CORE
Metadata, citation and similar papers at core.ac.uk
Provided by DSpace at Tartu University Library

UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Andre Tättar

Building and Configuring a Custom Private Cloud Using Consumer Hardware

Bachelor's Thesis (9 ECTS)

Supervisor: Artjom Lind, MSc

Tartu 2016

Privaatpilve ehitamine olemasolevatest ressurssidest

Lühikokkuvõte: Pilve kasutamine on tänapäeval moodne trend erinevatele organisatsioonidele. Selle ajendiks on effektiivsus, sest pilv lubab kasutada olemasolevaid ressursse kõige paindlikumal ja effektiivseimal viisil. Bakalaureusetöö aluseks oli kapis seisev madala astme riistvara, mida keegi ei kasutanud. Töö põhipanus on see, et olemasolev riistvara ehitati üles privaatpilveks. Kuna privaatpilve sobib pigem kõrgklassi/serveri tasemel riistvara, siis tekitab madala taseme riistvara mõningaid probleeme, kuid töö näitab, et nendest võib üle vaadata, kuna boonuseid on rohkem kui negatiivseid aspekte. Töö lõpptulemuseks on töötav OpenStack implementatsioon, mida on kerge kasutada igaühel, mis on igalt poolt kättesaadav ja piisavalt paindlik täitmaks erinevaid Tartu Ülikooli hajussüsteemide uurimisrühma vajadusi. Saadud infrastruktuur on kergelt skaleeruv ning füüsiliste masinate lisamine võtab vähem kui 30 minutit. Lisaks tagab OpenStack selle, et projektide ja kasutajate haldus on väga kerge ning teadlased saavad teha virtuaalmasinaid vähem kui minutiga.

Võtmesõnad: OpenStack, Mitaka, Ubuntu, privaatpilv

CERCS: T120 - Süsteemitehnoloogia, arvutitehnoloogia

Building and Configuring a Custom Private Cloud Using Consumer Hardware

Abstract: Moving into the cloud is a common trend for organizations to use existing hardware in an efficient way. organizations specifically use private or hybrid clouds. Existing unused hardware was the main problem in this thesis. The Contribution of this thesis was a description of how to build and configure OpenStack using consumer grade hardware. Using low-end hardware to build a working private cloud does have some disadvantages, but in our case, it was not that critical. In the end, there were more advantages than disadvantages. The end result is a working implementation of OpenStack, which is easy to use, accessible from anywhere in the world and flexible enough to fill the needs of Distributed Systems in University of Tartu platforms. This thesis and configurations in appendix provide a scalable solution - additional compute nodes can be setup with less than 30 minutes. Creation of virtual machines takes less than a minute by using a web

interface, which is very easy to understand and use.

Keywords: OpenStack, Mitaka, Ubuntu, private cloud

CERCS: T120 - Systems engineering, computer technology

Contents

1 Introduction 6
1.1 General Overview . . . 6
1.2 Scope . . . 6
1.3 Objective . . . 6
1.4 Contribution . . . 7
1.5 Roadmap . . . 7

2 Related Work 8
2.1 Literature overview . . . 8
2.2 Cloud . . . 9
2.2.1 Private cloud - advantages & disadvantages . . . 10
2.3 Cloud Software "first decisions" . . . 11
2.4 Cloud Computing software . . . 11
2.5 Hypervisor . . . 12
2.6 Operating System . . . 13
2.7 OpenStack . . . 13
2.7.1 OpenStack Services . . . 13
2.7.2 Storage projects . . . 15
2.7.3 Shared Services . . . 16
2.7.4 Higher-level service . . . 18
2.8 Extra services . . . 18
2.9 OpenStack specific terms . . . 19
2.10 Network . . . 20
2.10.1 Osi Layer 2 and 3 . . . 20
2.10.2 Routing . . . 20
2.10.3 VLAN . . . 22

3 Solution design 23
3.1 Context and Analysis . . . 23
3.2 Constraints . . . 24

4 Solution Implementation and Results 28
4.1 Network setup . . . 28
4.2 Setup . . . 30
4.3 Benchmarks . . . 31
4.3.1 Read/Write . . . 31
4.3.2 Network latency and bandwidth . . . 31
4.3.3 Comments on results . . . 33

5 Conclusion **36**

1 Introduction

1.1 General Overview

A common trend in computing is that services and organizations are moving into the cloud, more specifically cloud computing. Every person who has a network capable device probably uses the cloud - be it Dropbox for storage or Gmail for email. Moving into the cloud has many benefits, most notably scaling, reduced cost, less maintenance, increased availability and flexibility. Cloud is especially useful with a current business model, where startups are popular and need huge scaling over small time periods.
In the context of this thesis, private clouds are examined. Private clouds allow organizations to use existing or new resources in the most general way. Physical machines limit the use cases for hardware - basically one system per one instance. Cloud uses virtualization and allows to create any kind of needed resource. For example, one can have Windows, OS X and Linux operating systems running in one cloud, which is especially useful if developing applications for Windows, but all resources use Linux. This gives a lot of flexibility to organizations to create any kind of system for testing, developing and enterprise systems.

1.2 Scope

Last year I was a member of the team who represented the University of Tartu in the ISC'15 conference Student Cluster Competition and I was the team leader [1]. We were given hardware for the competition by our sponsors and after the competition this hardware was unused. As a consequence, the idea occurred in my mind to make use of this material and this is how the thesis work came to life.

1.3 Objective

The main objective of this thesis dissertation was to utilize existing resources such as old or unused hardware and bring it back to life by using them as an infrastructure for customized private cloud. From this perspective, the idea was to create and design a new private cloud for distributed system platforms with a friendly end user interface. In order to achieve the objective, OpenStack was used as basis platform. Therefore, it was needed to redesign and configure the network. Moreover, creating the friendly interface was achieved by OpenStack dashboard, which gives nice web interface for users with no knowledge of system administration, is accessible from anywhere and allows to create projects, where users can handle their own infrastructure.

1.4 Contribution

In this thesis the main contribution can be resolved to:

- Friendly user interface for researchers, without any knowledge about system administrating.
- Easy project management and creation.
- Custom setup by using clever networking.
- Easy scaling for administrators.
- Review of OpenStack services and their utilization.
- Advantages and disadvantages of the proposed solution.
- Ways to improve the design and customize even further.

1.5 Roadmap

In this section we present how this thesis dissertation is organized. The thesis contains five chapters and they are as follows:

- The first chapter provides a general overview of the thesis.
- Chapter 2 contains related work. Overview on related theses and papers will be given and a discussion of software used. OpenStack is described in detail, a lot of manuals have been summarized to give a good overview on how it is implemented and what different services do. Networking concepts used in OpenStack will be explained.
- Chapter 3 is for solution design - hardware, software used will be provided and detailed discussion on what the constraints are.
- Solution implementation is described in chapter 4, more specifically how to setup network, how to make many networks on one interface and lastly benchmarks are given.
- Future work will be discussed in chapter 5. There main upgrades for private cloud are discussed.
- In the end there is conclusion.

2 Related Work

2.1 Literature overview

High performance computing is moving into the clouds and much research has been done for that field, for example, the author in [2] did a similar task to this thesis – he built a private cloud for Mobile Cloud Lab which is research group of University of Tartu. He used OpenStack and stated that OpenStack is more stable than Eucalyptus. The author used KVM as hypervisor and stuck to default configurations/instructions wherever possible. The author used server grade hardware for their small private cloud setup with 2 network interfaces. The author experienced problems regarding local I/O and solved the problem with using raid0, but this was due to the constraint of having to create 25 VMs in parallel.

