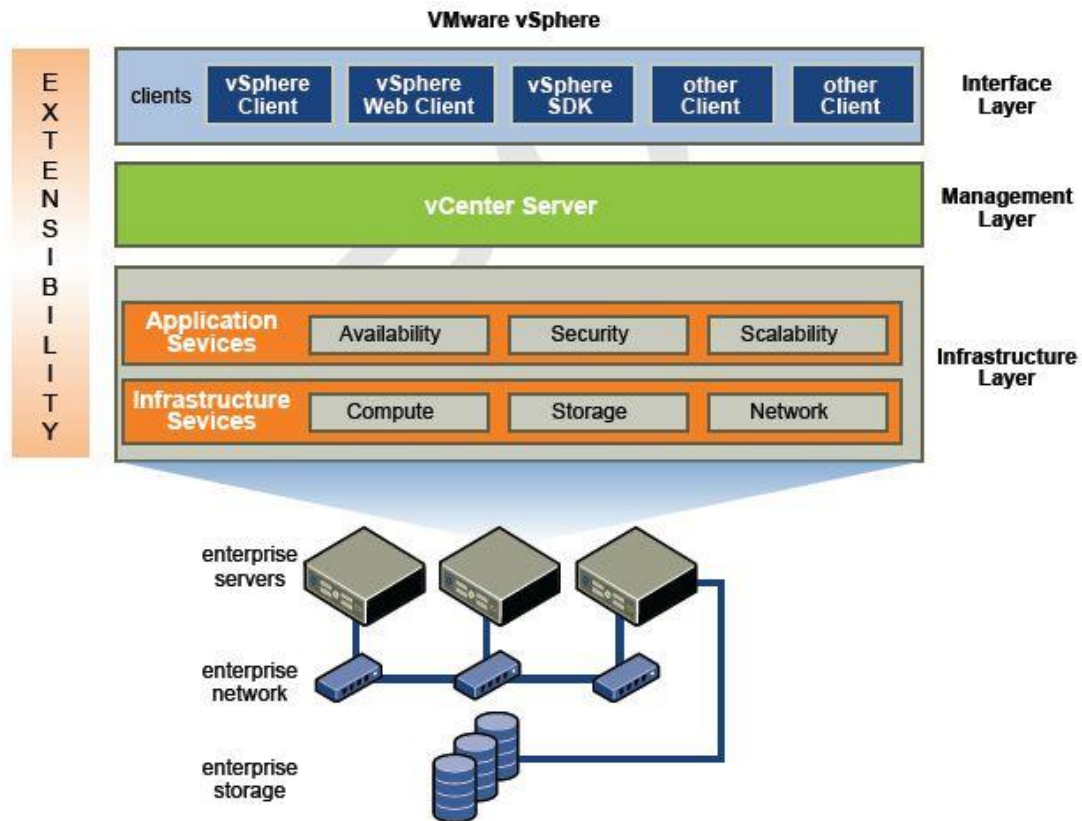


Figure 6 Overall structure of VMWare vSphere¹⁰

2.5.4 vSphere virtual machine file system (VMFS)

VMFS is a high-performance cluster file system for VMware hypervisor (ESXi), and it is optimized for Virtual machines. VMFS allow multiple vSphere hosts to access the same storage concurrently by leverage shared storage.

2.5.5 vSphere virtual symmetric multi-processing (SMP)

Virtual SMP is a utility which allows a single virtual machine to use multiple physical processors simultaneously.

2.5.6 vSphere vMotion/storage vMotion

VMware vMotion allows user to move an entire running virtual machine from one physical server to another one without downtime. It also allows automate and schedule migrations within the same datacentre. Storage vMotion allows live migration of virtual machine disk files within and across storage arrays, which is performed at zero-downtime and without service disruption.

2.5.7 vSphere High Availability (HA)

HA monitor vSphere hosts and virtual machines to detect hardware and OS failures, and restart virtual machine on other hosts in cluster automatically when a server outage is detected. HA reduces application downtime by automatic restart VM and delivers the availability.

¹⁰ Picture taken from https://pubs.vmware.com/vsphere-4-esx-vcenter/index.jsp?topic=/com.vmware.vsphere.intro.doc_41/c_vmware_infrastructure_introduction.html

2.5.8 vSphere distributed resource scheduler (DRS)/storage DRS

DRS groups vSphere hosts into resource clusters to segregate the computing demands of various business requirements. It provides HA resources for workloads, balance workloads for optimal performance and help scale computing resources.

2.5.9 vSphere fault tolerance (FT)

FT provides continuous availability for applications in the event of server failure. FT creates a live shadow instance of a VM which is always up-to-date with the primary VM, in the event of outage, FT automatically triggers failover – this ensures the shadow VM to become live and to prevent downtime or data loss.

2.5.10 vSphere distributed switch (VDS)

VDS provides a centralized interface for configuration, monitoring and administration of VMs across the whole data centre. It enables the same network identity when a VM is migrated between different hosts.

2.5.11 Amazon EC2 and Amazon S3 support

VMWare has two products, which are vCloud Automation Center [144] and vFabric Application Director [145] that can support deployment on Amazon. Automation Center allows policy-based provisioning across VMware-based private and public clouds, physical infrastructure, multiple hypervisors, and Amazon Web Services. Application Director enables applications to be deployed across multiple virtual and hybrid cloud infrastructures, including Amazon EC2.

2.5.12 Environment setup for feasibility test

During the cloud investigation, a small VMWare vSphere has been established in BED as described below. The VMs themselves can be configured to be clustered computing, Hadoop cluster of two VMs are configured and tested. The findings are very promising and the performance is good. However, due to the licence cost, this cloud is not used for the consortium within CHIC.

For feasibility research, two Dell T7400 servers (with two Xeon E5410, 1T storage space, and 16G memory each) are connected to gigabit network switch. A windows PC with vSphere client installed is used to configure and install VMs. The basic layout the validation platform is configured as shown in Figure 7.

The configurations of servers and networks are for validation of the VMware vSphere, so all the resources are dedicated to the test platform, which includes 32G full-buffered memory and 2T storage space and 16 cores of 2.33GHz CPU. For a normal VM, 4-6G memory and 1-2 cores are allocated, which gives us 6 powerful VMs to run tests.

Regarding security, Https are always used for the vCenter server web interface access, strong password policy are enforced to administrators and users who have access. Least rights are assigned to various parties, this is to control the range of damage might be done with stolen credentials. For VMs access, Linux shell access is given with only private/public key authentication allowed, while windows are allowed to access through Remote Desktop Connections.

One of the servers has vSphere vCenter Server installed, which allows the management of the servers from various platforms through browsers. For the validation purpose, we have carried out following activities on above platform.

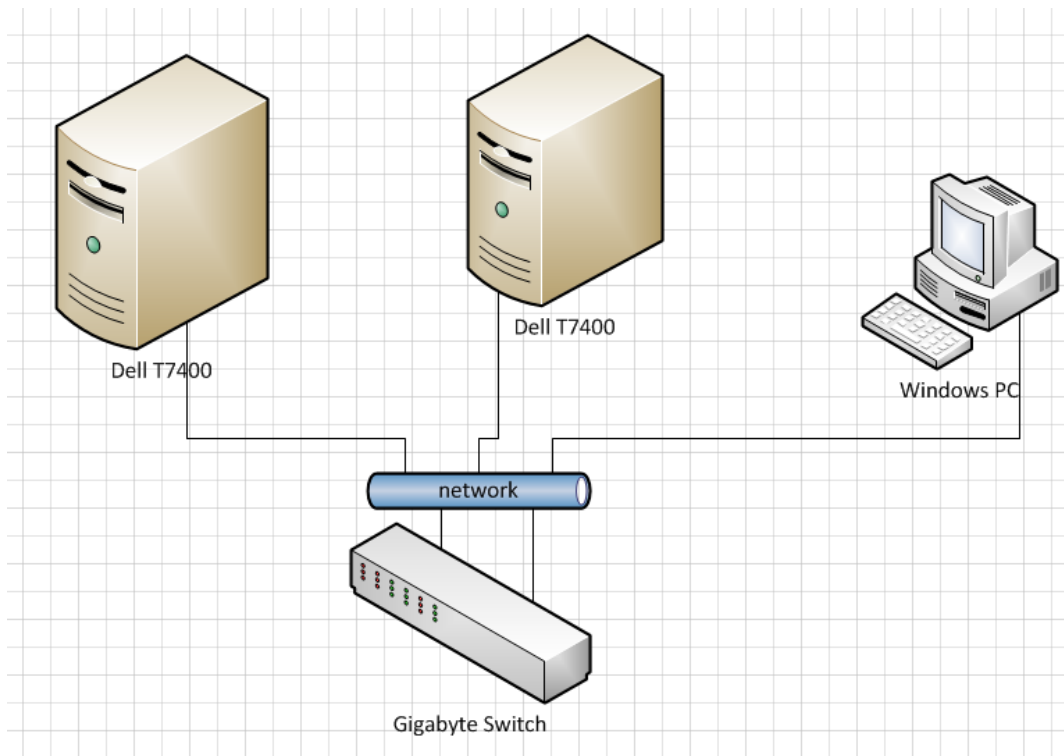


Figure 7 Overall structure of VMWare vSphere

2.5.12.1 Install vSphere hypervisor on both servers

Two 1T USB drives are used to install the EXSi on both servers, and the server then boot up from the USB drives. This allows all the hard drive capacities be used for VMs. USB access speed is not a concern here, since all content of USB is loaded into memory after initial boot process.

2.5.12.2 Connect to EXSi hosts and install vSphere vCenter server

Windows PC is utilized to install vSphere client which connects to EXSi hosts, configure the hosts and install vSphere vCenter Server (VM) on one of the server, which further allows the clustering and management of hosts through web interface.

2.5.12.3 Configure both server as cloud and install Linux/Windows VMs

Configuration is done to make two server cluster together as local cloud environment, the vCenter server allows configuration, management and monitoring in one web UI. Ubuntu Linux 12.04 LTS and Windows Server 2012 R2 are installed as a test of the VM hosting ability.

2.5.12.4 Configure VM templates and create VM from template

To have a standard template for consortium members to work on is very important, it avoid the configuration caused deployment issues. VMware vSphere platform support the template and create new VM based on template very well, the interface is intuitive and friendly.

2.5.12.5 Clone and migrate VM

Clone and migrate VM within same hosts and between two different hosts are tested, the processes are completed without any problem.

2.5.12.6 Monitoring and disaster recovery

The monitoring of VM through vCenter interface, and tested disaster situations includes one hosts get offline and power cut of both hosts. The recovery from disaster as behaves as documented by

VMware and the downtime depends on the situations (e.g. both hosts power cut will only go live when the power comes back again).

2.6 Comparison of cloud technologies

All four software platforms in review share the basic qualities of a cloud platform. Table 6 summarizes the basic characteristics of these platforms:

	OpenStack	Eucalyptus	CloudStack	vSphere
General				
Architecture	Fragmented into lots of “projects”	Five main components. AWS clone	Nine main components in nested structure	Lots of components of various functionalities
Installation	RPM/DEB packages	RPM/DEB packages	RPM/DEB packages	Standalone from USB drive, then vCenter Server VM
Administration	Web UI, native CLI	Strong CLI compatible with EC2 API	Customizable WEB UI and AWS compatible API	Windows Client, Web UI
Security	Baseline + Keystone, roles	Baseline + component registration	SSO, user roles	SSO, Security Token Service and internal LDAP for Users and Groups
High Availability	Swift Ring, multiple service endpoints, external load balancers	Primary/secondary component failover	Load balancing and HA options	High Availability and Fault Tolerant components
Hypervisors	Xen, KVM	Xen, KVM, VMware	Xen, KVM, VMware	VMware Hypervisor (ESXi)
Language	Python	Java, C	Java	Assembly, C, C++
OS Support	Various Linux distributions, Windows and BSD	Various Linux distributions, Windows and BSD systems	Various Linux distributions, Windows and BSD	OS X Server 10.6+, Various Linux, MS-DOS,

	systems		systems	Windows, BSD, OS/2 Warp 4, NetWare 6.x, Solaris 10
Storage				
Disk Images	Yes	Yes	Yes	Yes
Block devices	iSCSI, FC	iSCSI, FC	iSCSI, FC	iSCSI, FC
Object Storage	Swift API and Limited S3 API	S3 API	Limited S3 API	Yes
Fault tolerance	Yes	Yes	Yes	Yes
VM image facilities				
Image service	Yes	Yes	Yes	Yes
User VM images	Yes	Yes	Yes	Yes
Amazon API	Yes	Yes	Yes	No
User UI and Management Facilities				
Web Interface	Yes	Yes	Yes	Yes
Users & Quotas	Yes	Yes	Yes	Yes
Networking				
Floating IPs	Yes	Yes	Yes	Yes
L2 Support, VLANs	Yes	Yes	Yes	Yes
DHCP Support	Yes	Yes	Yes	Yes

Table 6 Comparative evaluation of OpenStack, Eucalyptus, Cloudstack and VSphere

All four platforms provide the basic functionality of a Cloud infrastructure and support all widely used and IT industry standards storage and network technologies. Moreover, these platforms can provide the three basic models of operation of a Cloud platform (IaaS, PaaS and SaaS).

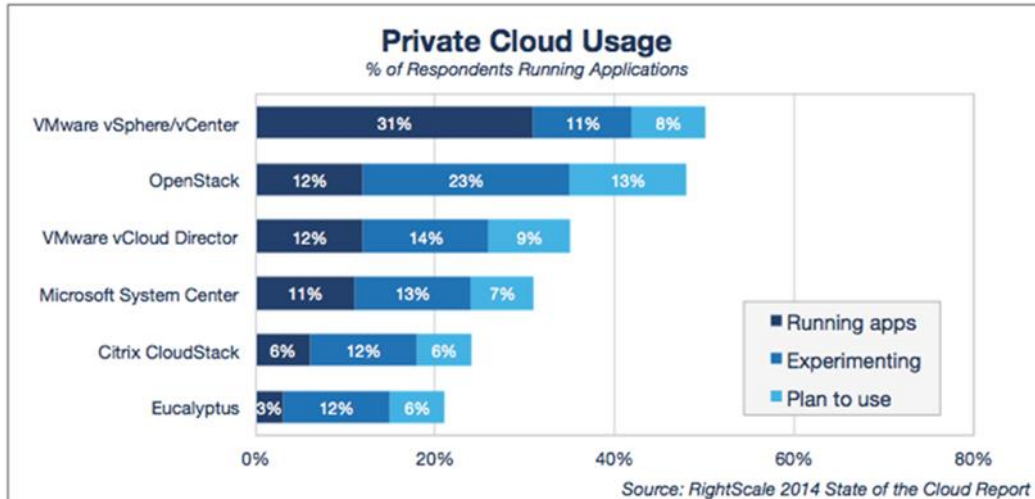


Figure 8 Cloud technology adoption and usage

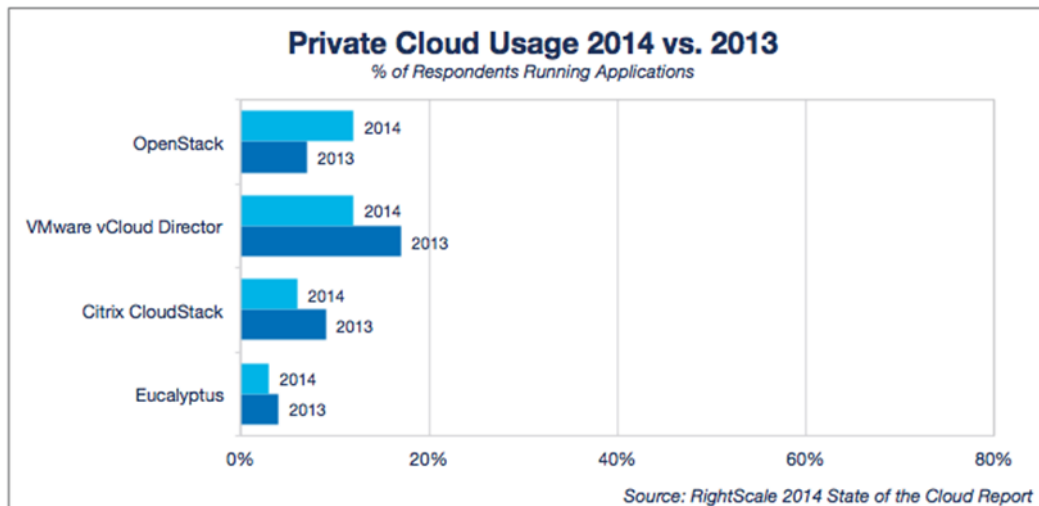


Figure 9 Private Cloud Usage trend (2014 vs. 2013)

In terms of the level of adoption of the corresponding cloud platforms, both VMware vSphere and OpenStack have good adoption and usage rate. In the case of private clouds, a recent (2014) survey by RightScale [72] indicated that OpenStack is catching up VMware vSphere to have much more preferable choice for private cloud deployment, as shown in Figure 8. A comparison of the 2014 results with the 2013 verifies the trend in private clouds (Figure 9). Although vSphere has good performance and adoption, we opt for OpenStack in our infrastructure implementation due to vSphere's non-open source license issue. Additionally, due to the fact that OpenStack, CloudStack and Eucalyptus are designed as Open Source technologies they can be extended and adjusted accordingly to the needs of each individual setup.