The decision on which hypervisor to use has been a discussion topic for many years and one author who contributed to this was Allan [3], who measured the performance of KVM and Xen. This is particularly interesting for this thesis, because hypervisor has to be chosen if one wants to build a cloud. The author has results which show that KVM performs better than Xen. Additionally, it is shown that virtualization does not add computational or memory overhead, but has big impact of input-output performance. The author also says that for HPC in the cloud, there should not be many virtual machines on one node, because if performance is critical, then instance count should be kept low also.

Every system goes through a process where system implementation is designed, one good paper on this is [4]. The paper provides valuable information on the whole setup for private HPC cloud. The authors built a private HPC Cloud with Xen, Eucalyptus, GlutserFS, Walrus and Puppet. They explained every choice they made and provided alternatives and reasons why they picked something. The most interesting was the comparison of virtualization technologies, where they compared Xen, KVM, VMVare ESX and Virtualbox. They point out three challenges for HPC Cloud:

1. The Virtualization overhead – virtualization adds another stack to the system, for example, Xen adds the Hypervisor. Low latency applications that are common in HPC are not effective on the cloud, because of latency caused by virtualization.

2. Systems administrating for end-users – researchers might have to set up their own environment and also have to maintain the system. This might be challenging for some and it also requires time.

3. Application-programming models – current technology is designed for HPC clusters.

Moreover, another benefit behind virtualization can be seen in the reduction of electricity consumption, improvement of utilization, performance isolation, increasement of availability, fault tolerance, ease of management, system security and flexibility [5].

Another aspect of using unused resources could be the decision of using a cluster versus cloud. There is a good paper on this [6], where the authors did research into using community cluster versus Amazon EC2. The main thing for the cloud is that there is no queue for the users. The authors compared performance of traditional cluster node and cloud (virtualized) cluster node, with the NAS benchmark, mostly CPU bound but also influenced by network performance. They found that the cloud nodes are faster. They reported some cons of the system – it is hard to build images for the cloud from scratch and software has to be manually installed (some might come with the image provided though). They stated that the cost model for cloud is different – the user has to pay for CPU cycles, disk storage, IO, IO requests and additional other fees. In community cluster the user only has to pay for CPU usage. Also, the cluster hardware prices were included. In the end, they stated what they found out:

1. HPC cloud is very useful if cluster queues are full and you have time-sensitive work.

2. Lone researchers without others to share the initial cluster building cost and low IT staff support can benefit from a private HPC cloud.

3. They state that using a community cluster costs 0.15-0.25 dollars, but it would cost 1.15-1.75 dollars per instance-hour, cloud is approximately 7 times more expensive.

4. Cloud is 40% faster than a community cluster.

2.2 Cloud

Cloud is a very loose term in computer science. More precisely when people talk about clouds, they refer to cloud computing and that means using resources that one can connect to over a network. These resources could be anything from virtual machine instances and virtual servers to mailhosts and databases hosted somewhere. Often people have no idea where their resources are physically located and only know the external Pi address of their instance and use the IP address. Usually when people talk about the cloud they refer to public clouds like Amazon

EC2 or Microsoft Azure. Generally clouds benefit from less IT management, accessibility, high computing power, lower costs, scalability and availability.
A great use case example for the cloud could be the following: a user has a very CPU intensive task and he/she does not have the resources for it, maybe it takes a year to run this task on the user's machine, the user uses the cloud which finishes the job in 12 hours. Now the user only has to pay for those 12 hours used.
Second use case - on a vacation, one does not want to carry data storage drives with them and use network storage to store their data like pictures, music and videoclips. Here is where the cloud comes to rescue and offers network storage services like Dropbox, Google Drive and Microsoft OneDrive.

2.2.1 Private cloud - advantages & disadvantages

Private cloud is a cloud resource that is managed, built and used only by one or more organizations. Access to private clouds is usually limited to organizations and their partners.
Private cloud advantages over the public cloud are the following:

Greater Reliability Because private clouds belong to one organization, they have greater control over it and can secure it exactly how they want. This means that more fault-tolerant and secure network can be created for the private cloud.

Greater flexibility Services and instances that run on a private cloud are there for a reason and admins can utilize resources in a better way. This also means better performance, because usually public clouds are overcommitted.

Going green Companies might have underutilized hardware, like a server that uses only 5% of maximum performance. Putting these resources in a private cloud and creating fully utilized hardware will mean less energy consumption and less maintenance.

Utilize unused hardware Existing hardware that nobody is using could be used to make a private cloud. This will provide good testing/developing tool for many purposes. This is also the case for this thesis.

Data security Data security is a concern when using public clouds, because they are likely targets of hacking - big names like Dropbox, Amazon Cloud Services have had their security breached. Using a private cloud, organizations have much more control over their data. Additionally, depending on where the data is physically located, it is applicable for different data security laws.

Private cloud disadvantages over Public cloud are following:

Greater cost Creating private clouds is costly, hardware costs a lot and know-how is also expensive.

Easier scalability If very high scaling is needed, it is easier to do it in public clouds, just click some buttons and instances are added. In a private cloud, at some point, physical hardware has to be added and configured for scaling.

More maintenance In private clouds, organizations have to take care of all hardware and underlying networks, however, there are no hardware and network maintenance costs in public clouds.

2.3 Cloud Software "first decisions"

Deploying any cloud is a challenging task and it should start with decisions regarding software. When making decisions, the primary concern is whether the tool is open source or not. The aim is to use only open source software. Secondary, the tool must have good support, because we are making a custom solution. Thirdly, the tool should be highly customizable.

2.4 Cloud Computing software

Firstly, an open-source cloud computing solution has to be chosen. There are many options for this, most popular and supported being OpenStack, CloudStack and Eucalyptus. It is currently most supported with a great user base. From the beginning we expected a custom setup and OpenStack had most resources on it with latest releases.

- OpenStack – open source software that controls small to large sized compute, storage and networking resources, managed by a dashboard. OpenStack is highly scalable and customizable solution, with a huge amount of documentation. Current installation guides are for openSuse, Suse Linux Enterprise server, RHEL 7, CentOS 7 and Ubuntu 14.04 [7].

- CloudStack – open source cloud computing software designed to deploy and manage small and big clusters of virtual machines while being easily scalable infrastructure as a service platform. It has a lot of features – a full and open native API, network-as-a-service, compute orchestration, authentication management, resource accounting and great user interface. Latest installation manuals are for Ubuntu 14.04, Ubuntu 12.04, CentOS 6 & 7 and RHEL [8] .

- Eucalyptus – free and open source cloud computing software, mainly developed for building AWS (Amazon Web Services) compatible private and

hybrid cloud environments. Their official documentation shows support for CentOS 6 and RHEL [9].

As a related work pointed out, Eucalyptus is rather hard to use and we will not aim at building AWS compatible clouds. Therefore, we can rule Eucalyptus out. The decision between OpenStack and CloudStack comes down to support and manuals. OpenStack while being a more mature solution, has more manuals and is more supported than CloudStack.

2.5 Hypervisor

OpenStack limits our choices on hypervisors. OpenStack Mitaka lists these hypervisors [10]:

- KVM (Kernel-based Virtual Machine) - Open source full-virtualization solution for Linux that has near-native performance using virtualization extensions (AMD-v and Intel VT).
- LXC (Linux Containers) – Would enable to use Docker, but limits the use cases of the cloud because using containers, because each container has to use the same kernel as host.
- QEMU – Enables emulation of almost any hardware. Generally used for testing due to performance being not as good as with Xen or KVM. QEMU can be used with KVM/Xen for near native performance.
- UML (User Mode Linux) – Generally used for testing, performance is not as good as with KVM or Xen.
- VMWare vSphere – This is not open source, so we will not consider this.
- Xen (Using libvirt) – Xen using libvirt would be required if XCP is not supported. This method is not well supported and tested [11].
- XenServer/XCP (Xen Cloud Platform) – XenServer is a commercial version of XCP by Citrix. XCP uses the Xen hypervisor. Xen hypervisor has near native performance using hardware virtualization extensions (AMD-v and Intel VT). XCP adds functionality to Xen hypervisor [12].
- Hyper-V – This is not open source, so we will not consider this.

2.6 Operating System

There are many options here as well.

- OpenSuse – Open-source operating system and well documented in OpenStack documentation. Does not have a very good Xen support and documentation.
- Suse Enterprise Server – Not open-source.
- CentOS 7 – Open-source operating system and well documented in OpenStack. CentOS is first choice for XCP, because of that, there is a lot of documentation on CentOS, Xen and XCP.
- Red Hat Enterprise Linux 7 – Not open-source.
- Ubuntu – According to Ubuntu, more than 55% of all OpenStack deployments use Ubuntu [13]. A lot of documentation written for Ubuntu. Ubuntu is also open-source OS and both Xen and KVM work well with Ubuntu.

2.7 OpenStack

The following paragraph is a summary of the extensive OpenStack manual about its services, which is available here [7].
Since OpenStack is a huge project, more information on this should be given. OpenStack supports all types of cloud environments and they aim for a simple implementation, massive scalability and an extensive list of features. OpenStack provides Infrastructure-as-a-Service solution through many modules, each module has its own Application Programming Interface. Some modules are core components, which have to be set up, some are optional components. OpenStack distributes these modules into differet services, which are called: Service, Storage, Shared services and Higher-level services.

2.7.1 OpenStack Services

OpenStack services are as follows:

- Dashboard – Project name is Horizon. This is the graphical web-based user interface of OpenStack and links together other OpenStack services in order to view, create and manage resources. It provides easy interface for a lot of services:
  - Authentication – Log in as admins/users
  - Upload and manage images

  - Configure access and security for instances – This includes creation of key pairs, security groups with rules
  - Launching, managing, tracking, creating snapshots of images
  - Creating and managing networks

- Compute – Project name is Nova. Manages compute instances in OpenStack. Consists of different services as well. Nova-api service – link between dashboard and other nova services. It is the nova controller. Endpoint for all API calls (OpenStack API, EC2 API, Special Admin API for privileged users).

  - Nova-compute service – This daemon runs on all compute nodes. Creates virtual machines through hypervisor APIs. Also responsible for termination. Example hypervisors are XenApi for XenServer/XCP and libvirt for KVM/QEMU. Daemon accepts orders to launch a KVM virtual machine and updates its metadata in database.
  - Nova-scheduler service – When a virtual machine is created, the request is handled by nova-scheduler and it decides on which compute node to host this virtual machine.
  - Nova-conductor module – Link between the interaction of nova-compute services and the database. Does not allow nova-compute to access the cloud database directly.
  - Nova-consoleauth daemon – Authorizes tokens to use console proxies for users. Requirement for nova-novncproxy daemon.
  - Nova-novncproxy daemon – Together with nova-consoleauth daemon provide a remote console or remote desktop access to a running instance through a VNC connection. Supports browser-based novnc clients.
  - Nova client – Allows users to submit commands as a tenant administrator or end user.
  - The Queue – Central hub for messages passed between daemons. Usual implementations are with RabbitMQ.
  - SQL database – Stores information about build-time and run-time states, like instance types available, instances in use, networks and projects. Support is for SQL-Alchemy databases, most used are sqllite3 (testing and developing only), MySQL and PostgreSQL.

- Networking – Project name is Neutron. Goal is to provide network connectivity as a service between OpenStack services, most importantly Nova

compute services. It implements the Neutron API. API enables users to define and create networks, subnets and ports that other OpenStack services can use and attach compute instances to networks. Neutron supports Layer 2 and Layer 3 networking. Consists of the following services:

  - Neutron API server – Accepts and routes Layer 2 networking, IP address management and Layer 3 router construct to appropriate OpenStack Networking plug-in for action.
  - OpenStack Networking plug-in and agents – creates networks or subnets, provides ip addressing, plugs and unplugs ports. Plug-ins and agents differ depending on vendor and technologies used. Common agents are L3 agent, DHCP and a plug-in agent.
  - Messaging queue – Routes information between neutron-server and all kinds of agents. Most OpenStack installations use it. Also stores networking state for some plug-ins.

2.7.2 Storage projects

OpenStack has several storage projects. Most used are:

- Object Storage – Project name is Swift. This is an optional module of OpenStack, a multitenant object storage system, capable of managing large amounts of unstructured data at low cost through a RESTful API. Replicates and writes objects/files on multiple drives, which make this a highly scalable, fast and fault tolerant project. Has many components.
  - Proxy servers – Accepts requests to upload files, modify metadata and creation of containers.
  - Account servers – Administers accounts created by Object Storage.
  - Container servers – Administers mapping of folders and containers inside Object Storage.
  - Object servers – Administers real objects like files on storage nodes.
  - Various preiodic processes – Perform cleaning tasks on data storage. Ensures consistency and availability when replicating.
  - WSGI middleware – Performs authentication, this is usually Keystone.
  - Swift client - Allows users to communicate to REST API through CLI.
  - Swift-init – Responsile for running the script to initialize building of the right file.

  - Swift-recon – CLI tool that is responsible for gathering metrics and telemetry data.
  - Swift-ring builder – Utility that builds and rebalances storage rings.

- Block Storage – Project name is Cinder. This is an optional module of OpenStack. Block storage devices for guest virtual machines. Capable in many configurations and drivers such as NFS, iSCSI, CEPH, NAS/SAN and more. Consists of 5 components:
  - Cinder-api – Redirects correct API requests to cinder-volume for action.
  - Cinder-volume – Communicates with processes like cinder-scheduler. Uses message queue to interact with services. Responds to I/O requests sent to Block Storage service.
  - Cinder-scheduler daemon – Determines on which Cinder node to create volumes. Similar to Nova-scheduler.
  - Cinder-backup daemon – Responsible for backing up all types of volumes to a backup storage provider.
  - Messaging queue – Used for messaging between Block Storage processes.

2.7.3 Shared Services

OpenStack has 3 shared services. They are listed here:

- Identity Service – Project name is Keystone. It is responsible for authentication and authorization for other OpenStack services. Other OpenStack services use Keystone as a common unified API. Can be integrated with existing authentication services like LDAP. Each service used in the OpenStack cloud must be registered with Keystone. Consists of following components [11]:
  - Server – Using the RESTful API provides authentication and authorization through a centralized server.
  - Drivers – Allows the Keystone server to be used with existing/other authorization infrastructure like a database or LDAP server.
  - Modules – These middleware modules live in the OpenStack services which use Keystone. Using the Python Web Server Gateway Interface, these modules extract user credentials from service requests and then send them to a centralized server for authorization.