As a prerequisite, a cloud platform must provide at least the required functionality for implementing some or all of the three basic models of operation (IaaS, PaaS and SaaS). Especially in the context of the biomedical domain, due to the sensitive nature of the biomedical data, a cloud platform must provide enhanced security and comply with industry standards. Moreover, the size of biomedical data grows in time as research methods advance and thus, an important factor for choosing a cloud platform over another is the scalability in terms of storage. As biomedical data enter the big-data era,

it is desirable for a cloud platform to have integrated support and provide, as-a-Service, interoperability with Hadoop clusters [69].

For a private cloud infrastructure, it is also critical to provide an API for public clouds. Depending on the needs that may arise, a private cloud platform should be able to provision and manage virtual machines and storage in public clouds.

An additional requirement established, in the context of this work, was the need for the selected cloud platform to provide a flexible and easy to use API for developers in order to extend, customize and deploy custom applications on top of the cloud.

In relation to this requirement, OpenStack has some major advantages over the other two open source platforms:

- i. Apart from the traditional storage options, it provides an object storage (OpenStack Swift) functionality, with a strong and complete API which can be used to create applications directly attached to the object storage on top of the cloud. This means that, despite the underlying structure of the storage, which is entirely handled by the OpenStack, developers can create applications which can store and retrieve data and metadata directly from/to the OpenStack's object storage.
- ii. OpenStack Swift, with the introduction of Policies, can maintain data locality which can be a requirement especially when handling Biomedical data
- iii. OpenStack, has introduced *project Sahara*, which enables the provisioning and management of Hadoop clusters on OpenStack [70]
- iv. OpenStack provides an S3 Amazon API which enables the interoperability of an in-house, private cloud infrastructure with Amazon's public cloud
- v. The OpenStack project continues to evolve; since the ninth OpenStack release ("Icehouse"), it deploys new types of cloud service like desktop-as-a-service (DaaS), database-as-a-service and more.
- vi. Almost all major Linux distributions contain their own packages for OpenStack and many, such as Redhat, extend their own cloud platforms to support interoperability with OpenStack [71].

As a result, OpenStack has at least the same functionality as the other two open source platforms in question and furthermore, it offers a ready to use API for creating cloud storage aware applications and the functionality to create, extend and adopt a private cloud storage infrastructure.

3 OpenStack architecture

3.1 Introduction

OpenStack consists of a number of services, which can be categorized as follows:

- Messaging services
- Authentication/Authorization services
- Network Controller services
- Computing services
- Imaging services
- Storage services

In brief, OpenStack elements communicate with each other through the *Messaging Services*. *Network Controller Services* provide network infrastructure consisting of a combination of private and public networks available to virtual machines and the means to connect the internal infrastructure to the outside world. *Computing Services* handle virtual machines and their resources, while *Imaging* and *Storage Services* provide images and storage devices either to virtual machines or to the outside world.

Using OpenStack's own terminology, its conceptual architecture is presented at the following image:

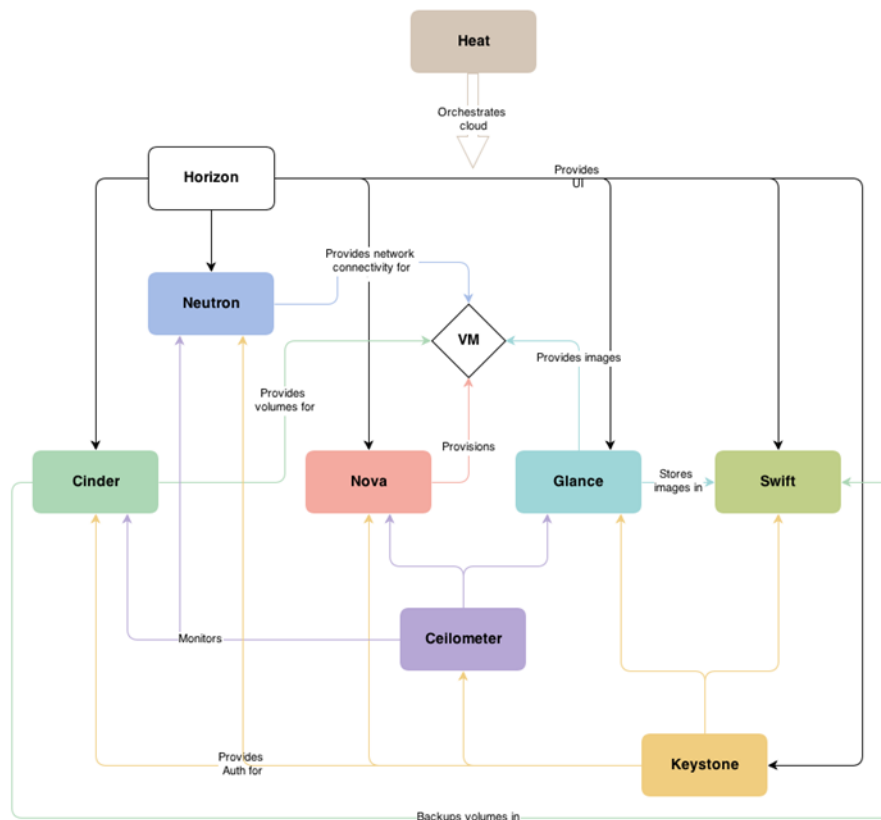


Figure 10 OpenStack architecture

All major architectural modules of OpenStack have codenames, as seen in the figure above:

- *Cinder*: is the block storage service of an OpenStack cloud
- *Nova*: is the computing service, e.g., the service responsible handling and provide resources to VMs

- *Glance*: is the imaging service, which handles operating system images
- *Swift*: is an OpenStack specific storage service which provides Object Storage infrastructure
- *Neutron*: represents the OpenStack networking portion, which consists of a set of services responsible for providing a NaaS (Network as a Service) infrastructure
- *Keystone*: is the service responsible for authenticating and authorizing services and users inside the OpenStack cloud
- *Horizon*: implements a graphical user interface for basic cloud operations

End users can interact with OpenStack only through *Horizon* or to each individual service through their APIs. End users, or *clients*, are assigned into *Tenants* (groups of computing resources) and *Roles* (structures which provide control over shared resources).

In order to interact with each OpenStack service API, one has first to authenticate through the *Identity Service*. After a successful authentication, the client receives a token which in turn authenticates the client against various OpenStack services based on its role and tenant assignment.

APIs can be accessed with the following methods:

- cURL
- OpenStack command line clients
- REST clients
- OpenStack Python SDK

The authentication and API request workflow, can be summarized as follows:

1. Authentication token request with the appropriate payload
2. Send API request including the X-Auth-Token header
3. Continue sending API requests
4. If a “401 error” arises, request new token

The payload for each request, can contain:

1. Username (type xsd:string)
2. Password (type xsd:string)
3. tenantName (type xsd:string)
4. tenantID (type: UUID)
5. token (type: UUID)

As an example, a typical cURL command for requesting a new token can be as follows:

```
curl -i 'http://<Server IP Address>:5000/v2.0/tokens' \
-X POST -H "Content-Type: application/json"
-H "Accept: application/json"
-d '{"auth": {"tenantName": "admin", "passwordCredentials": {"username":
"user1", "password": "usersecret"}}}'
```

If the request is successful, it will result to a HTTP/1.1 200 OK response and the token in the form of *"id": "token"* and its expiration information.

After acquiring a valid token, more API requests can be send using the X-Auth-Token header, for example:

```
curl -i -X GET http:// <Server IP Address>:35357/v2.0/tenants
-H "User-Agent: python-keystoneclient" -H "X-Auth-Token: token"
```

OpenStack provides SDKs and toolkits for a variety of technologies, like Python, Java, C++ and .NET. More information can be found at the official OpenStack Wiki pages.

3.2 Object Storage

The term “Traditional Storage” often refers to storage types that present themselves either as block devices or in the form of file storage. In fact, with this term we imply hard disk devices, disk arrays, directed attached storage devices or even network attached storage devices. In order to use these devices, we rely on a volume manager and a Filesystem, or just a Filesystem built on top of the block devices. A Filesystem is the actual point of reference for performing read and write operations on the actual raw hardware.

In this way the storage is made available either locally on a single machine, or available to several machines in the network via sophisticated and usually inflexible protocols, like NFS, AFS, CIFS etc. Of course, even if the actual storage is available locally on a machine a Web Service can provide new and more agile methods of read, write and sharing the data and thus, the available storage.

A common problem with this types of storage, is that data manipulation is limited by the Filesystem itself, for example, read and write operations are limited to block level, security is weak (with a limited set of access rights) or very complex using ACL's. Moreover, sometimes expanding the storage can be an issue and the performance improvement is questionable.

In the case of large datasets, of the order of several petabytes, both expandability and performance are crucial.

As a result, storage cloud providers and organizations handling large amount of data often seek alternatives. In the early 1990's a new type of storage was introduced, object storage. Since 2005 and forward the IT industry has implemented a variety of object based devices, filesystems and storages. Some include T10 compatible storage devices [73], filesystems like Lustre [74] and cloud storage, like Amazon Web Services (S3) [75] and HP Cloud [76].

By design, an object storage refers to a storage architecture where data, the relevant metadata and a unique identifier are packed together as an entity, called “object”. Unlike files, objects do not rely under a hierarchy, but instead, they occupy a flat address space. The entire collection of these objects form an “object store” or an “Object Storage”. Several mechanisms are deployed in an object storage, which is responsible for handling objects and performing operations like read, write, create, update and delete. The objects are stored into containers instead of trees (like a filesystem) and each operation on the objects are validated through credentials. Access to object storage is usually provided through Web Services via a RESTful API.

Interaction with data through the API is holistic, since one must read or write the entire object with a single operation. Although this may seem a drawback, it drastically reduces the overhead of random I/O for a single “file”, but at the same time it prohibits traditional POSIX compliant utilities to interact with the actual data.

Another significant advantage, is that of by-design redundancy and scalability architecture of an object storage. Almost all implementations create multiple replicas of the data as efficient as possible, are designed to scale up beyond the PB range and implement mechanisms which handle hardware failures.

Given the unique characteristics of an object storage, it makes it ideal for handling unstructured data, like media files and web content but essentially unfit for relational databases or data that demand random access within the objects.

One important feature of object storage, is that it can be deployed in a mixed hardware environment and make use of even commodity hardware which is a huge advantage over traditional hardware especially when cost is an issue.

3.3 OpenStack Object Store (Swift)

The OpenStack cloud platform is actually a collection of individual projects. One of them is Swift, which implements a cloud storage, distributed in a number of storage nodes, in the form of an object storage accessible through a REST API [77] [78]. API calls are handled by a set of specialized servers which pass the calls to the actual storage devices. The storage devices are organized into clusters and data are evenly distributed between them.

OpenStack Swift, being a distributed storage system by design, delivers High Availability (HA), and scalability. In order to achieve that, it uses a data structure called the “Ring” which is a modified version of a consistent hash ring, in conjunction with multiple proxy servers, accounts and the concepts of *zones* and *partitions*.

Due to its API, Swift can be intergraded directly into applications and features and its design has some profound benefits:

- Can be deployed with commodity hardware
- Does not use a central database which could cause a bottleneck
- Built-in replication
- Supports the S3 API
- Direct object access
- Virtually unlimited storage

3.3.1 Authentication

Authentication is provided by the WSGI middleware or by external systems. Each client request contains an authentication token which is validated by Swift. The authentication tokens are cached and are valid for subsequent requests until they expire.

The token is a simple string and can be for example, a UUID or an MD5 hash. The token, is passed into Swift using the *X-Auth-Token* or the *X-Storage-Token* header. Tokens are acquired by the clients by presenting a set of valid credentials to an authentication service. After successful authentication, the authentication service responds with a valid token and the account URL.

The authentication provider, apart from the user accounts, also defines roles and user types and potentially can make use of API key tokens or x.509 SSL certificates instead of traditional credentials [79] .

3.3.2 Object Storage data hierarchy

Objects in Swift are organized into a hierarchy, which is defined by the *Account*, *Container* and the *Object*:

- *Account*: the account is created by the service provider and defines a namespace for the *Containers*.
- *Containers*: containers, define a namespace and control access to *Objects* with Access Control Lists (ACL).
- *Objects*: objects contain the actual data and possibly a set of custom metadata.

Interaction with Swift, uses a hierarchy defined by the Account, the Container and the Object. Each object is accessible with a unique URL of the form:

`http://servername/v1/{account}/{container}/{object}`

The URL contains the *service endpoint* (the address where the request is to be sent), and the resource path which contains the location of the object. The cluster location is defined as “servername” and the resource path as “v1/account/container/object”.

In brief, the account, container and object relationship is illustrated at Figure 11:

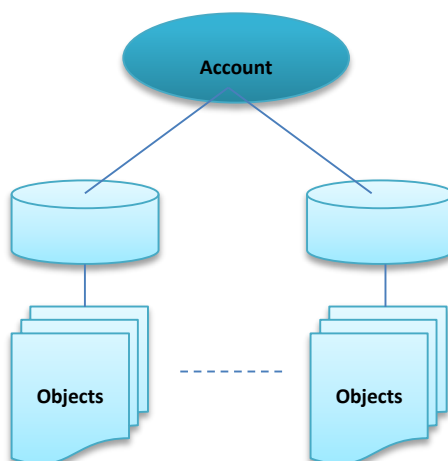


Figure 11: Account-Container-Object hierarchy

3.4 Swift: model of operation

In order to achieve this model of operation, the OpenStack is built on top of three basic components, servers, processes and rings:

3.4.1 Servers

There are four different servers:

- Proxy servers: the handle the incoming API requests to perform several operations on objects.
- Account servers: are responsible of managing the accounts related to the objects.
- Container servers: they handle groups of objects which are organized into containers.
- Object servers: the servers operating directly on the objects and handle the interoperability with the underlying filesystems.

3.4.2 Processes

Several process are involved into data replication, data consistency operations handling hardware failures and so on.

3.4.3 Rings

Rings, are defined as mapping between the physical location of objects and objects’ logical names. Rings in Swift, are internal structures which keep track of the location of objects, including the storage node and disks where the data resign.

All three components result into a cluster of machines, often called Swift Cluster, which operates as illustrated in Figure 12:

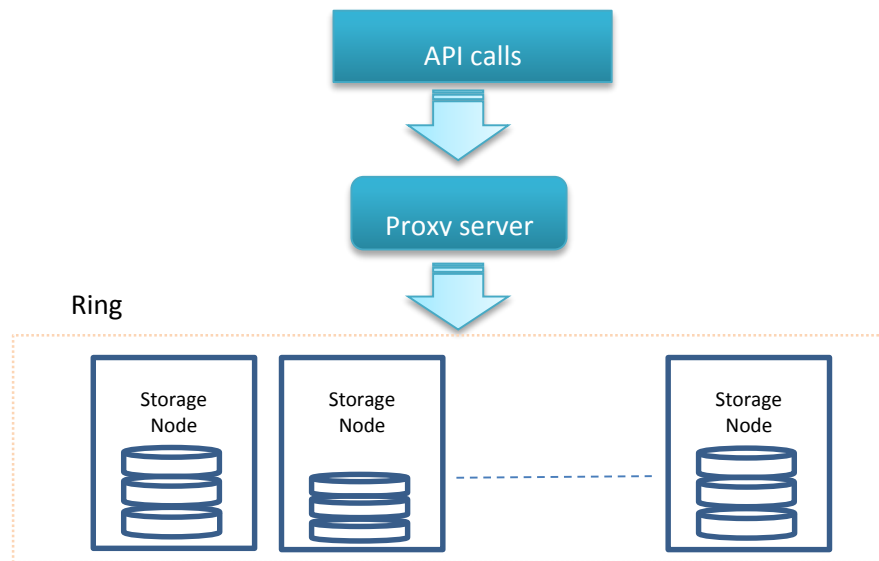


Figure 12 Swift cluster operation

The proxy server receives the API calls from the clients and directs the calls to the appropriate *Object Server* which operates on the *Storage Node*. Given the structure of the object URL, the proxy server is capable of determining the object location from the container portion of the URL. Containers in Swift are tightly coupled with Rings, which are the actual holders of the information concerning the location of each object.

Data integrity and High Availability is mainly achieved with data replication into multiple storage servers inside a Ring. For better efficiency, Swift uses *Zones* which provide data isolation, in the sense that, replication takes place across zones and thus, in case of hardware failure in a zone, other replicas in other zones can serve the correct data back to the clients.

The replication system of Swift is based on the concept of *Partitions*. A Partition, is a collection of data like container and account databases and objects and they are implemented with traditional directories with the corresponded hash table of what it contains.

3.5 Discussion

OpenStack is a collection of several projects which constitute a fully featured cloud platform. Although most of these projects feature independent characteristics, they promote, through an efficient API, interoperability and present their functionality as part of a whole.

OpenStack, uses a messaging system for service connectivity and provides a common identity management service for authentication and authorization purposes. Each service has its own API through which it inter-operates with other services, featuring a RESTful interface, also available to end users outside the cloud infrastructure.

Apart from the traditional storage options OpenStack comes with its own object storage, solution, Swift, which has its own RESTful API capable of providing the means to developers to implement cloud storage applications on top of the cloud infrastructure. Furthermore, the Swift structure, promotes scalability, facilitates load balancing and redundancy operations and thus permits the implementation of an enterprise scale cloud storage.

4 Implementation of a private cloud infrastructure

4.1 Introduction

Our implementation of a private cloud infrastructure involved the usage of several machines. Due to security and regulatory requirements, the IP addresses, the domain names and other technical details of the real machines involved in this setup have been masqueraded with randomly selected private Class B networks and hostnames.

The implementation of a cloud infrastructure requires a large number of physical machines in order to deploy the full extent of its features. The implementation of the CHIC private cloud was based on a development setup consisting of a limited amount of hardware resources, compared to a cloud of commercial scale, and due to this fact an incremental process of infrastructure deployment was chosen in order to gradually reach a stable and robust installation. After elaborate testing, the initial development infrastructure was promoted to productive usage and is gradually expanded with additional hardware resources based on the needs of the project and the requirements of the cloud users.

In order to deploy OpenStack version 9 (codename: Icehouse) we used several server machines, mainly of the following type:

- Dell PowerEdge SC1425, dual CPU Xeon 3.00
- Dell PowerEdge 720XD, dual CPU E5-2690
- A number of PCs with commodity hardware implementing a portion of Swift Object Storage

All machines involved in the storage portion of the cloud were equipped with RAID controllers and enough number of physical disks which can deliver hundreds of thousands of IOPS and a bandwidth of at least 500 MB/sec in order to eliminate possible bottlenecks.

For the network configuration on the other hand, due to hardware limitations we used 1Gbps network cards without bonding which would multiply the available bandwidth for read and write operations.

4.2 Software

We implemented a multi-machine/multi-controller model of operation, without redundancy due to the lack of additional nodes. Our setup is defined as follows:

- One node providing Keystone, Glance and Messaging services
- One dedicated Network Controller node providing the Neutron services
- One node responsible for providing block and object storage services (Cinder and Swift) along with the Compute Services (Nova)

An in-depth look at our implementation can be as shown in Figure 14:

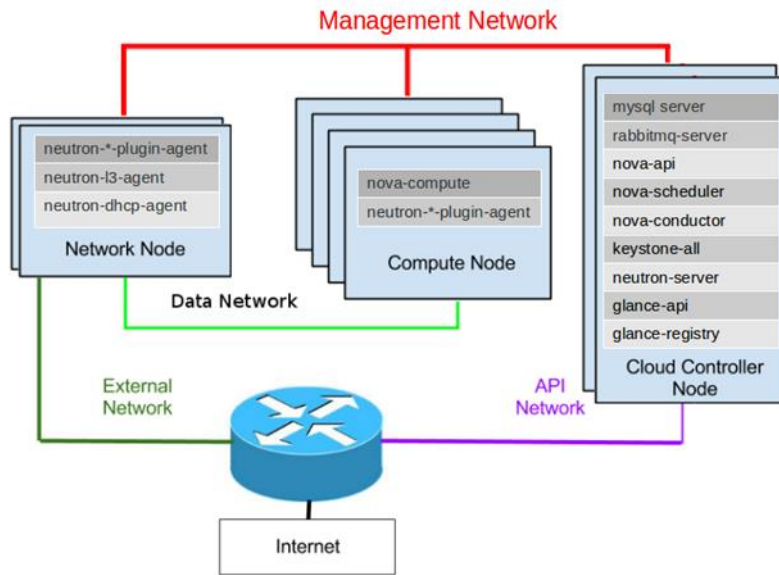


Figure 14 CHIC OpenStack deployment architecture¹¹

As one can notice, at least 3 internal networks are needed in addition to a public/external network. For the current implementation, we used 3 separate VLANs, 2 with private address space and 1 with public since the scale of the implementation permits us of a more modest configuration. Another significant point in this setup, is that all VM traffic is routed through the Network Controller node which acts a router machine and also provides a full-scale firewall for the virtual machines. In fact, OpenStack defines a set of firewall rules based on Linux IPtables and Open vSwitch flows [85]. These rules, are used on conjunction with Neutron security groups, which provide a complex but efficient way to establish security levels and access restrictions either to virtual machines or groups of resources [86].

By default, OpenStack uses Open vSwitch for implementing the network infrastructure. OpenStack networking built upon Open vSwitch provides two options for creating virtual networks:

1. Generic Routing Encapsulation (GRE) and
2. Virtual LANs (VLANs)

The implementation of our choice for creating virtual networks with Open vSwitch was VLAN, due to its plain and clean configuration it requires. GRE tunnels on the other hand, although are commonly used especially in many VPNs, usually need special tuning since the tunnelling function itself has an additional overhead over the actual data.

4.3 Network

We deployed OpenStack on three machines running Ubuntu 12.04.4 LTS with the latest official updates. For the network setup, we used three VLANs, tagged as 510, 511 and 512. VLAN 510 is a public VLAN and VLANs 511 and 512 are used internally by OpenStack. These VLANs were attached to the nodes with VLAN interfaces as follows:

Each machine has a VLAN interface, namely eth0.510 with no IP address. This interface is bridged to interface namely br-ex, which carries the public IP address of the node and used as a “gateway” interface by OpenStack in order to route all VM traffic from the internal networks to the outside world. Two additional VLAN interfaces are enabled on each node for VLAN 511 and 512 (eth0.511

¹¹ Picture taken from: http://docs.OpenStack.org/icehouse/install-guide/install/apt/content/ch_overview.html

and eth1.512), with network addresses 10.1.1.0/24 and 10.1.2.0/24 respectively. Network 10.1.1.0/24 is used as both the management and data network and 10.1.2.0/24 handles only virtual machine traffic.

The instances of the virtual machines running in OpenStack, have IP addresses allocated in the 10.1.1.0/24 network. Each virtual machine owns one or more virtual network interfaces which are bridged to the default bridge interface of OpenStack, namely br-int. This particular interface is enabled on each compute and network controller node (just one in our case). In this way, the traffic from every virtual machine is carried out through this network interfaces (br-int) to the network controller node(s) which in turn routes the traffic to br-ex and to the outside world.

In the present study, just one internal network was created for the virtual machines, although more should have been created in real world scenarios, so that, traffic from separate projects would be isolated in different logical networks, providing a secure environment for the virtual machines.

A network diagram of the current implementation is presented at Figure 15:

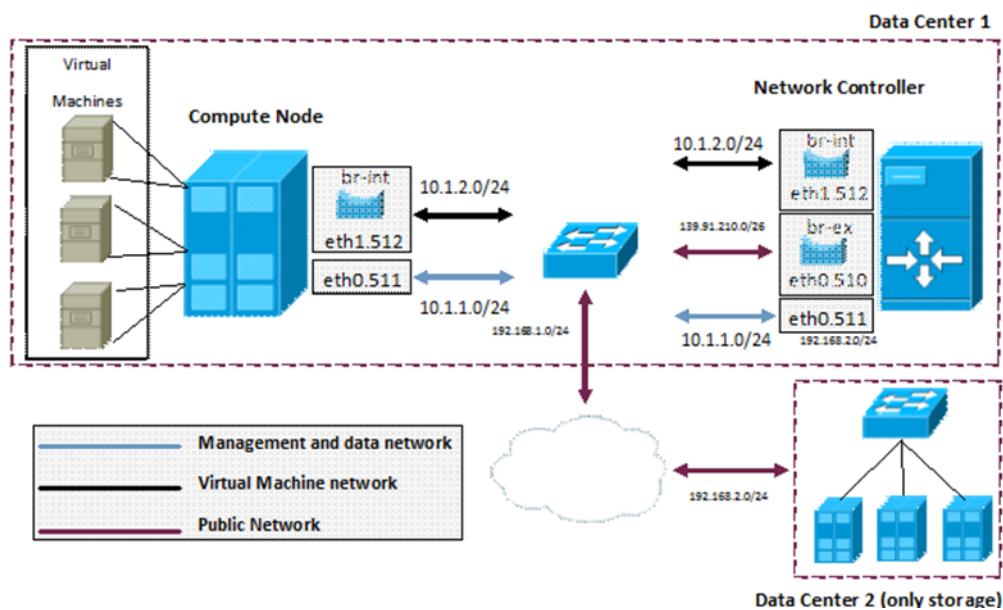


Figure 15 CHIC cloud cluster network diagram

The network topology, as visualized by OpenStack itself, is presented in Figure 16.

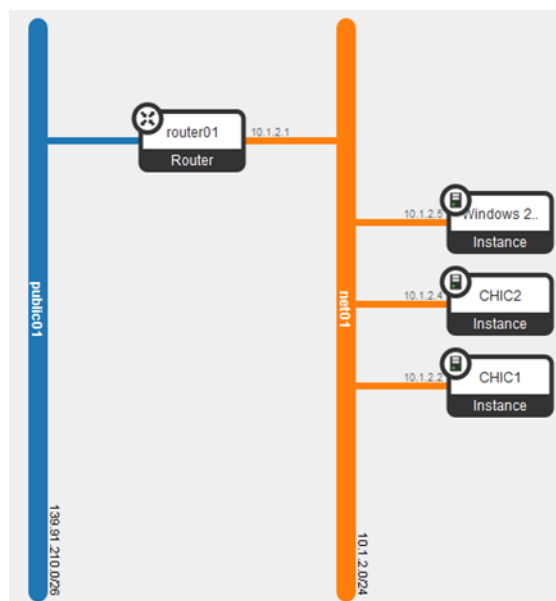


Figure 16 CHIC cloud network topology

Network *net01* is the internal network of the virtual machines. On this network are allocated the private IP addresses of the virtual machine. Multiple internal networks can be defined in the future, providing separate address spaces to each project. Each instance (virtual machine) gets its own IP address automatically from the DHCP service which runs on the network controller node.

The *router01*, is a router device which connects all virtual networks together and routes the traffic to or from the external network. This is actually a virtual device instantiated by the Open vSwitch on the network controller node.

Multiple virtual routers and networks can be defined for every distinct project or group of instances running on OpenStack compute nodes.

Since all VM traffic is routed through the network controller node, certain rules for inbound and outbound traffic can be applied. For this purpose, OpenStack has an embedded mechanism for defining and applying firewall rules which are defined as “*Security Groups*”. For this implementation we have slightly altered the default security group allowing traffic to virtual machine ports 80 (HTTP), 443(HTTPS) and 3389 (RDP) for security reasons.

4.4 Rings

For our initial deployment of the cloud storage and in order to implement different policies for potentially different types of data, we created four object rings and for two different sites (SiteA and SiteB), emulating different geographic locations for testing different security configurations:

1. A default ring which holds all the available storage devices at all locations:

```
/etc/Swift/object.ring.gz, build version 20
```

```
1024 partitions, 3.000000 replicas, 1 regions, 8 zones, 8 devices, 0.00 balance
```

```
The minimum number of hours before a partition can be reassigned is 1
```

id	region	zone	ip	port	replication ip	replication port	name	weight	partitions	balance	meta
----	--------	------	----	------	----------------	------------------	------	--------	------------	---------	------

0	1	1	10.1.1.10	6010	10.1.1.10	6010	sdb1	1.00	384	0.00
1	1	2	10.1.1.20	6010	10.1.1.20	6010	sdb1	1.00	384	0.00
2	1	3	10.1.1.30	6010	10.1.1.30	6010	sdb1	1.00	384	0.00
3	1	4	10.1.1.40	6010	10.1.1.40	6010	sdb1	1.00	384	0.00
4	1	5	10.1.2.10	6010	10.1.2.10	6010	sdb1	1.00	384	0.00
5	1	6	10.1.2.30	6010	10.1.2.20	6010	sdb1	1.00	384	0.00
6	1	7	10.1.2.30	6010	10.1.2.30	6010	sdb1	1.00	384	0.00
7	1	8	10.1.2.40	6010	10.1.2.40	6010	sdb1	1.00	384	0.00

This ring has 3 replicas 1 region and 8 zones on 8 different devices. We kept the number of zones equal to the number of devices, in order to achieve the maximum diversity in data placement.

2. One ring for SiteA:

```
/etc/Swift/object-9.ring.gz, build version 20
```

```
1024 partitions, 3.000000 replicas, 1 regions, 4 zones, 4 devices, 0.00 balance
```

```
The minimum number of hours before a partition can be reassigned is 1
```

id	region	zone	ip	port	replication ip	replication port	name	weight	partitions	balance	meta
1	1	1	10.1.2.10	6010	10.1.2.10	6010	sdb1	1.00	384	0.00	
2	1	2	10.1.2.30	6010	10.1.2.20	6010	sdb1	1.00	384	0.00	
3	1	3	10.1.2.30	6010	10.1.2.30	6010	sdb1	1.00	384	0.00	
4	1	4	10.1.2.40	6010	10.1.2.40	6010	sdb1	1.00	384	0.00	

This ring has 3 replicas 1 region and 4 zones on 4 different devices and represents a storage location for crucial data where more replicas are needed.