- Image Service – Project name is Glance. It stores and manages virtual machine disk images. Accepts and handles API requests for disk or server images and metadata definitions. It also supports storage of images on different repository types, for example, normal file system folder, nfs folder and object storage. Image service is made of the following components:
  - Glance-api – Accepts Image API calls for image storing, retrieving or discovering.
  - Glance-Registry – Handles calls regarding metadata (size, type, token, etc) about images.
  - Database – Stores image metadata. Various options for database, most popular being MySQL.
  - Storage repository for image files – Handles storing of disk/server images. Supports normal file systems, network file systems, object storage, RADOS block devices, HTTP and Amazon S3.
  - Metadata definition service – A standardized API for users, admins, services and vendors in order to define their own custom metadata. Metadata is used with different services. The Metadata consists of a unique key, description, name, constraints, resource types, size and some more.
- Telemetry – Project name is Ceilometer. This is an optional package. It collects event and metering data from other OpenStack services. This data can be used for billing. Consists of two services:
  - Telemetry Data Collection service – Efficiently gathers metering data related to other OpenStack service and stores them.
    * Compute agent – Runs on compute nodes and gathers resource utilization data.
    * Central agent – Gathers information on central management server regarding utilization statistics for other services except compute instances.
    * Notification agent – Agent on one to many central management servers that keeps an eye on message queue(s) to gather event and metering data.
    * Collector – Sends collected unmodified Telemetry data to data store or external consumer and runs on one to many central management servers.
    * API server – Runs on one to many central management servers to allow data access from data store.

- Telemetry Alarming service – Raises alarms when collected data breaks defined rules/quotas.
    * API server – Runs on one to many central management servers in order to access data store information regarding alarms.
    * Alarm evaluator – Runs on one to many central management servers in order to determine when alarms are raised. It uses statistic trends over a time period to determine when some threshold will be crossed.
    * Notification Listener – Determines when to fire alarms on a central management server. Alarms are generated automatically depending on the defined rules against an event.
    * Alarm notifier – Runs on one to many central management servers and executes alarms based on threshold evaluations.

2.7.4 Higher-level service

Optional higher-level services are not core components, but make life easier for integration or orchestration.

- Orchestration – Project name is Heat. This project makes it available to use higher-level template-based orchestration for describing cloud applications that get created through OpenStack API calls. Heat integrates other OpenStack components into a one-file template system. One can create almost all OpenStack resources with Heat like virtual machines, assign IP addresses, volumes, security groups, users and more. Has advanced functionality to increase availability and scaling. Consists of 4 components:
  - Heat command-line client – Link between Heat-api and AWS CloudFormation APIs.
  - Heat-api – Native REST API for OpenStack that sends API requests to heat-engine using Remote Procedure Call (RPC).
  - Heat-engine – Coordinates template launches and gives responses to API consumer.

2.8 Extra services

There are extra services that OpenStack uses and are required to have a working cloud. These are not maintained by OpenStack and several options for each service are available. Services used are:

- Network Time Protocol (NTP) – networking protocol for clock synchronization. All OpenStack nodes should have the same system time and NTP servers are used for that. It is highly recommended that controller nodes reference some other trusted NTP servers and all other nodes have controller as their NTP server. One suggested implementation that OpenStack uses is Chrony.
- SQL database – Most OpenStack services store their data on SQL databases. It should be placed inside the controller node. OpenStack supports many databases, most notably MySQL and its fork project MariaDB and also other popular ones like PostgreSQL.
- Messaging queue service (MQ) – It is used in distributed systems to send and receive messages. It uses a queue as a data structure in order to temporary store messages when destination service is not responding or is currently busy. OpenStack makes use of messaging queues to coordinate information about status and operations among services. Typically runs on controller node and many MQ services are supported including RabbitMQ, Qpid and ZeroMQ. Most used and supported is the RabbitMQ.
- Memcached – distributed memory caching. OpenStack uses Memcached to cache tokens for authentication. As this service is used in authentication, only OpenStack servers should have access to it. This could be achieved with firewalls, authentication and encryption. The Main functionality of Memcached is to make central authentication faster.

2.9 OpenStack specific terms

These terms are defined with the help of OpenStack Dev Ops manual [14].

Rings – Used in Swift to determine data location between Object Storage nodes.

Endpoints – Url that can be used in OpenStack to access some service.

Service – Any OpenStack service in Openstack like Nova, Neutron or Keystone.

Roles – Roles define the actions users can do.

Tenants – Groups of users like Service tenants, where each service has their own user.

Regions – Regions are usually separated OpenStack environments, sharing only Keystone service for authentication.

Security groups – list of firewall rules that are applied to the virtual machines.

Floating IP address – IP address that is assigned to a virtual machine (VM) once will be given to the same VM each time it boots. Creation and handling of floating IP pools are handled by Neutron.

2.10 Network

Table 1: Osi Model table

OSI Model		
Layer		Protocol data unit (PDU)
Host layers	7. Application	Data
	6. Presentation	
	5. Session	
	4. Transport	TCP/UDP
Media layers	3. Network	Packet
	2. Data link	Frame
	1. Physical	Bit

2.10.1 Osi Layer 2 and 3

Open Systems Interconnection model (OSI Model) in table 1 is a standardized reference model which describes how communication is handled between computers/networks.

Layer2: Data link layer "The data link layer provides error-free transfer of data frames from one node to another over the physical layer, allowing layers above it to assume virtually error-free transmission over the link" [15]. Bridging and switching are operated on this layer.

Layer3: Network layer "The network layer controls the operation of the subnet, deciding which physical path the data should take based on network conditions, priority of service, and other factors" [15]. The most important task of Layer 3 is routing.

2.10.2 Routing

Routing is a fundamental part of any network. IP routing means that if a host has to send a packet to some other host, routing table will be consulted. The

routing table is just a table of routes. Usually, hosts that are connected to at least one route, know three routes. Host knows about itself and this is usually referred to as localhost. The host knows about locally connected other hosts in a LAN network, where hosts are in the same subnet and can communicate with each other. Thirdly, host knows about everything else, which can be referred to as default gateway - every packet that host has to send, which is not himself or in the same subnet, will be sent to the default gateway.
Network address translation (NAT) was designed to conserve IP address space

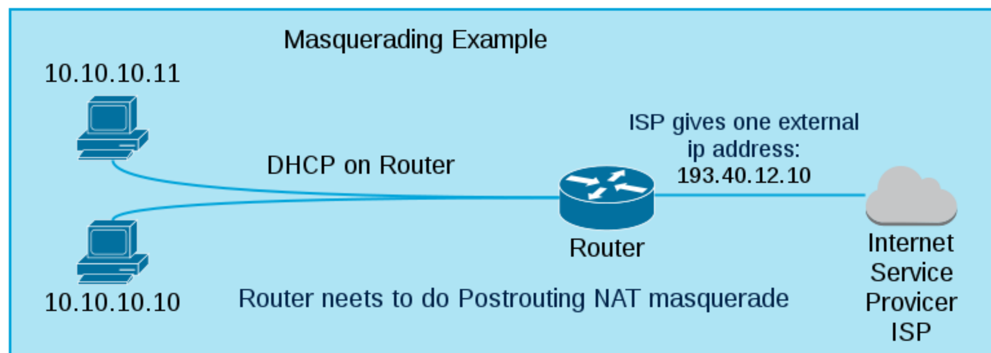

Figure 1: Simple Masquerade example

by translating private IP addresses (like ones from 10.0.0.0/8 subnet) to public IP addresses by clever usage of ports. NAT enables to translate a group of private IP addresses into one public IP address. For example, all traffic that goes through university network, will be seen as coming from one IP address. The basic example of this can be seen in figure 1. To achieve this, postrouting is required. Postrouting is altering of packets when they go out from an interface.
One thing that goes together with postrouting to achieve the example is source NAT (SNAT) or masquerade. Source NAT is the act of changing the IP header in the source address packet. Masquerade is a special kind of SNAT, because it is used when the source IP address is not known. If for example router has DHCP configured and IP addresses are dynamic in the network, masquerade has to be used. Command to achieve NAT postrouting masquerade is very simple. It is shown on figure 2.

```
1 root@pc:~# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Figure 2: Postrouting masquerade code

On figure 2 we specify the table(-t) as `nat`. The table specifies in what way packet

matching is done. `-A POSTROUTING` means that the command should be appended to the `POSTROUTING` chain. `-o eth0` means that if packets are moving out from eth0, then apply this rule. `-j MASQUERADE` means to do masquerading on the packets that leave the interface.

2.10.3 VLAN

VLAN (Virtual local area network) is a virtual broadcast domain defined in the data link layer (OSI Layer 2). Because it is virtual, it can exist on top of many LANs. One great thing about VLAN is that it enables to create multiple subnets on one ethernet port. Secondly, VLANs enable the creation of separate domains on top of existing physical network.
Common use is to create a VLAN for management and one public VLAN, so that public networks don not see the traffic on management VLAN. One VLAN usually defines their own subnet, so one broadcast domain per subnet. This easily provides a lot of manageability and security to networks. Example of this is shown on figure 3. On the figure, workstations cannot see any traffic that is being sent over management vlan1. Administrators however have access to both services vlan and management vlan. Switches on the figure are not included in neither of vlans, because if someone were to connect a cable to the switch, they would not be able to see vlan traffic. This is for security reasons. Additionally, the server can be accessed from workstations, but it would be easy to configure which services can be accessed from services vlan and which cannot. This example should clarify the use cases of vlans and explain why it is very easy to make secure networks with vlans.
Routing between VLANs is required if different VLANs define their own subnets. This is the case in most situations, because one to one mapping of VLANs and subnets is often used. For all kinds of routing OSI Layer 3 capable device has to be used, this can be a router or even a L3 switch (switch with some L3 routing abilities). In any network, there exists a gateway, let's call it a router. Routing between VLANs will happen on these gateway machines, if there exists a route between gateways, a rule for routing between VLANs can be added. Basically VLANs act like any other network and same rules apply to them (including routing). They just can have the ability to define many networks on the same network interface.

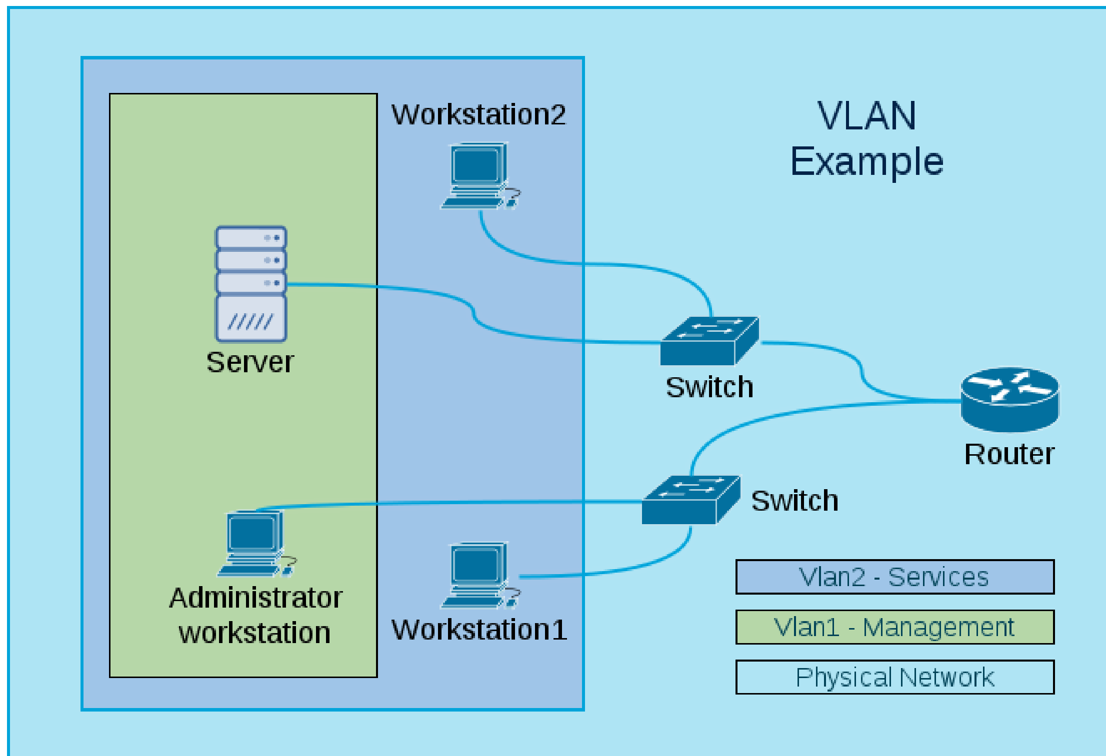

Figure 3: Simple VLAN example

3 Solution design

3.1 Context and Analysis

In this section, presentation and analysis of the requirements given by Distributed System Group are written. This later can be resumed as follows:

1. In System Administration (MTAT.08.021) course a virtual machine is given to each student. Currently there are 81 students registered to the course and additionally there are virtual machines for testing and for teachers to use. This means that there are around 90 virtual machines. There exists a very custom setup right now, which is not scalable and is very hard to maintan. For example, if a virtual machine of one student is down, it requires much work to find out on where it is located.

2. In Distributed Systems Seminar, flexibility is required, as graduate students have a need to test different systems or develop systems that are meant to be used in a distributed manner. For example, a docker based solution was

used to load balance between 5 servers.

3. In every research group there is a need to value their research activities by visualization and simulation. Therefore, a private cloud can be used to host their machines that show simulations or that are used as backends like databases for simulations.

4. Researchers need flexibility similar to DS Seminar - they research different topics and have various needs.

5. Easy scaling, both up and down, is required. Scaling down is required because there are no courses in the summer, so there is no need for a system to run on high utilization. Scaling up is required because there can come courses, where around hundred virtual machines have to be set up.