3. One ring for SiteB:

```
/etc/Swift/object-1.ring.gz, build version 5
```

```
1024 partitions, 2.000000 replicas, 1 regions, 4 zones, 4 devices, 0.00 balance
```

```
The minimum number of hours before a partition can be reassigned is 1
```

id	region	zone	ip	port	replication ip	replication port	name	weight	partitions	balance	meta
0	1	1	10.1.1.10	6010	10.1.1.10	6010	sdb1	1.00	384	0.00	
1	1	2	10.1.1.20	6010	10.1.1.20	6010	sdb1	1.00	384	0.00	
2	1	3	10.1.1.30	6010	10.1.1.30	6010	sdb1	1.00	384	0.00	
3	1	4	10.1.1.40	6010	10.1.1.40	6010	sdb1	1.00	384	0.00	

This ring has 2 replicas 1 region and 4 zones on 4 different devices. It represents a storage location for non-crucial data that can be reproduced easily and fast and do not require three replicas.

4. A ring which holds all the available storage devices divided into 2 regions and 4 zones:

```
/etc/Swift/object-10.ring.gz, build version 4
```

```
1024 partitions, 3.000000 replicas, 2 regions, 4 zones, 8 devices, 0.00 balance
```

```
The minimum number of hours before a partition can be reassigned is 1
```

id	region	zone	ip	port	replication ip	replication port	name	weight	partitions	balance	meta
----	--------	------	----	------	----------------	------------------	------	--------	------------	---------	------

0	1	1	10.1.1.10	6010	10.1.1.10	6010	sdb1	1.00	384	0.00
1	1	2	10.1.1.20	6010	10.1.1.20	6010	sdb1	1.00	384	0.00
2	1	3	10.1.1.30	6010	10.1.1.30	6010	sdb1	1.00	384	0.00
3	1	4	10.1.1.40	6010	10.1.1.40	6010	sdb1	1.00	384	0.00
4	2	1	10.1.2.10	6010	10.1.2.10	6010	sdb1	1.00	384	0.00
5	2	2	10.1.2.30	6010	10.1.2.20	6010	sdb1	1.00	384	0.00
6	2	3	10.1.2.30	6010	10.1.2.30	6010	sdb1	1.00	384	0.00
7	2	4	10.1.2.40	6010	10.1.2.40	6010	sdb1	1.00	384	0.00

This ring has 3 replicas 2 region and 4 zones on 8 different devices, which is essentially a ring split in 2 regions to emulate storage nodes in different data centres.

We also created some more rings just for benchmarking purposes, with the following characteristics:

- 3 replicas, 2 regions, 2 zones, 8 devices
- 3 replicas, 1 regions, 1 zones, 8 devices
- 3 replicas, 1 regions, 1 zones, 4 devices
- 3 replicas, 1 regions, 2 zones, 4 devices
- 2 replicas, 2 regions, 2 zones, 8 devices
- 2 replicas, 1 regions, 1 zones, 8 devices
- 2 replicas, 1 regions, 1 zones, 4 devices
- 2 replicas, 1 regions, 2 zones, 4 devices

The above rings represent different setups where the number of devices, zones and replicas varies. These setups, could help evaluate the effect of different configurations in terms of performance and failure scenarios.

Although we did not have the capability of using extra hardware components, like multiple Ethernet switches and different proxy servers for the object storage, it is recommended that when implementing such setups, non-blocking communication between the storage nodes should be enabled in order to eliminate possible performance degradation due to data transfer or sync operations.

4.5 Discussion

For this deployment, several nodes were used in order to implement different policies on Swift rings and emulate real world scenarios where data could often spread into different data centres, usually geographically isolated.

In this setup we took advantage of the network isolation that OpenStack provides, to create different networks for different projects and thus, isolate traffic between the virtual machines. Due to the limited hardware resources available, we only used three individual machines for the three major service roles, controller, network and storage. For the cloud portion of our setup, we deployed a SAIO installation, in which one machine emulated four storage nodes and we also used three more nodes in a different data centre in order to examine the behaviour of the cloud when using remote locations as part of the infrastructure. Finally, we tried to deploy high performance computing characteristics to our cloud, using several virtual machines as compute nodes and a dedicated virtual machine as management node. Unfortunately, the virtualization overhead resulted in degraded performance and the lack of additional hardware resources, which for example, may be used directly from the guest operating system taking advantage of the PCIe pass through livrtid option [95], forced us away from this direction.

However, despite all the drawbacks, we ended up with a fully functional cloud platform which provided IaaS and NaaS service models along with an independent cloud storage service.

5 Security

5.1 Introduction

It is outside the purpose of this document to provide a full and comprehensive OpenStack security analysis. In this section we provide an analysis of a few basic aspects of security:

- Identity management
- Authorization and Access control
- Data encryption

In addition we refer to how the cloud security mechanisms could be integrated with the CHIC platform security framework.

5.2 Identity Management

The identity service in OpenStack performs user management and provides a list of the available services and their API endpoints.

Internally, OpenStack uses the concepts of users, tenants, domains, regions, endpoints, services and roles. With these concepts, OpenStack virtually isolates system resources, group together users and resources and controls access to services.

Apart from users and services which are essentially self-explanatory concepts, we need to provide a brief introduction for the rest:

- Tenants: is a configurable virtual container which isolates resources and identity objects. It usually represent users and their relevant resources
- Endpoint: the network address of an OpenStack service in URL format
- Domain: can represent an organization, a company or an individual and provide administrative boundaries to the *Keystone* service
- Region: represents physically or virtually isolated resources inside a domain
- Role: a set of privileges which apply to users

Most commonly, a mapping is performed between tenants, users, roles and permissions. Each available resource must be first allocated to tenants. Tenant users can access this resource if they hold the corresponding role and have the appropriate permissions.

OpenStack uses a pluggable authentication system, in the sense that it delegates control of identification to external providers, most commonly the *OpenStack Keystone*.

Keystone performs user and their relevant permissions tracking, provides a catalog of available services and API endpoints and issues tokens for the API calls.

In Figure 17, we visualize the *Keystone* operation which in brief, is a series of identification retrieval based on valid credentials, a token issue and authorization of OpenStack services after a successful authentication check.

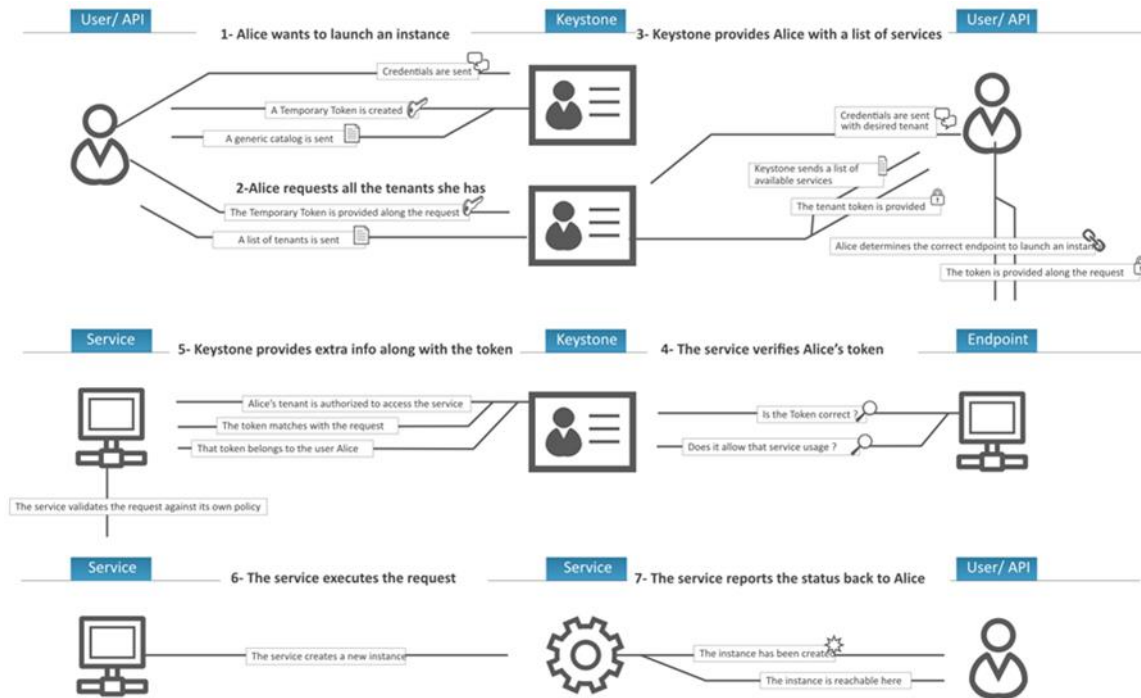


Figure 17 Keystone identity Manager¹²

Keystone provides integration with several back ends, like Pluggable Authentication Module (PAM) and Lightweight Directory Access Protocol (LDAP). Although the default storage option for identification information is SQL databases, like SQLite3 or MySQL, the architecture of keystone enables proxying external identification systems like OAuth [101], SAML [99] or OpenID [100].

5.3 Authorization and Access Control

The design of the authorization and access control model of OpenStack is heavily based on tokens. Resources can be successfully accessed with a valid token. OpenStack deploys a number of mechanisms to handle tokens, like an expiration system, revocation procedures and PKI store.

Internally, OpenStack uses the concepts of users, tenants, domains, regions, endpoints, services and roles. With these concepts, OpenStack virtually isolates system resources, groups users and resources together and controls access to services.

In principle, any Web service exposed to the Internet may be a subject of malicious attacks and possibly expose weaknesses, which may prove fatal to the underlying infrastructure. Several mechanisms have been deployed in OpenStack in order to minimize the threats, like sanitization of input in API calls, secure communications of the service endpoints over SSL connections, a PKI infrastructure and more.

To enhance security, the Identity service in OpenStack operates along with a kind of policy enforcement, since services in OpenStack acquire policy rules associated with the resources this service provides.

Most OpenStack projects implement *Role Based Access Control (RBAC)* and *Role Based Security* [102], which was standardized by ANSI (ANSI INCITS 359-2004). As an example the Identity API v3 of OpenStack, uses a JSON encoded policy file which contains declarations of the form [103]:

¹² Picture taken from: <http://docs.OpenStack.org/admin-guide-cloud/content/keystone-concepts.html>

API_NAME: RULE_STATEMENT or MATCH_STATEMENT

Where:

- **RULE_STATEMENT:** a *RULE_STATEMENT* or a *MATCH_STATEMENT*
- **MATCH_STATEMENT:** is a set of identifiers that must match between the token provided by the caller of the API and the parameters or target entities of the API call in question. A *MATCH_STATEMENT* is of the form:

ATTRIB_FROM_TOKEN: CONSTANT or ATTRIB_RELATED_TO_API_CALL

In order for OpenStack to comply with security standards, contributors from IBM have provided auditing functionality to OpenStack with the MTF Cloud Auditing Data Federation (CADF) standard [104].

5.4 Data Encryption

Data are not encrypted when stored in Swift. Although data encryption is a default behaviour in Swift, it is part of its functionality, for both storing and transferring data over the internet. This concept might prevent unauthorized access to the actual data, so that, even if cloud's security is compromised, a potential attacker will not be able to investigate, read or process user files. Even if a malicious action is not the case, storing unencrypted data on storage devices are at the disposal of the administrator of the storage. This can break legal and trust relationships and compromise the service.

In the case of biomedical data, data encryption can increase the level of data protection and prevent unauthorized access to sensitive information. Although server-side encryption seems a possibility, in cases where datasets of several GBs are involved, it may not be an optimal solution.

Data encryption can be applied to several levels:

- Swift object storage entities
- Data over the network
- Volumes residing on block storage

Encrypted network communication for both API calls and data transfer can be initiated with IPsec and tunnelling techniques [107]. Volume encryption can be achieved by selecting the appropriate back end solution, like Logical Volume Manager (LVM) [108] [109] encrypted volumes [110].

Object storage does not implement server side encryption in its upstream version, but modified and fully operational, open source Swift branches exist, which provide this functionality [111][112].

5.5 Integration with CHIC security framework

In section 5.2 we referred to the identity management mechanisms that are supported by OpenStack, such as SAML, OpenID and OAuth. In the CHIC security framework we also use SAML for the identity management, via a dedicated Identity Management service (CHIC SAML IDP), so an integration of those two mechanisms could be easily achieved based on this standard to allow a single identity management service for all the users. This integration has not been implemented in the initial phase of the cloud infrastructure deployment as we have to investigate the maturity and robustness of such an integration without compromising neither the functionality nor the security of both services. In addition, we lack of any real usage scenario that would drive such an integration in the basis of user requirements, because all cloud resources are allocated via the usage of Virtual

Machines (VMs) in the current setup, so there isn't any actual necessity to integrate the access to CHIC services with the access to the underlying, virtual or physical, computing resources.

Another field of possible integration between the security framework of CHIC and the cloud infrastructure is the auditing services, which could be extended to include the information of accessing the cloud resources. The audit logs of the cloud infrastructure are currently held and maintained separately of the CHIC auditing services, so an integration of them would allow to easily filter and examine access logs and reveal audit trails, either for performance enhancement or adherence to legal requirements. This task will be investigated during the course of the CHIC project for the feasibility and actual end-user functionality.

6 OpenStack quick installation guide

The following text is based on the official OpenStack Ubuntu 12.04.4 LTS / 14.04 LTS installation guide, adapted to reflect our own deployment configuration. For extensive and up-to-date documentation it is recommended that the users or administrators refer to the official installation manual, as this document only intends to provide a quick and brief guide.

This guide assumes that we have at least three machines available and that we have configured their network interfaces as follows:

- Controller node: VLAN interfaces eth0.510 (192.168.1.10), eth0.511 (10.1.1.10), eth1.512 (10.1.2.10)
- Network Controller node: VLAN interfaces eth0.510 (no IP), eth0.511 (10.1.1.11), eth1.512 (10.1.2.11) – bridge interfaces, br-ex (192.168.1.11), br-int (no IP). Interfaces eth0.510 and br-ex should be bridged.
- Compute node: VLAN interfaces eth0.510 (192.168.1.12), eth0.511 (10.1.1.12) and eth1.512 (10.1.2.12) – bridge br-int (no IP).

It is also crucial that clocks are synced on all nodes so an NTP service must be enabled and configured accordingly.

We are going to use a separate account for each service OpenStack runs, and each service requires a database backend. We will install MySQL on the controller node, create a database for each service and the corresponding database users:

Database name	Database user	Related service
cinder	cinder	Block storage
dash	dash	Horizon
glance	Glance	Imaging service
keystone	Keystone	Identity service
neutron	Neutron	Networking service
nova	nova	Compute service

Note: services running on different nodes, other than the controller node which runs the database will connect remotely to this database.