6. Easy maintenance is a must. There are no full time system administrators in Distributed Systems Group and maintaining a hundred VMs can be a very time consuming task.

We hope to satisfy every constraint given here by using OpenStack. In terms of flexibility OpenStack can use lxc for Docker, KVM/Xen to start any instance and so on. This covers flexibility requirements and is very easy to setup. OpenStack by nature is also easily scalable and maintainable so OpenStack is a good solution to use.

3.2 Constraints

In the practical part of this thesis access to three nodes, each has one ASUS A58M-A motherboard, one AMD APU, 2x4GB RAM, PSU, SSD hard drive, was available. The hardware list can be seen in table 2. One port of master CCR1016 router is connected to CRS125 switch that is only used for this thesis. Figure 4 states that controller and compute nodes should have 2 network interfaces (NICs), other constraints are satisfied.

This setup faces one mayor problem: ASUS A58M-A has only one network interface (NIC). This means that we have to create virtual networks and basically fake OpenStack into believing that we in fact have 2 networks.

OpenStack is built in a way that has one management network, there all of OpenStack services like Nova and Neutron run and for example Keystone, Swift, Cinder can only be accessed from management interfaces. This provides extra security and better performance in clouds, because it separates underlying management communication from virtual instances communication.

The other network interface (NIC) is used as unbound and is actually left unconfigured on setup/configuration phase. The second NIC will be configured by

Table 2: Hardware list

Hardware		Amount
Motherboard	ASUS A58M-A	3
Processor	AMD APU A8-7600K	3
CPU Cores	3.1 GHz	4
RAM	4 GB Kingston HyperX	6
Harddrive	Samsung SSD 840 EVO	3
Power supply unit (PSU)	ARLT PSU 500 W 85%	3
Router	Mikrotik CCR1016-12G	1
Switch	Mikrotik CRS125-24G-1S-RM	1
Eth link speed	1 Gigabit Ethernet (1GbE)	
Avg Combined power usage	75 W	

neutron on controller and neutron-linuxbridge-agent on compute nodes.
Additionally, our setup will not setup any storage nodes. First of all, because there simply is no hard drives. Secondly, because this is currently a testing environment, usage of much storage space is not expected and everything will fit on the hard drives of the nodes.

This thesis follows the Mitaka installation for Ubuntu. OS of choice is Ubuntu Server 14.04 LTS. Hypervisor of choice is KVM. Additionally, we will use MariaDB for SQL database, Chrony as NTP Server, RabbitMQ as a message queue and Memcached. That is a basic setup and after configuring these services, the next task is to set up networking.
The choice for OS was rather easy. Debian is the preferred OS in DS group with courses teaching Debian, machines running on Debian and personnel is used to it. Since Ubuntu is based on Debian, we use Ubuntu as our OS. As a bonus, Ubuntu and Debian have great community support with friendly irc chatrooms. Openstack also has a lot of content on setting up with Ubuntu and support forums are filled with Ubuntu related issues. KVM as a hypervisor was chosen because according to related work, KVM has near-native performance and is said to be better than Xen [3]. Additionally KVM is used in OpenStack installation guide for Mitaka [16].
OpenStack also has a default overcommiting ratio, which means that OpenStack allows to use more virtual resources than physically available[14]. The default overcommitment ratio is:

- CPU allocation ratio 16:1
- RAM allocation ration 1.5:1

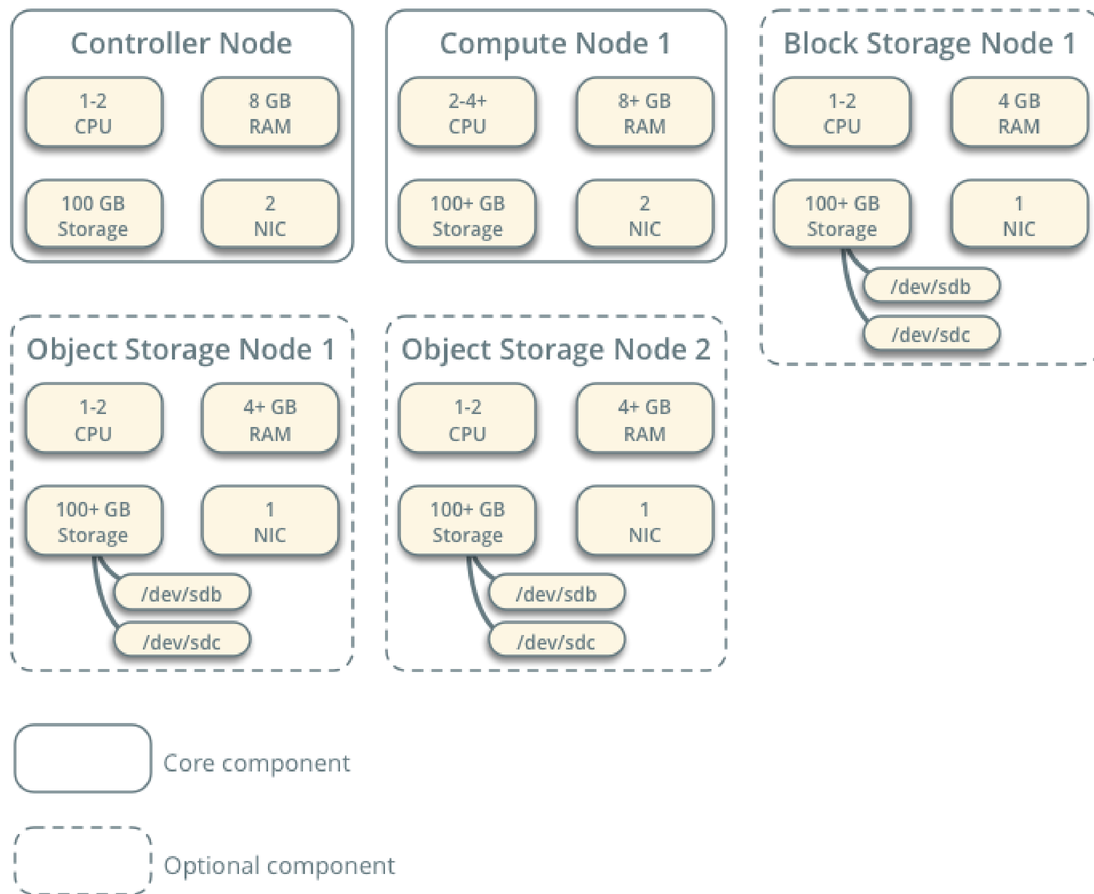

Figure 4: Recommended OpenStack configuration [16]

This means, that OpenStack allows allocating 16 times more virtual CPUs. AMD APU has 4 physical cores, so OpenStack would allow using up to 48 Physical cores. The case is similar to RAM, instead of 8 GB, 12 GB is allowed to allocate.
The idea behind overcommitting is that virtual machines usually do not use all the resources that they were given. Heavy overcommitting has negative effect on performance. If OpenStack default overcommitting was taken into consideration, one node could support up to 20 instances. One thing to keep in mind here is that overcommitting does not mean that you can use more RAM than physically available. This solution is also very efficient in terms of power consumption. Each node uses around 75 W and if one instance uses 1 virtual CPU and 512MB RAM (like instances given to System Administration students), then we could run around 12-16 instances on one node. Heavy overcommitting is not optimal for this thesis,

because the hardware is consumer grade and overheating could be a problem. The course is usually taken by 60-70 people which means that a controller plus 5 nodes would be sufficient. Additional nodes for other purposes could also be added. That means just for the course, energy consumption would be 450 W, which is very acceptable.

4 Solution Implementation and Results

First steps are very easy:

1. Set up and configure basic router for simple ethernet network - 10.11.12.0/24
2. Set up switch, basic configuration, change password
3. Set up hardware and install Ubuntu on each node. The naming convention should be kept in mind, hostname should be `controller` for management node and `computeX` (X is a number starting from 1) for compute nodes.

4.1 Network setup

```
1 # apt-get install vlan <- Install vlan packages
2 # modprobe 8021q <- Load 8021q module into the kernel
3 # echo "8021q" >> /etc/modules <- Load 8021q mod on boot
4 # nano /etc/network/interfaces <- Configure network interfaces
5 # ifup vlan10 && ifup vlan2 <- Bring up vlan interfaces
```

Figure 5: Configuring Network Interfaces

Like mentioned before, 2 separate networks have to be created for OpenStack. VLANs are used for this purpose. Setting up VLANs is easy. Following commands from figure 5 is all that is needed to set it up. I named my VLANs as vlan2 (unbounded) and vlan10 (management) as can be seen in figure 6 and network configuration files are in the appendix [17].
Routing for vlan10 has to be done next. There are two options here:

1. Configure router as a gateway - This means that vlan10 has to be created on the router. It is less secure and puts extra overhead on the router.
2. Configure controller as a gateway - This means that all vlan10 traffic will be routed through controller, which is the gateway for us. Communication that is happening between OpenStack nodes will not reach the router, but will remain inside the switch and nodes. This provides us some extra security. This is a preferred approach for us, there is no need to occupy router resources, when there is no need. Management will mostly communicate inside vlan10 anyway. However, this means that controller node will have to act as a router.

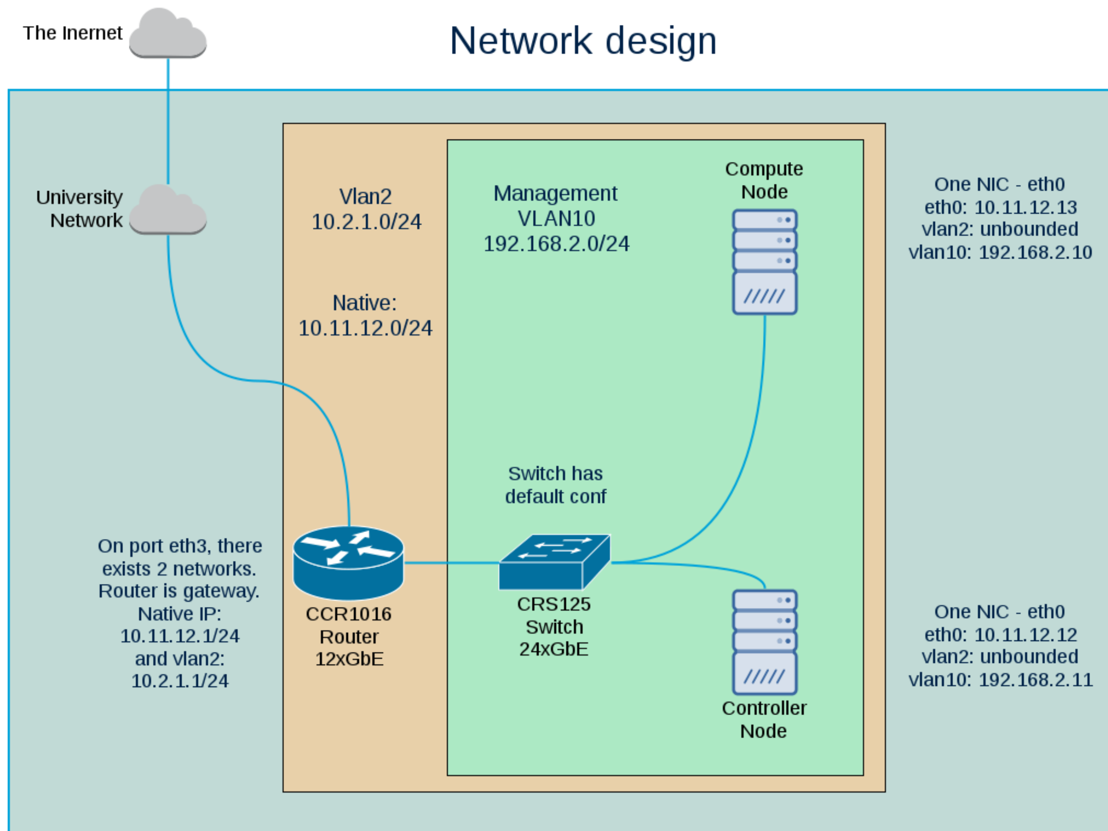

Figure 6: Network Topology

Configuring controller to be a router is a bit harder task. All these commands can be seen in the appendix [17]. Basic things that we do:

1. Allow ipv4 routing on the controller node
2. Allow ipv4 routing on each boot (make it persistent)
3. Allow routing from vlan10 to eth0
4. Allow routing from eth0 to vlan10
5. Add postrouting masquerade to eth0
6. Save firewall rules and make script to restore firewall rules on startup
7. Test compute node connectivity to web, `ping 8.8.8.8` is a good example. All nodes except controller should have default gateway as controller node's VLAN IP address.

4.2 Setup

Official OpenStack install guide for Ubuntu is straightforward and easy to follow [16]. I did setup everything according to the guide. Couple of noteworthy things here:

- There are a lot of (around 20 to 25) different usernames and passwords to keep track of.
- Hundreds of configuration options have to be set. Pay very good attention to them, typos will cause most of the errors.
- All errors that I had were either typos or small parts, which were overlooked and not configured.
- OpenStack services show running status and will not say if they had any errors, logs or verification commands are of help here.
- When debugging, the option –debug should be used for more information, for example `openstack --debug compute service list`.
- When seeing errors, check configuration files two times. If errors still persist, go over the configuration files one more time. Most of the errors are misconfigurations.
- OpenStack IRC and support forums provide answers relatively quickly.
- When configuring network, keep in mind that our provider network will be on vlan2 interface.

After a successful installation, but before launching images, vlan2 network has to be set up because neutron expects a gateway. Here routing for unbound VLAN (vlan2) has to be configured, which is managed by neutron. Routing for vlan2 however has to be done from the router. Lets say that CRS125 switch is connected to ether10 on CCR1016 router. Following things have to be done for this:

1. Create vlan2 on the router interface ether10
2. Create subnet on vlan2 and assign IP address (this is the gateway)
3. Add source NAT masquerading for the subnet created in step 2. Specifically mark source address as the subnet and outbound interface as the ether1. Here ether1 is the port on which connection to external networks is established.

Finish up installation with creating subnets with Neutron. Make sure the subnets created with Neutron are subnets of the network set up on vlan2 by router. Finish up by powering on some instances.

4.3 Benchmarks

As mentioned before in the related work it was proven over and over how virtualization does not add computational or RAM overhead. As a concequence, computational and RAM performance benchmarking is not needed. However, the read/write and network performance will be investigated, because this has been reported as being pitfalls of virtualization [3] [2].
Commands used for benchmarking were ping for latency, iperf for bandwidth, hdparm for read speeds and for write speeds. Command on figure 8 was used. Additionally iperf was used to create heavy loads on networks.