Before starting the installation of the various components, we also create the service users for each OpenStack service:

```
keystone user-create --name=glance --pass=GLANCE_PASS \
  --email=glance@example.com
keystone user-role-add --user=glance --tenant=service --role=admin
keystone user-create --name=nova --pass=NOVA_PASS --email=nova@example.com
keystone user-role-add --user=nova --tenant=service --role=admin
keystone user-create --name=cinder --pass=CINDER_PASS \
  --mail=cinder@example.com
keystone user-role-add --user=cinder --tenant=service --role=admin
keystone user-create --name=neutron --pass=NEUTRON_PASS \
  --email=neutron@example.com
keystone user-role-add --user=neutron --tenant=service --role=admin
```

```
keystone user-create --name=nova --pass=NOVA_PASS --email=nova@example.com
keystone user-role-add --user=nova --tenant=service --role=admin
```

I. Step 1: Install OpenStack repositories and required software

The following commands must be executed on all three nodes:

```
apt-get install python-software-properties
add-apt-repository cloud-archive:icehouse
apt-get update && apt-get dist-upgrade
reboot
```

II. Step 2: Controller node setup

Install the identity service (*keystone*):

```
apt-get install keystone
```

Configure the identity service (/etc/keystone/keystone.conf):

```
[DEFAULT]
admin_token = XXXXXXXXXXXX
log_file = keystone.log
log_dir = /var/log/keystone
[sql]
connection = mysql://keystone:XXXXXXXXXXXX@localhost/keystone
[identity]
driver = keystone.identity.backends.sql.Identity
[credential]
driver = keystone.credential.backends.sql.Credential
[trust]
driver = keystone.trust.backends.sql.Trust
[os_inherit]
[catalog]
driver = keystone.catalog.backends.sql.Catalog
[endpoint_filter]
[token]
driver = keystone.token.backends.sql.Token
[cache]
[policy]
driver = keystone.policy.backends.sql.Policy
[ec2]
driver = keystone.contrib.ec2.backends.kvs.Ec2
[assignment]
[oauth1]
[ssl]
[signing]
[lldap]
[auth]
methods = external,password,token,oauth1
password = keystone.auth.plugins.password.Password
token = keystone.auth.plugins.token.Token
oauth1 = keystone.auth.plugins.oauth1.OAuth
[paste_deploy]
config_file = keystone-paste.ini
```

Next, we need to create users, roles and tenants:

```
keystone tenant-create --name=admin --description="Admin Tenant"
keystone tenant-create --name=service --description="Service Tenant"
keystone user-create --name=admin --pass=ADMIN_PASS --email=admin@example.com
keystone role-create --name=admin
keystone user-role-add --user=admin --tenant=admin --role=admin
```

The next step, is to create services and API endpoints. All OpenStack services must be registered to the identity service along with their service endpoints. The endpoints are used for accessing the OpenStack services either internally or from external clients:

```
keystone service-create --name=keystone --type=identity --description="Keystone Identity Service"
keystone endpoint-create \
  --service-id=the_service_id_above \
  --publicurl=http://controller:5000/v2.0 \
  --internalurl=http://controller:5000/v2.0 \
  --adminurl=http://controller:35357/v2.0
```

Imaging service is also installed on the controller node:

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

Configure the imaging service `/etc/glance/glance-api.conf` and `/etc/glance/glance-registry.conf`:

/etc/glance/glance-api.conf:

```
[DEFAULT]
admin_token = XXXXXXXXXXXX
log_file = keystone.log
log_dir = /var/log/keystone
[sql]
connection = mysql://keystone:XXXXXXXXXXXX@localhost/keystone
[identity]
driver = keystone.identity.backends.sql.Identity
[credential]
driver = keystone.credential.backends.sql.Credential
[trust]
rbd_store_ceph_conf = /etc/ceph/ceph.conf
rbd_store_user = glance
rbd_store_pool = images
rbd_store_chunk_size = 8
sheepdog_store_address = 10.1.1.10
sheepdog_store_port = 7000
sheepdog_store_chunk_size = 64
delayed_delete = False
scrub_time = 43200
scrubber_datadir = /var/lib/glance/scrubber
image_cache_dir = /var/lib/glance/image-cache/
[keystone_authtoken]
auth_uri = http://10.1.1.10:5000
auth_host = 10.1.1.10
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = glance
admin_password = XXXXXXXXXXXX
[paste_deploy]
flavor = keystone
```

/etc/glance/glance-registry.conf:

```
[DEFAULT]
admin_token = XXXXXXXXXXXX
log_file = keystone.log
log_dir = /var/log/keystone
[sql]
connection = mysql://keystone:XXXXXXXXXXXX@localhost/keystone
[identity]
driver = keystone.identity.backends.sql.Identity
```

```
[credential]
driver = keystone.credential.backends.sql.Credential
[trust]
admin_user = glance
admin_password = XXXXXXXXXXXXX
[paste_deploy]
flavor = keystone
root@ctrl_node:/etc/glance# egrep -v "^#|^$" glance-registry.conf
[DEFAULT]
bind_host = 0.0.0.0
bind_port = 9191
log_file = /var/log/glance/registry.log
backlog = 4096
sql_connection = mysql://glance:XXXXXXXXXXXX@10.1.1.10/glance
sql_idle_timeout = 3600
api_limit_max = 1000
limit_param_default = 25
[keystone_authtoken]
auth_uri = http://10.1.1.10:5000
auth_host = 10.1.1.10
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = glance
admin_password = XXXXXXXXXXXXX
[paste_deploy]
flavor = keystone
```

Populate the glance database:

```
glance-manage db_sync
```

And register the glance service and glance endpoint to the identity service:

```
keystone service-create --name=glance --type=image \
  --description="Glance Image Service"
keystone endpoint-create \
  --service-id=the_service_id_above \
  --publicurl=http://controller:9292 \
  --internalurl=http://controller:9292 \
  --adminurl=http://controller:9292
```

III. Step 3: Compute Controller setup

Install the required software:

```
apt-get install nova-novncproxy novnc nova-api \
  nova-ajax-console-proxy nova-cert nova-conductor \
  nova-consoleauth nova-doc nova-scheduler \
  python-novaclient
apt-get install nova-compute-kvm python-guestfs
apt-get install nova-network nova-api-metadata
```

Configure the nova service:

/etc/nova/nova.conf:

```
[DEFAULT]
admin_token = XXXXXXXXXXXXX
log_file = keystone.log
log_dir = /var/log/keystone
[sql]
```

```
connection = mysql://keystone:XXXXXXXXXXXX@localhost/keystone
[identity]
driver = keystone.identity.backends.sql.Identity
[credential]
driver = keystone.credential.backends.sql.Credential
[trust]
rabbit_password = XXXXXXXXXXXX
my_ip=10.1.1.12
vnc_enabled=True
vncserver_listen=0.0.0.0
vncserver_proxyclient_address=10.1.1.12
novncproxy_base_url=http://192.168.1.10:6080/vnc_auto.html
glance_host=10.1.1.10
network_api_class=nova.network.neutronv2.api.API
neutron_url=http://10.1.1.11:9696
neutron_auth_strategy=keystone
neutron_admin_tenant_name=service
neutron_admin_username=neutron
neutron_admin_password= XXXXXXXXXXXX
neutron_admin_auth_url=http://10.1.1.10:35357/v2.0
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSIfaceDriver
firewall_driver=nova.virt.libvirt.firewall.IptablesFirewallDriver
security_group_api=neutron
volume_api_class = nova.volume.cinder.API
volume_name_template = volume-%s
volume_group = cinder-volumes
libvirt_inject_password=true
libvirt_inject_key=true
[database]
connection = mysql://nova:XXXXXXXXXXXX@10.1.1.10/nova
```

Populate the nova database:

```
nova-manage db sync
```

Create service and service endpoints:

```
keystone service-create --name=nova --type=compute \
  --description="Nova Compute service"
keystone endpoint-create \
  --service-id=the_service_id_above \
  --publicurl=http://controller:8774/v2/%\ (tenant_id\ )s \
  --internalurl=http://controller:8774/v2/%\ (tenant_id\ )s \
  --adminurl=http://controller:8774/v2/%\ (tenant_id\ )s
```

Enable read access to linux kernel (slight security risk, but a requirement!)

```
dpkg-statoverride --update --add root root 0644 /boot/vmlinuz-$(uname -r)
```

and make this change permanent, by creating `/etc/kernel/postinst.d/statoverride` with the following content:

```
#!/bin/sh
version="$1"
# passing the kernel version is required
[ -z "${version}" ] && exit 0
dpkg-statoverride --update --add root root 0644 /boot/vmlinuz-${version}
```

IV. Step 4: WEB GUI Setup

Even though OpenStack network infrastructure is not complete, since *Neutron* is not installed, we can enable the WEB GUI of OpenStack (*Horizon*) in order to perform additional tasks without the need of the command line tools. Horizon can be installed on the controller node.

Install the required packages:

```
apt-get install memcached libapache2-mod-wsgi OpenStack-dashboard
```

Create a self-signed certificate to use with the Apache2 web server:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/apache2/ssl/apache.key -out etc/apache2/ssl/apache.crt
```

Edit `/etc/OpenStack-dashboard/local_settings.py` and insert `"USE_SSL = True"` at the begging of the file in order to enable HTTP over SSL. Next, replace the `"SECRET_KEY"` variable with a random alphanumeric string and define the `OPENSTACK_HOST`, `OPENSTACK_KEYSTONE_URL` and `OPENSTACK_KEYSTONE_DEFAULT_ROLE` variables:

```
OPENSTACK_HOST = "192.168.1.10"
OPENSTACK_KEYSTONE_URL = "http://s:5000/v2.0" % OPENSTACK_HOST
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "Member"
```

Finally, we need to define the database credentials, used by horizon to connect to the MySQL database:

```
DATABASES = {
    'default': {
        # Database configuration here
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'dash',
        'USER': 'dash',
        'PASSWORD': 'XXXXXXXXXX',
        'HOST': 'localhost',
        'default-character-set': 'utf8'
    }
}
```

Apache needs to be configured separately:

`/etc/apache2/conf-available/OpenStack-dashboard.conf`:

```
<VirtualHost *:80>
ServerName 192.168.1.10
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteCond %{HTTPS} off
RewriteRule (.*?) https://%{HTTP_HOST}%{REQUEST_URI}
</IfModule>
<IfModule !mod_rewrite.c>
RedirectPermanent / https://192.168.1.10
</IfModule>
</VirtualHost>
<VirtualHost *:443>
ServerName 192.168.1.10

SSLEngine On
SSLCertificateFile /etc/apache2/ssl/apache.crt
SSLCertificateKeyFile /etc/apache2/ssl/apache.key
SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown

WSGIScriptAlias / /usr/share/OpenStack-dashboard/OpenStack_dashboard/wsgi/django.wsgi
```



```
WSGIDaemonProcess horizon user=www-data group=www-data processes=3 threads=10
Alias /static /usr/share/OpenStack-dashboard/OpenStack_dashboard/static/
<Directory /usr/share/OpenStack-dashboard/OpenStack_dashboard/wsgi>
Order allow,deny
Allow from all
</Directory>
</VirtualHost>
```

V. Step 5: Block Storage Service

Install the required packages. The block storage service (“Cinder”) can be installed on the compute node.

```
apt-get install cinder-api cinder-scheduler cinder-volume
```

Configure the service:

/etc/cinder/cinder.conf:

```
[DEFAULT]
rootwrap_config = /etc/cinder/rootwrap.conf
api_paste_config = /etc/cinder/api-paste.ini
iscsi_helper = tgtadm
iscsi_ip_address = 10.1.1.12
volume_name_template = volume-%s
volume_group = cinder-volumes
verbose = True
auth_strategy = keystone
state_path = /var/lib/cinder
lock_path = /var/lock/cinder
volumes_dir = /var/lib/cinder/volumes
rpc_backend = cinder.OpenStack.common.rpc.impl_kombu
rabbit_host = 10.1.1.10
rabbit_port = 5672
rabbit_userid = guest
rabbit_password = XXXXXXXXXXXX
glance_host = 10.1.1.10
scheduler_driver=cinder.scheduler.filter_scheduler.FilterScheduler
my_ip=10.1.1.12
[database]
connection = mysql://cinder:XXXXXXXXXX@10.1.1.10/cinder
```

/etc/cinder/api-paste.ini:

```
[composite:osapi_volume]
use = call:cinder.api:root_app_factory
/: apiversions
/v1: OpenStack_volume_api_v1
/v2: OpenStack_volume_api_v2
[composite:OpenStack_volume_api_v1]
use = call:cinder.api.middleware.auth:pipeline_factory
noauth = faultwrap sizelimit noauth apiv1
keystone = faultwrap sizelimit authtoken keystonecontext apiv1
keystone_nolimit = faultwrap sizelimit authtoken keystonecontext apiv1
[composite:OpenStack_volume_api_v2]
use = call:cinder.api.middleware.auth:pipeline_factory
noauth = faultwrap sizelimit noauth apiv2
keystone = faultwrap sizelimit authtoken keystonecontext apiv2
keystone_nolimit = faultwrap sizelimit authtoken keystonecontext apiv2
[filter:faultwrap]
paste.filter_factory = cinder.api.middleware.fault:FaultWrapper.factory
```

```
[filter:noauth]
paste.filter_factory = cinder.api.middleware.auth:NoAuthMiddleware.factory

[filter:sizelimit]
paste.filter_factory = cinder.api.middleware.sizelimit:RequestBodySizeLimiter.factory

[app:apiv1]
paste.app_factory = cinder.api.v1.router:APIRouter.factory
[app:apiv2]
paste.app_factory = cinder.api.v2.router:APIRouter.factory
[pipeline:apiversions]
pipeline = faultwrap osvolumeverversionapp
[app:osvolumeverversionapp]
paste.app_factory = cinder.api.versions:Versions.factory
[filter:keystonecontext]
paste.filter_factory = cinder.api.middleware.auth:CinderKeystoneContext.factory
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
auth_host = 10.1.1.10
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = cinder
admin_password = XXXXXXXXXXXX
```

This particular configuration, implies that, a volume group with name “cinder-volumes” already exists in the target machine. Next, we need to populate the cinder database:

cinder-manage db sync

Create the cinder service and endpoint API:

```
keystone service-create --name=cinder --type=volume \
  --description="Cinder Volume Service"
keystone endpoint-create \
  --service-id=the_service_id_above \
  --publicurl=http://controller:8776/v1/%(tenant_id)s \
  --internalurl=http://controller:8776/v1/%(tenant_id)s
```

We also need to register a service and an endpoint for version 2 of the block storage API:

```
keystone service-create --name=cinderv2 --type=volumev2 \
  --description="Cinder Volume Service V2"
keystone endpoint-create \
  --service-id=the_service_id_above \
  --publicurl=http://controller:8776/v2/%(tenant_id)s \
  --internalurl=http://controller:8776/v2/%(tenant_id)s \
  --adminurl=http://controller:8776/v2/%(tenant_id)s
```

VI. Step 6: Networking Service

We installed networking services on a dedicated network node. The scenario we implemented involved the installation of the all the required software on one node, which provides the DHCP, NAT, DNS Masquerading and L3 switching operations. Some necessary software packages were also installed on the compute node, to provide OpenStack networking functionality to the virtual machines. For this scenario, we had to implement Neutron, with one tenant, one private network, one public network and one router.

First, we need to register services and endpoints:

```
keystone service-create --name=neutron --type=network \
  --description="OpenStack Networking Service"
keystone endpoint-create \
```

```
--service-id the_service_id_above \  
--publicurl http://controller:9696 \  
--adminurl http://controller:9696 \  
--internalurl http://controller:9696
```

Next, we need to perform the basic setup of networking services on the network controller and the compute node.

VII.Dedicated network controller node

Install required packages:

```
apt-get install neutron-server neutron-dhcp-agent neutron-plugin-openvswitch-agent  
neutron-l3-agent
```

Enable packet forwarding and disable packet filtering (/etc/sysctl.conf):

```
net.ipv4.ip_forward=1  
net.ipv4.conf.all.rp_filter=0  
net.ipv4.conf.default.rp_filter=0
```

Configure the neutron service:

```
/etc/neutron/neutron.conf:  
[DEFAULT]  
verbose = True  
state_path = /var/lib/neutron  
lock_path = $state_path/lock  
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.OVSNeutronPluginV2  
auth_strategy = keystone  
fake_rabbit = False  
rabbit_host = 10.1.1.10  
rabbit_password = XXXXXXXXXXXX  
rabbit_port = 5672  
rabbit_hosts = 10.1.1.10:5672  
rabbit_virtual_host = /  
notification_driver = neutron.OpenStack.common.notifier.rpc_notifier  
[quotas]  
[agent]  
root_helper = sudo /usr/bin/neutron-rootwrap /etc/neutron/rootwrap.conf  
[keystone_authtoken]  
auth_host = 10.1.1.10  
auth_port = 35357  
auth_protocol = http  
admin_tenant_name = service  
admin_user = neutron  
admin_password = XXXXXXXXXXXX  
signing_dir = $state_path/keystone-signing  
[database]  
connection = mysql://neutron:XXXXXXXXXXXX@10.1.1.10:3306/neutron  
[service_providers]  
service_provider=LOADBALANCER:Haproxy:neutron.services.loadbalancer.drivers.hapr  
oxy.plugin_driver.HaproxyOnHostPluginDriver:default
```

/etc/neutron/api-paste.ini:

```
[composite:neutron]  
use = egg:Paste#urlmap  
/: neutronversions  
/v2.0: neutronapi_v2_0  
[composite:neutronapi_v2_0]  
use = call:neutron.auth:pipeline_factory
```

```
noauth = extensions neutronapiapp_v2_0
keystone = authtoken keystonecontext extensions neutronapiapp_v2_0
[filter:keystonecontext]
paste.filter_factory = neutron.auth:NeutronKeystoneContext.factory
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
auth_host = 10.1.1.10
auth_uri = http://10.1.1.10:5000
admin_tenant_name = service
admin_user = neutron
admin_password = XXXXXXXXXXXX
[filter:extensions]
paste.filter_factory = neutron.api.extensions:plugin_aware_extension_middleware_
factory
[app:neutronversions]
paste.app_factory = neutron.api.versions:Versions.factory
[app:neutronapiapp_v2_0]
paste.app_factory = neutron.api.v2.router:APIRouter.factory
```

/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini:

```
[ovs]
tenant_network_type = vlan
network_vlan_ranges = physnet1,physnet2:512:512
integration_bridge = br-int
bridge_mappings = physnet1:br-ex,physnet2:br-eth0
[agent]
[securitygroup]
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewal
lDriver
[database]
connection = mysql://neutron:XXXXXXXXXXXX@10.1.1.10/neutron
```

/etc/neutron/metadata_agent.ini:

```
[DEFAULT]
debug = True
auth_url = http://10.1.1.10:5000/v2.0
auth_region = RegionOne
admin_tenant_name = service
admin_user = neutron
admin_password = XXXXXXXXXXXX
endpoint_type = http://10.1.1.10:35357/v2.0
nova_metadata_ip = 10.1.1.10
nova_metadata_port = 8775
metadata_proxy_shared_secret = XXXXXXXXXXXX
```

VIII. Dedicated Compute node

Install the required software:

```
apt-get install neutron-plugin-openvswitch-agent openvswitch-datapath-dkms
```

Disable packet filtering (/etc/sysctl.conf):

```
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
```

Configure the neutron services:

/etc/neutron/neutron.conf:

```
[DEFAULT]
state_path = /var/lib/neutron
lock_path = $state_path/lock
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.OVSNeutronPluginV2
auth_strategy = keystone
rpc_backend = neutron.OpenStack.common.rpc.impl_kombu
rabbit_host = 10.1.1.10
rabbit_password = XXXXXXXXXXXXX
rabbit_port = 5672
rabbit_userid = guest
rabbit_virtual_host = /
notification_driver = neutron.OpenStack.common.notifier.rpc_notifier
[quotas]
[agent]
root_helper = sudo /usr/bin/neutron-rootwrap /etc/neutron/rootwrap.conf
[keystone_auth]
auth_host = 10.1.1.10
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = neutron
admin_password = XXXXXXXXXXXXX
signing_dir = $state_path/keystone-signing
auth_url = http://10.1.1.10:35357/v2.0
[database]
connection = mysql://neutron:XXXXXXXXXXXX@10.1.1.10/neutron
[service_providers]
service_provider=LOADBALANCER:Haproxy:neutron.services.loadbalancer.drivers.haproxy.plugin_driver.HaproxyOnHostPluginDriver:default
```

/etc/neutron/api-paste.ini:

```
[composite:neutron]
use = egg:Paste#urlmap
/: neutronversions
/v2.0: neutronapi_v2_0
[composite:neutronapi_v2_0]
use = call:neutron.auth:pipeline_factory
noauth = extensions neutronapiapp_v2_0
keystone = authtoken keystonecontext extensions neutronapiapp_v2_0
[filter:keystonecontext]
paste.filter_factory = neutron.auth:NeutronKeystoneContext.factory
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
auth_host = 10.1.1.10
auth_uri = http://10.1.1.10:5000
admin_tenant_name = service
admin_user = neutron
admin_password = XXXXXXXXXXXXX
[filter:extensions]
paste.filter_factory = neutron.api.extensions:plugin_aware_extension_middleware_factory
[app:neutronversions]
paste.app_factory = neutron.api.versions:Versions.factory
[app:neutronapiapp_v2_0]
paste.app_factory = neutron.api.v2.router:APIRouter.factory
```

/etc/neutron/l3_agent.ini:

```
[DEFAULT]
debug = False
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
external_network_bridge = br-ex
l3_agent_manager = neutron.agent.l3_agent.L3NATAgentWithStateReport
enable_multi_host = True
auth_url = http://10.1.1.10:35357/v2.0
admin_tenant_name = service
admin_user = neutron
admin_password = XXXXXXXXXXXX
metadata_ip = 10.1.1.10
use_namespaces = True
```

IX. Creation of basic networks

The following set of commands can be used to create the public network portion of the networking infrastructure. The public networks are assigned to the default service tenant. First, we need to create the networks and then create subnets under these networks:

```
tenant=$(keystone tenant-list | awk '/service/ {print $2}')
neutron router-create router01
neutron net-create --tenant-id $tenant public01 \
    --provider:network_type flat \
    --provider:physical_network physnet1 \
    --router:external=True
neutron subnet-create --tenant-id $tenant --name public01_subnet01 \
    --gateway 139.91.210.1 public01 139.91.210.0/26
neutron router-gateway-set router01 public01
```

Next, we will create the private network of the CHIC tenant:

```
tenant=$(keystone tenant-list|awk '/demo/ {print $2}')
neutron net-create --tenant-id $tenant net01 \
    --provider:network_type vlan \
    --provider:physical_network physnet2 \
    --provider:segmentation_id 512
neutron subnet-create --tenant-id $tenant --name net01_subnet01 net01 10.1.2.0/24
neutron router-interface-add router01 net01_subnet01
```

X. Step 7: Object Storage

In principle, Object storage (Swift) uses multiple dedicated storage servers and includes set of servers, processes, and rings. Servers, include proxy server operations, server for objects, containers and accounts. Proxy servers provide a unified interface for clients and a single point of communication with underlying storage architecture. Object servers perform simple operations like upload, modify, and retrieve objects. Container servers handle containers, which actually represent the logical structure of the objects, like directories. Account servers provide account manipulation.

Swift was implemented using both Swift All-In-One (SAIO) the distinct servers model. The reason for this was that, we had only one machine capable of providing fast storage services (which hosted the SAIO deployment) but at the same time, we needed a remote storage to implement different policies and test the data localization features of Swift. For our SAIO implementation, we had to install all required software on one machine:

```
apt-get install curl gcc memcached rsync sqlite3 xfsprogs \
    git-core libffi-dev python-setuptools \
    python-coverage python-dev python-nose \
```

```
python-simplejson python-xattr python-eventlet \
python-greenlet python-pastedeploy \
python-netifaces python-pip python-dnspython \
python-mock python-swiftclient python-Swift\
Swift-account Swift-container Swift-object Swift-proxy
```

Swift storage will be emulated on a dedicated filesystem on the object storage server. We setup a single partition on the target system, formatted with the xfs filesystem and mounted under the /mnt directory, with options: *noatime,nodiratime,nobarrier,logbufs=8*.

Swift rings were defined as mapping between the physical location of objects and objects' logical names.

For this project, we emulated a ring of a four node Swift cluster. In order to accomplish this, we create 4 distinct locations under the mount point of the Swift partition, with the following commands (sdc1 represents the partition on the node containing the Swift data):

```
mkdir /mnt/sdc1/1 /mnt/sdc1/2 /mnt/sdc1/3 /mnt/sdc1/4
chown ${USER}:${USER} /mnt/sdc1/*
mkdir /srv
for x in {1..4}; do ln -s /mnt/sdc1/$x /srv/$x; done
mkdir -p /srv/1/node/sdc1 /srv/2/node/sdc2 /srv/3/node/sdc3 \
        /srv/4/node/sdc4 /var/run/Swift
chown -R ${USER}:${USER} /var/run/Swift
for x in {1..4}; do chown -R ${USER}:${USER} /srv/$x/; done
```

\$USER is defined as the default Swift user, in our case "Swift".

Then, we need to setup rsync. Swift uses rsync in order to keep coherent data between Swift cluster servers:

/etc/rsyncd.conf:

```
uid = Swift
gid = Swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 127.0.0.1

[account6012]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/account6012.lock

[account6022]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/account6022.lock

[account6032]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/account6032.lock

[account6042]
max connections = 25
path = /srv/4/node/
read only = false
lock file = /var/lock/account6042.lock
```

```
[container6011]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/container6011.lock
```

```
[container6021]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/container6021.lock
```

```
[container6031]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/container6031.lock
```

```
[container6041]
max connections = 25
path = /srv/4/node/
read only = false
lock file = /var/lock/container6041.lock
```

```
[object6010]
max connections = 25
path = /srv/1/node/
read only = false
lock file = /var/lock/object6010.lock
```

```
[object6020]
max connections = 25
path = /srv/2/node/
read only = false
lock file = /var/lock/object6020.lock
```

```
[object6030]
max connections = 25
path = /srv/3/node/
read only = false
lock file = /var/lock/object6030.lock
```

```
[object6040]
max connections = 25
path = /srv/4/node/
read only = false
lock file = /var/lock/object6040.lock
```

Next, we need to configure the Swift service itself.

/etc/Swift/Swift.conf:

```
[Swift-hash]
Swift_hash_path_prefix = XXXXXXXXXXXX
Swift_hash_path_suffix = XXXXXXXXXXXX
```

/etc/Swift/proxy-server.conf:

```
[DEFAULT]
bind_port = 8080
workers = 1
```



```

user = Swift
log_facility = LOG_LOCAL1
eventlet_debug = true
[pipeline:main]
pipeline = catch_errors healthcheck cache authtoken keystoneauth proxy-server
[filter:catch_errors]
use = egg:Swift#catch_errors
[filter:healthcheck]
use = egg:Swift#healthcheck
[filter:proxy-logging]
use = egg:Swift#proxy_logging
[filter:bulk]
use = egg:Swift#bulk
[filter:ratelimit]
use = egg:Swift#ratelimit
[filter:crossdomain]
use = egg:Swift#crossdomain
[filter:dlo]
use = egg:Swift#dlo
[filter:slo]
use = egg:Swift#slo
[filter:tempurl]
use = egg:Swift#tempurl
[filter:staticweb]
use = egg:Swift#staticweb
[filter:account-quotas]
use = egg:Swift#account_quotas
[filter:container-quotas]
use = egg:Swift#container_quotas
[filter:cache]
use = egg:Swift#memcache
[filter:gatekeeper]
use = egg:Swift#gatekeeper
[app:proxy-server]
use = egg:Swift#proxy
allow_account_management = true
account_autocreate = true
[filter:keystoneauth]
use = egg:Swift#keystoneauth
operator_roles = Member,admin,swiftoperator,_member_
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
delay_auth_decision = 20
signing_dir = /etc/Swift/keystone-signing
auth_version = v2.0
auth_host = 10.1.1.10
auth_port = 35357
auth_protocol = http
auth_url = http://10.1.1.10:35357/v2.0
admin_tenant_name = service
admin_user = Swift
admin_password = XXXXXXXXXX
[filter:cache]
use = egg:Swift#memcache
[filter:catch_errors]
use = egg:Swift#catch_errors
[filter:healthcheck]
use = egg:Swift#healthcheck

```

Finally, we need to create the rings, with the following commands:

```
Swift-ring-builder object.builder create 10 3 1
```

```
Swift-ring-builder object.builder add r1z1-127.0.0.1:6010/sdc1 1
Swift-ring-builder object.builder add r1z2-127.0.0.1:6020/sdc2 1
Swift-ring-builder object.builder add r1z3-127.0.0.1:6030/sdc3 1
Swift-ring-builder object.builder add r1z4-127.0.0.1:6040/sdc4 1
Swift-ring-builder object.builder rebalance
Swift-ring-builder container.builder create 10 3 1
Swift-ring-builder container.builder add r1z1-127.0.0.1:6011/sdc1 1
Swift-ring-builder container.builder add r1z2-127.0.0.1:6021/sdc2 1
Swift-ring-builder container.builder add r1z3-127.0.0.1:6031/sdc3 1
Swift-ring-builder container.builder add r1z4-127.0.0.1:6041/sdc4 1
Swift-ring-builder container.builder rebalance
Swift-ring-builder account.builder create 10 3 1
Swift-ring-builder account.builder add r1z1-127.0.0.1:6012/sdc1 1
Swift-ring-builder account.builder add r1z2-127.0.0.1:6022/sdc2 1
Swift-ring-builder account.builder add r1z3-127.0.0.1:6032/sdc3 1
Swift-ring-builder account.builder add r1z4-127.0.0.1:6042/sdc4 1
Swift-ring-builder account.builder rebalance
```

7 References

All the links to online articles and resources listed below were valid and accessible during the compilation of this report [Last accessed: March 2015].

- [1] "The NIST Definition of Cloud Computing," 2011.
<http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>.
- [2] "Deep sequencing", http://en.wikipedia.org/wiki/Deep_sequencing.
- [3] N. Shomron, "Deep Sequencing Data Analysis," Springer.
- [4] L. Ying-Chih, Y. Chin-Sheng and L. Yen-Jen, "Enabling Large-Scale Biomedical Analysis in the Cloud," *BioMed Research International*, vol. 2013, p. 6, 2013.
- [5] A. Rosenthal, . Mork, . Li, . Stanford, D. Koester and . Reynolds, "Cloud computing: a new business paradigm for biomedical information sharing," *J Biomed Inform*, pp. 342-353, 04 2010.
- [6] M. Penhaker, . Krejcar, . Kasik and . Snášel, "Cloud Computing Environments for Biomedical Data Services," *Intelligent Data Engineering and Automated Learning - IDEAL 2012*, vol. 7435, pp. 336-343, 2012.
- [7] H. Kaplan, M. Cowing and G. Egli, "A primer for data-protection principles in the European Union," 2009.
<http://www.shb.com/attorneys/CowingMark/APrimerforDataProtectionPrinciples.pdf>
- [8] U. D. o. H. a. H. S. (HHS), "Standards for Privacy of Individually Identifiable Health Information", <http://www.hhs.gov/ocr/privacy/hipaa/understanding/coveredentities/introduction.html>
- [9] U. D. o. H. a. H. S. (HHS), "Code of Federal Regulations",
<http://www.hhs.gov/ohrp/humansubjects/guidance/45cfr46.html>
- [10] V. Bonazzi, G. Komatsoulis and P. Bourne, " Biomedical Research as a Connected Digital Enterprise",
http://bd2k.nih.gov/pdf/Documents_for_ADDS_Data_Science_Meeting_the_nih_commons.pdf
- [11] Zhou, Minqi; Zhang, Rong; Zeng, Dadan; Qian, Weining;, "Services in the cloud computing era: a survey," in *4th International Universal Communication Symposium (IUCS)*, IEEE, Shanghai, 2010.
- [12] P. Costa, M. Migliavacca, P. Pietzuch and A. L. Wolf, "NaaS: network-as-a-service in the cloud," in *2nd USENIX conference on Hot Topics in Management of Internet, Cloud and Enterprise Networks and Services*, San Jose, CA, 2012.
- [13] K. Rubenstein, "Cloud Computing in Life Sciences R&D," Cambridge Healthtech Institute, 2010.
- [14] C. w. paper, "Challenges and Opportunities with Big Data",
<https://www.purdue.edu/discoverypark/cyber/assets/pdfs/BigDataWhitePaper.pdf> .
- [15] A. Rosenthal and et al, "Cloud computing: A new business paradigm for biomedical information sharing," *Journal of Biomedical Informatics*, no. 43, p. 342–353, 2010.
- [16] D. Thilakanathan, S. Chenb, S. Nepal and R. L. Calvo, "A platform for secure monitoring and sharing of generic health data in the Cloud," *Future Generation Computer Systems*, vol. 35, pp. 102-113, 2014.
- [17] Microsoft, "MicrosoftHealthVault", <http://www.microsoft.com/en-us/healthvault/>.

- [18] F. Rocha, S. Abreu and M. Correia, "The final frontier: confidentiality and privacy in the Cloud," *IEEE Computer*, vol. 44, no. 9, p. 44–50, 2011.
- [19] W. Jansen and T. Grance, "Guidelines on security and privacy in public cloud computing".
- [20] D. Riley, "Using google wave and docs for group collaboration," *Library Hi Tech News* 27, 2010.
- [21] A. Gellin, "Facebook's benefits make it worthwhile," *Buffalo News*, 2012.
- [22] J. Saarinen, 7 8 2012. <http://www.itnews.com.au/News/311079,ukhealth-trust-fined-for-privacy-breach.aspx>.
- [23] R. Sarathy and K. Muralidhar, "Secure and useful data sharing," *Decision Support Systems*, vol. 1, no. 42, pp. 204–220, 2006.
- [24] D. Butler, "Data Sharing Threatens Privacy," vol. 449, no. 7163, p. 2007.
- [25] L. Feldman, D. Patel, L. Ortmann, K. Robinson and T. Popovic, "Educating for the future: another important benefit of data sharing," *The Lancet*, vol. 379, no. 9829, p. 1877–1878, 2012.
- [26] F. Rocha, S. Abreu and M. Correia, "The final frontier: confidentiality and privacy in the Cloud," *IEEE Computer*, vol. 44, no. 9, pp. 44–50, 2011.
- [27] W. Jansen and T. Grance, "Guidelines on Security and Privacy in Public Cloud Computing," NIST Special Publication 800-144, 2011.
- [28] M. Li, S. Yu, Y. Zheng, K. Ren and W. Lou, "Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 131–143, 2013.
- [29] S.-s. Tu, S.-z. Niu, H. Li, Y. Xiao-ming and M.-j. Li, "Finegrained access control and revocation for sharing data on clouds," in *Parallel and Distributed Processing Symposium Workshops & Ph.D. Forum (IPDPSW)*, 2012.
- [30] HIPAA. U.S. Department of Health and Human Services, <http://www.hhs.gov/ocr/privacy/index.html>.
- [31] J. Saarinen, "UK health trust fined for privacy breach," *Itnews Technology News*, 7 04 2012. <http://www.itnews.com.au/News/311079,ukhealth-trust-fined-for-privacy-breach.aspx>.
- [32] S. SeongHan, K. Kobara and H. Imai, "International Conference on a Secure Public Cloud Storage System," in *Internet Technology and Secured Transactions*, 2011.
- [33] Z. Minqi, Z. Rong, X. Wei, Q. Weining and Z. Aoying, "Security and Privacy in Cloud Computing: A Survey," in *Sixth International Conference on Semantics Knowledge and Grid (SKG)*, 2010.
- [34] A. Duncan, S. Creese and M. Goldsmith, "Insider attacks in Cloud computing," in *IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2012.
- [35] Z. H. C. Deyan, "Data security and privacy protection issues in Cloud computing," in *International Conference on Computer Science and Electronics Engineering (ICCSEE)*, 2012.
- [36] C. Kuner, *European Data Protection Law: Corporate Regulation and Compliance*, Oxford University Press, 2007.
- [37] J. Herveg, "The ban on processing medical data in European law: consent and alternative solutions to legitimate the processing of medical data in healthgrid," in *Proceedings of Healthgrid*, 2006.

-
- [38] I. A. o. P. P. Staff, "European commission sends draft regulation out for review", <https://privacyassociation.org/news/a/european-commission-sends-draft-regulation-out-for-review/>
 - [39] D. OPTIMIS, "Cloud Legal Guidelines: Data Security, Ownership Rights and Domestic Green Legislation", <http://www.optimis-project.eu/sites/default/files/content-files/document/d7213-cloud-legal-guidelines.pdf> .
 - [40] M. Freedman, K. Nissim and B. Pinkas, "Efficient private matching and set intersection," in *Advances in Cryptology-EUROCRYPT 2004*, 2004.
 - [41] V. Danilatu and S. Ioannidis, "Security and privacy architectures for biomedical cloud computing," in *10th IEEE International Conference on Information Technology and Applications in Biomedicine (ITAB '10)*, 2010.
 - [42] T. Dillon, C. Wu and E. Chang, "Cloud computing: Issues and challenges," in *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, 2010.
 - [43] S. SeongHan, K. Kobara and H. Imai, "International Conference on a Secure Public Cloud Storage System," in *Internet Technology and Secured Transactions (ICITST)*, 2011.
 - [44] B. R. Kandukuri, V. R. Paturi and A. Rakshit, "Cloud Security Issues," in *IEEE International Conference on Services Computing*, Bangalore, 2009.
 - [45] "Amazon Elastic Compute Cloud (Amazon EC2)," Amazon, <http://aws.amazon.com/ec2/>
 - [46] "Amazon Simple Storage Service (Amazon S3)," Amazon, <http://aws.amazon.com/s3/>
 - [47] ZDNet, "Rackspace, NASA launch OpenStack: Can it prevent cloud lock-in?," 19 07 2010. <http://www.zdnet.com/blog/btl/rackspace-nasa-launch-openstack-can-it-prevent-cloud-lock-in/36850>
 - [48] OpenStack, "Contributors/Corporate", <https://wiki.openstack.org/wiki/Contributors/Corporate>
 - [49] "The Apache Software Foundation", <http://www.apache.org/licenses/LICENSE-2.0.html> . The Apache Software Foundation.
 - [50] OpenStack, "Summit", <https://wiki.openstack.org/wiki/Summit>.
 - [51] Amazon, "AWS CloudFormation", <http://aws.amazon.com/cloudformation/>
 - [52] OpenStack, "OpenStack Storage", <http://www.openstack.org/software/openstack-storage/>
 - [53] KVM, "Kernel based virtual machine", http://www.linux-kvm.org/page/Main_Page
 - [54] X. Project, "The Hypervisor", <http://www.xenproject.org/developers/teams/hypervisor.html>
 - [55] Wikipedia, "x86", <http://en.wikipedia.org/wiki/X86>
 - [56] Wikipedia, "X86-64", http://en.wikipedia.org/wiki/X86-64#Intel_64
 - [57] Wikipedia, "ARM architecture", http://en.wikipedia.org/wiki/ARM_architecture
 - [58] OpenStack, "OpenStack Compute", <http://www.openstack.org/software/openstack-compute/>
 - [59] "OpenStack, Eucalyptus and Cloudstack EC2 API comparison", <https://wiki.openstack.org/wiki/Nova/APIFeatureComparison>
 - [60] "Open Source Private Cloud Software", <http://www.eucalyptus.com/eucalyptus-cloud/iaas>
 - [61] Amazon, "Amazon Web Services", <http://aws.amazon.com/>.
 - [62] G. O. System, "GNU General Public License", <https://gnu.org/licenses/gpl.html>.
-

- [63] VMware, "Data Center Virtualization and Cloud Infrastructure Products", <http://www.vmware.com/products/datacenter-virtualization/>
- [64] "Eucalyptus FAQ," Eucalyptus, <http://www.eucalyptus.com/faq#q4>
- [65] "Eucalyptus," Amazon, <http://www.aws-partner-directory.com/PartnerDirectory/PartnerDetail?id=3016>
- [66] Eucalyptus, "Eucalyptus and Amazon Web Services Compatibility", <https://www.eucalyptus.com/aws-compatibility>
- [67] "12.1. Amazon Web Services Compatible Interface," Apache Cloudstack, http://cloudstack.apache.org/docs/en-US/Apache_CloudStack/4.2.0/html/Installation_Guide/aws-ec2-introduction.html
- [68] Eucalyptus, "Amazon Web Services EC2 Compatible Interface", https://cloudstack.apache.org/docs/en-US/Apache_CloudStack/4.0.2/html/Installation_Guide/aws-ec2-introduction.html
- [69] Apache Foundation, "Apache™ Hadoop®", <http://hadoop.apache.org/>
- [70] Openstack, "Welcome to Sahara! ", <http://docs.openstack.org/developer/sahara/>
- [71] "oVirt 3.3 release notes", http://www.ovirt.org/OVirt_3.3_release_announcement#OpenStack_and_oVirt:_A_match_made_in_heaven
- [72] "RightScale", <http://www.rightscale.com/2014-cloud-report>
- [73] T. C. o. t. I. C. o. I. T. S. (INCITS), "T10 Working Drafts", <http://www.t10.org/cgi-bin/ac.pl?t=f&f=osd2r05a.pdf>.
- [74] Lustre, "Lustre", <http://wiki.lustre.org>
- [75] "Amazon S3", <http://aws.amazon.com/s3/>
- [76] "HP Cloud Object Storage", <http://www.hpcloud.com/products-services/object-storage?t=faq>
- [77] Openstack SWIFT, <https://wiki.openstack.org/wiki/Swift>
- [78] "Openstack Object Storage API v1 Reference", http://docs.openstack.org/api/openstack-object-storage/1.0/content/ch_object-storage-dev-overview.html
- [79] "Openstack Security Guide", <http://docs.openstack.org/security-guide/content/>
- [80] A. Fox and E. Brewer, "Harvest, Yield and Scalable Tolerant Systems," in Proc. 7th Workshop Hot Topics in Operating Systems (HotOS 99), IEEE CS, 1999.
- [81] W. Vogels, "Eventually Consistent," Queue, vol. 6, no. 6, pp. 14-19, 2008.
- [82] Wikipedia, "MD5", <http://en.wikipedia.org/wiki/MD5>
- [83] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine and D. Lewin, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," in Twenty-ninth Annual ACM Symposium on Theory of Computing, 1997.
- [84] O. Swift, "SAIO - Swift All In One", http://docs.openstack.org/developer/swift/development_saio.html
- [85] O. Foundation, "Open vSwitch concepts", <http://docs.openstack.org/havana/install-guide/install/apt/content/concepts-neutron.openvswitch.html>
- [86] O. Foundation, "Neutron Security Groups" Openstack Foundation, <https://wiki.openstack.org/wiki/Neutron/SecurityGroups>

- [87] A. Computing, "TORQUE Resource Manager",
<http://www.adaptivecomputing.com/products/open-source/torque/>
- [88] A. Computing, "Maui", <http://www.adaptivecomputing.com/products/open-source/maui/>
- [89] "Niflheim Linux supercomputer cluster", <https://wiki.fysik.dtu.dk/niflheim/>
- [90] "Burrows-Wheeler Aligner", <http://bio-bwa.sourceforge.net/>
- [91] "SAMTools", <http://samtools.sourceforge.net/>
- [92] "1000 Genomes; A Deep Catalog of Human Genetic Variation", <http://www.1000genomes.org/>
- [93] OpenMPI, "Open MPI: Open Source High Performance Computing", <http://www.open-mpi.org/>
- [94] P. B.-W. Aligner, <http://pbwa.sourceforge.net/>
- [95] IBM, "Linux virtualization and PCI passthrough",
<http://www.ibm.com/developerworks/library/l-pci-passthrough/>.
- [96] I. Swiftstack, "swiftstack/ssbench," Swiftstack, Inc, <https://github.com/swiftstack/ssbench>.
- [97] "GitHub", <https://github.com/swiftstack/ssbench>
- [98] D. Catteddu and G. Hogben, "Cloud Computing, Benefits, risks and recommendations for information security," EINSa, 2009.
- [99] "OASIS SAML Wiki," OASIS Security Services (SAML) Technical Committee, <https://wiki.oasis-open.org/security/FrontPage>
- [100] O. Foundation, "Specifications", <http://openid.net/developers/specs/>
- [101] "oAuth", <http://oauth.net/>
- [102] N. C. S. R. C. (CSRC), "Role Based Access Control (RBAC) and Role Based Security",
<http://csrc.nist.gov/groups/SNS/rbac/>
- [103] Openstack, " Identity API protection with role-based access control (RBAC)," [Online].
Available: <http://docs.openstack.org/admin-guide-cloud/content/identity-service-api-protection-with-role-based-access-control.html>
- [104] C. M. Initiative. <http://dmtf.org/standards/cloud>
- [105] T. Xie, L. Fanbao and F. Dengguo, "Fast Collision Attack on MD5," Cryptology ePrint Archive, 2013.
- [106] M. Stevens, A. K. Lenstra and B. De Weger, "Chosen-prefix collisions for MD5 and applications," Int. J. Applied Cryptography, vol. 2, no. 4, 2012.
- [107] S. Frankel, K. Kent, R. Lewkowski, A. D. Orebaugh, R. W. Ritchey and S. R. Sharma, "Guide to IPsec VPNs," NIST, 2005.
- [108] "Logical volume management", http://en.wikipedia.org/wiki/Logical_volume_management
- [109] H. Mauelshagen, "LVM 1.0.8 README", <http://ftp.gwdg.de/pub/linux/misc/lvm/1.0/README>
- [110] C. Fruhwirth, "LUKS On-Disk Format Specification Version 1.1," 8 12 2008,
<http://cryptsetup.googlecode.com/svn-history/r42/wiki/LUKS-standard/on-disk-format.pdf>
- [111] "Openstack", <https://wiki.openstack.org/wiki/ObjectEncryption>
- [112] Mirantis. <https://www.mirantis.com/blog/on-disk-encryption-prototype-for-openstack-swift/>
- [113] S. Institute, "Global Information Assurance Certification Paper; Man-In-the-Middle Attack Brief," 200-2002.

- [114] "OSSA-2014-005, Missing SSL certificate check in Python Swift client ", <http://openstack-security-advisories.readthedocs.org/en/latest/advisories/OSSA-2014-005.html>
- [115] "OpenStack Security Notes," <https://wiki.openstack.org/wiki/OSSN> .
- [116] "OpenStack Security Notes, Cinder SSH Pool will auto-accept SSH host signatures by default" <https://wiki.openstack.org/wiki/OSSN/OSSN-0019> .
- [117] "QEMU open source machine emulator and virtualizer," http://wiki.qemu.org/Main_Page .
- [118] J. Morris, "sVirt: Hardening Linux Virtualization with Mandatory Access Control," 2009. <http://namei.org/presentations/svirt-lca-2009.pdf> .
- [119] Wikipedia, "Mandatory access control," http://en.wikipedia.org/wiki/Mandatory_access_control .
- [120] N. S. A. S. S. (NSA/CSS), "SELinux Frequently Asked Questions (FAQ)," <https://www.nsa.gov/research/selinux/faqs.shtml>.
- [121] IETF, "Internet Security Glossary, Version 2," <http://tools.ietf.org/html/rfc4949>.
- [122] T. O. Foundation, "REST Security Cheat Sheet," https://www.owasp.org/index.php/REST_Security_Cheat_Sheet .
- [123] “. S. G. team, ““OpenStack Security Group” team,” <https://launchpad.net/~openstack-oss> .
- [124] “. V. M. t. team. <https://launchpad.net/~openstack-vuln-mgmt> .
- [125] BCN, "Harvard, MIT Broad Institute deploys OpenStack for private cloud," <http://www.businesscloudnews.com/2014/04/22/harvard-mit-broad-institute-deploys-openstack-for-private-cloud/> .
- [126] "Broad Institute of Harvard and MIT," <https://www.broadinstitute.org/>.
- [127] CERN, ““The importance of OpenStack for CERN”, <http://home.web.cern.ch/about/updates/2014/01/importance-openstack-cern> .
- [128] CERN, "The Large Hadron Collider", <http://home.web.cern.ch/topics/large-hadron-collider> .
- [129] OpenStack, "Heat", <https://wiki.openstack.org/wiki/Heat> .
- [130] OpenStack, "Ceilometer", <https://wiki.openstack.org/wiki/Ceilometer> .
- [131] D. TechCenter, "HPC in an OpenStack Environment", http://en.community.dell.com/techcenter/high-performance-computing/b/general_hpc/archive/2014/07/15/hpc-in-an-openstack-environment.
- [132] "The virtualization API", <http://www.libvirt.org> .
- [133] OpenStack, "Enhanced-platform-awareness-pcie," <https://wiki.openstack.org/wiki/Enhanced-platform-awareness-pcie>.
- [134] "CloudStack Administrator's Guide," Apache Cloudstack, http://cloudstack.apache.org/docs/en-US/Apache_CloudStack/4.2.0/html/Admin_Guide/whatis.html .
- [135] "CloudStack Administrator's Guide," Apache Cloudstack, http://cloudstack.apache.org/docs/en-US/Apache_CloudStack/4.0.0-incubating/html-single/Admin_Guide/#whatis .
- [136] "Swauth 1.0.6 documentation", <http://greg.brim.net/swauth/stable/>
- [137] K. Rubenstein. [Online].
- [138] OpenStack, "Harvard University," <http://www.openstack.org/user-stories/harvard-university/>

- [139] A. Rodriguez, "RESTful Web services: The basics," 06 11 2008.
<http://www.ibm.com/developerworks/library/ws-restful/>
- [140] R. Fielding, "Representational State Transfer (REST)," http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm .
- [141] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka and E. Zeidner, "Internet Small Computer Systems Interface (iSCSI)," 094 2004. <http://tools.ietf.org/html/rfc3720> .
- [142] A. N. S. f. I. T. I. C. f. I. T. S. T. G. T11, "Fibre Channel: Backbone - 5 revision 2.00," Fibre Channel over Ethernet.
- [143] VMWare vSphere5, <https://www.vmware.com/support/vsphere5/doc/vsphere-esx-vcenter-server-50-new-features.html>
- [144] VMWare vCloud Automation Center, <http://www.vmware.com/products/vcloud-automation-center>
- [145] VMWare vFabric Application Director, <https://www.vmware.com/files/pdf/vfabric/VMware-vFabric-Application-Director-Datasheet.pdf>

Appendix 1 – Abbreviations and acronyms

<i>ACL</i>	Access Control List
<i>AMI</i>	Amazon Machine Image
<i>API</i>	Application Programming Interface
<i>AWS</i>	Amazon Web Services
<i>CADF</i>	Cloud Auditing Data Federation
<i>CC</i>	Cluster Controller
<i>CLC</i>	Cloud Controller
<i>DaaS</i>	Desktop as a Service
<i>DHCP</i>	Dynamic Host Configuration Protocol
<i>DNS</i>	Domain Name System
<i>EBS</i>	Elastic Block Store
<i>EC2</i>	(Amazon) Elastic Compute Cloud
<i>FC</i>	Fiber Channel
<i>FCoE</i>	Fiber Channel over Ethernet
<i>GRE</i>	Generic Routing Encapsulation
<i>GUI</i>	Graphical User Interface
<i>HA</i>	High Availability
<i>HTTP(S)</i>	Hypertext Transfer Protocol (Secure)
<i>IaaS</i>	Infrastructure as a Service
<i>IAM</i>	Identity and Access Management
<i>ICT</i>	Information and Communication Technology
<i>IDS</i>	Intrusion Detection System
<i>IP</i>	Internet Protocol
<i>iSCSI</i>	Internet SCSI
<i>IT</i>	Information Technology
<i>JSON</i>	JavaScript Object Notation
<i>KVM</i>	Kernel-based Virtual Machine
<i>LDAP</i>	Lightweight Directory Access Protocol

<i>LTS</i>	Long Term Support
<i>LVM</i>	Logical Volume Manager
<i>NaaS</i>	Network as a Service
<i>NAT</i>	Network Address Translation
<i>NC</i>	Node Controller
<i>NIC</i>	Network Interface Controller
<i>PaaS</i>	Platform as a Service
<i>PAM</i>	Pluggable Authentication Module
<i>PKI</i>	Public Key Infrastructure
<i>POSIX</i>	Portable Operating System Interface
<i>RBAC</i>	Role Based Access Control
<i>RDP</i>	Remote Desktop Protocol
<i>REST</i>	Representational State Transfer
<i>S3</i>	(Amazon) Simple Storage Service
<i>SaaS</i>	Software as a Service
<i>SAIO</i>	Swift All-in-one
<i>SAML</i>	Security Assertion Markup Language
<i>SC</i>	Storage Controller
<i>SCSI</i>	Small Computer System Interface
<i>SDK</i>	Software Development Kit
<i>SLA</i>	Service Level Agreement
<i>SOAP</i>	Simple Object Access Protocol
<i>SSL</i>	Secure Sockets Layer
<i>SSO</i>	Single Sign On
<i>UI</i>	User Interface
<i>URL</i>	Uniform Resource Locator
<i>VLAN</i>	Virtual Local Area Network
<i>VM</i>	Virtual Machine
<i>vNIC</i>	Virtual NIC (Network Interface Controller)
<i>VPN</i>	Virtual Private Network

Appendix 2 – List of tables and figures

List of figures

Figure 1 OpenStack components	10
Figure 2 OpenStack Storage Model.....	11
Figure 3 Eucalyptus Components.....	13
Figure 4 Computer nodes for the deployment of Eucalyptus.....	15
Figure 5 Cloudstack nested organization	17
Figure 6 Overall structure of VMWare vSphere.....	19
Figure 7 Overall structure of VMWare vSphere.....	21
Figure 8 Cloud technology adoption and usage.....	24
Figure 9 Private Cloud Usage trend (2014 vs. 2013).....	24
Figure 10 OpenStack architecture.....	26
Figure 11: Account-Container-Object hierarchy	30
Figure 12 Swift cluster operation	31
Figure 13 CHIC OpenStack architecture	32
Figure 14 CHIC OpenStack deployment architecture	33
Figure 15 CHIC cloud cluster network diagram.....	34
Figure 16 CHIC cloud network topology	35
Figure 17 Keystone identity Manager	40
Figure 18 Horizon login screen.....	70
Figure 19 Horizon interface: overview screen	70
Figure 20 Horizon interface: instances.....	71
Figure 21 Horizon interface: volumes	71
Figure 22 Horizon interface: object storage	71
Figure 23 Horizon interface: images & snapshots	72
Figure 24 Horizon interface: networks.....	72
Figure 25 Horizon interface: routers.....	72

List of tables

Table 1 OpenStack components.....	10
Table 2 OpenStack EC2 and S3 API.....	12
Table 3 Eucalyptus components	14
Table 4 Maximum virtual machine instances for the Eucalyptus deployment.....	16
Table 5 Cloudstack components	17
Table 6 Comparative evaluation of OpenStack, Eucalyptus, Cloudstack and VSphere	23

Appendix 3 – OpenStack GUI overview

OpenStack provides a web interface which can handle most of the everyday tasks, like creating networks, instances, images and VM templates, defining security groups and rules, users etc.

This web interface is called **Horizon** and requires a functional web server (preferably Apache). For our convenience, we installed Horizon on the *Controller* node, accessible over HTTPS (port 443 with self-signed certificate):

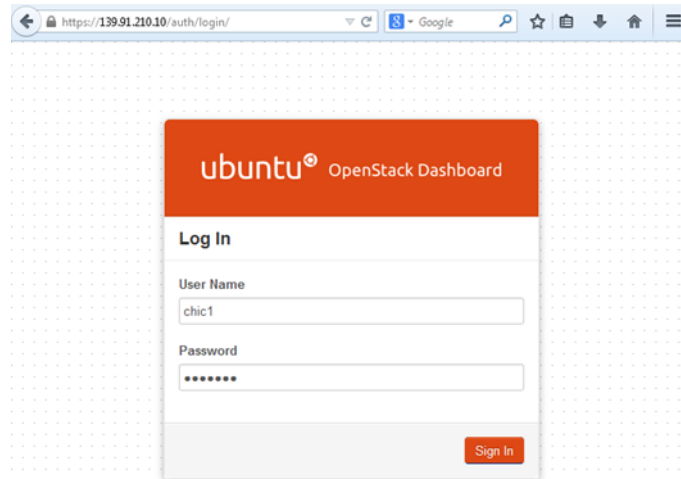


Figure 18 Horizon login screen

After logging into Horizon, the user can perform a set of tasks based on his *role*. A user can act as an administrator, project administrator or project member. An overview of the running system is displayed after the user logs in:

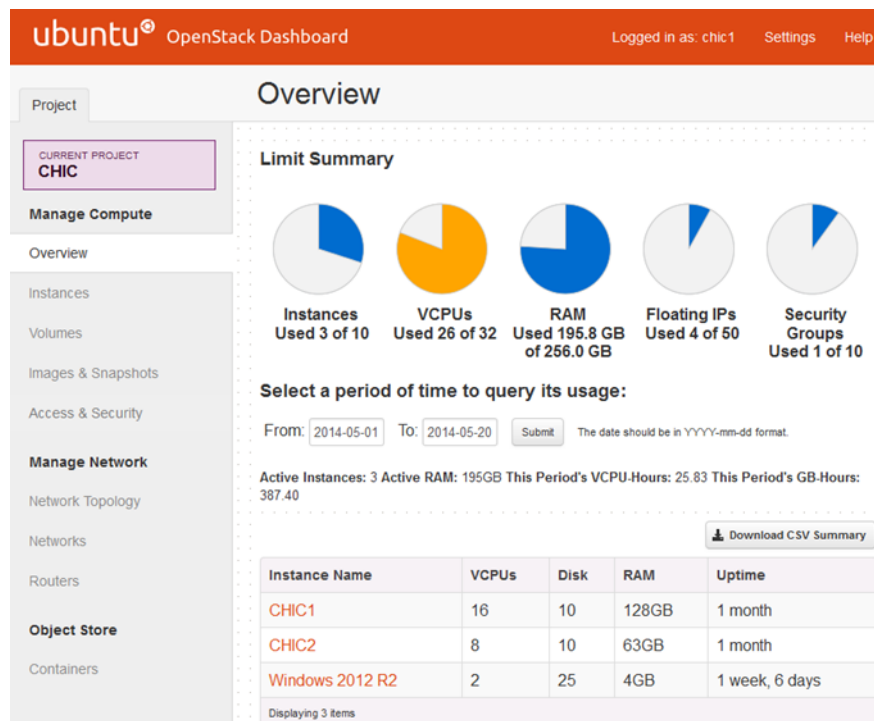


Figure 19 Horizon interface: overview screen

On the left side of the screen, under “Project”, are all the project related operations. We have created one project with the name “CHIC”.

Instances can be manipulated under the “Instances” menu:

Instance Name	Image Name	IP Address	Size	Keypair	Status	Task	Power State	Uptime	Actions
Windows 2012 R2	-	10.1.2.5 139.91.210.25	Small v1 4GB RAM 2 VCPU 25.0GB Disk	-	Shutoff	None	Shutdown	1 week, 6 days	Start Instance More
CHIC2	(not found)	10.1.2.4 139.91.210.24	Medium v1 63GB RAM 8 VCPU 10.0GB Disk	CHIC	Active	None	Running	1 month	Create Snapshot More
CHIC1	-	10.1.2.2 139.91.210.22	Ultra 128GB RAM 16 VCPU 10.0GB Disk	-	Active	None	Running	1 month	Create Snapshot More

Figure 20 Horizon interface: instances

Volumes, are block devices which are allocated to instances. They can be created, allocated or removed under the “Volume” menu:

Name	Description	Size	Status	Type	Attached To	Actions
fee9a50d-df61-478f-b3bf-181658ce2f14		25GB	In-Use	-	Attached to Windows 2012 R2 on vda	Edit Attachments
CHIC2-data		500GB	In-Use	VM Data	Attached to CHIC2 on /dev/vdb	Edit Attachments
CHIC1-data		990GB	In-Use	VM Data	Attached to CHIC1 on /dev/vdb	Edit Attachments
CentOS 6.5 x86_64		10GB	In-Use	VM	Attached to CHIC1 on vda	Edit Attachments

Figure 21 Horizon interface: volumes

In addition to traditional block storage, we have enabled OpenStack’s *object storage* (Swift). We have created three separate object storage containers for testing purposes (*Swift-chic1*, *Swift-chic2* and *Swift-chic3*):

Container Name	Actions
swift-chic1	View Details More
swift-chic2	View Details More
swift-chic3	View Details More

Figure 22 Horizon interface: object storage

Operating system images are handled under “Images & Snapshots”. There is also an option which enables the user to launch an instance directly from images menu:

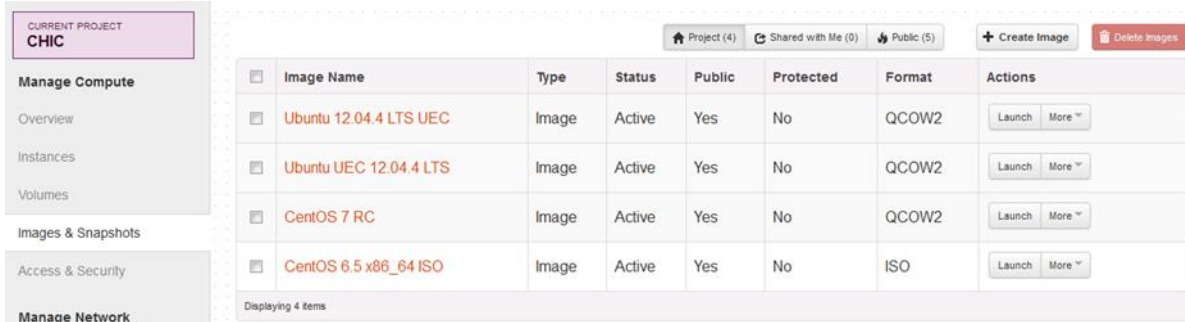


Image Name	Type	Status	Public	Protected	Format	Actions
Ubuntu 12.04.4 LTS UEC	Image	Active	Yes	No	QCOW2	Launch More
Ubuntu UEC 12.04.4 LTS	Image	Active	Yes	No	QCOW2	Launch More
CentOS 7 RC	Image	Active	Yes	No	QCOW2	Launch More
CentOS 6.5 x86_64 ISO	Image	Active	Yes	No	ISO	Launch More

Displaying 4 items

Figure 23 Horizon interface: images & snapshots

Network operations are performed under “Network” and “Routers”:



Name	Subnets Associated	Shared	Status	Admin State	Actions
net01	net01_subnet01 10.1.2.0/24	Yes	ACTIVE	UP	
public01	public01_subnet01 139.91.210.0/26	Yes	ACTIVE	UP	

Displaying 2 items

Figure 24 Horizon interface: networks



Name	Status	External Network	Actions
router01	Active	public01	Clear Gateway More

Displaying 1 item

Figure 25 Horizon interface: routers