```
time sh -c "dd if=/dev/zero of=testfile bs=128k count=10k && sync";
rm testfile
```

Figure 7: Command used to measure write speed

4.3.1 Read/Write

The write speed test shown on figure 8 shows the real time of data writing. Syncing shows the real time, otherwise a higher number would be shown, which uses caching. We measure how long it takes to write file to a disk. The size of the file is 1.28 GB. Here the test is run first on physical machine, then one virtual machine, then two virtual machines up to six virtual machines. We see that write scales up well and splits speed between virtual machines nicely, load is balanced almost perfectly.

The read speed test uses hdparm with -t and -T parameters for testing and output the memory bandwidth for buffered reads and cached reads. Buffered reads - "speed of reading through the buffer cache to the disk without any prior caching of data" [18]. Cache reads - "speed of reading directly from the Linux buffer cache without disk access" [18]. The graph is shown on figure 9. This is similar to the case of write speeds, KVM does perform very well for disk reads and writes.

4.3.2 Network latency and bandwidth

The bandwidth test was very simple. We used public iperf server iperf.eenet.ee to achieve these results. We tested network bandwidth with iperf command, they are shown on figure 10. The results show that using VLANs and controller node for routing does add a bit of overhead, but is not very huge. Overall bandwidth is fine.

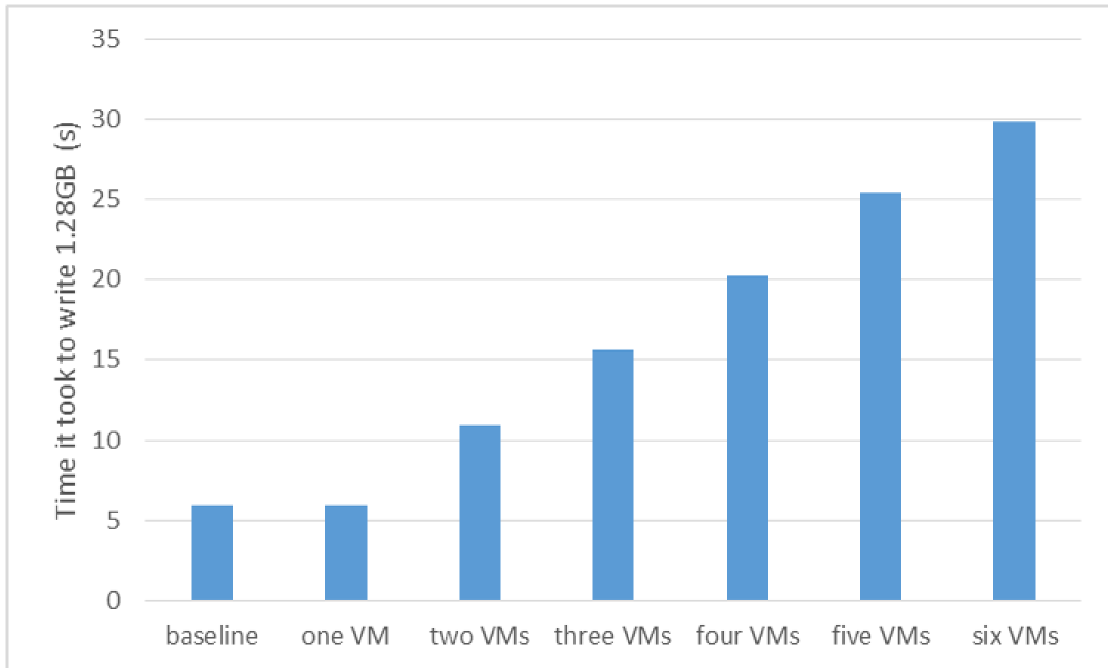

Figure 8: Time it takes to write a file of size 1.28GB

Laptop, virtual machine and controller only have 6 hops to reach the destination, while compute node has one additional hop - controller node.

Last test was the latency test, the aim of this test is to measure management VLAN latency when there are network tests ongoing. Average latency is given, which is measured over 20 tests. Additionally standard error for tests is given. Because test descriptions are long for figure 11, they are provided here:

Test1 - Baseline for latency, measured while no load on network. Pinging is between controller and compute2 node.

Test2 - Pinging is between compute2 and controller node, while virtual machine that is on compute2 node and another virtual instance on compute1 are creating heavy load. Compute2 has the virtual machine, which acts as a client for iperf.

Test3 - Pinging is between compute1 and controller node, while virtual machine that is on compute2 node and another virtual instance on compute1 are creating heavy load. Compute1 has the virtual machine, which acts as a server for iperf.

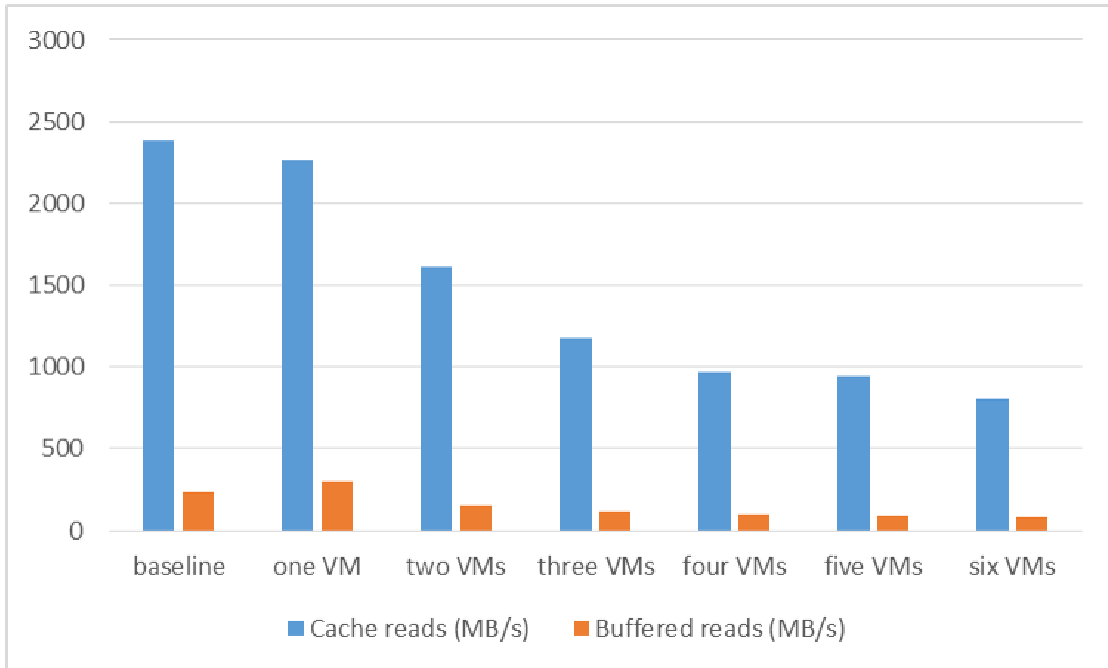

Figure 9: Cached reads and Buffered reads

Test4 - Pinging is between controller and compute1, while virtual machine on compute1 acts as a client iperf for public iperf server - iperf.eenet.ee.

Test5 - Pinging is between controller and compute2, while virtual machine on compute1 acts as a client iperf for public iperf server - iperf.eenet.ee.

Test6 - Pinging is between controller, compute1 and compute2, while virtual machine on compute1 node and virtual machine on compute2 node have iperf clients running for public iperf server - iperf.eenet.ee.

We see that only one test adds a lot of latency to management VLAN. So the pitfall definately is the network here, but only in some cases heavy loads between virtual machines would have to be created. This is a disadvantage of this system that OpenStack tries to avoid by having separate physical networks for management and virtual machines.

4.3.3 Comments on results

Our proposed private cloud is not meant for heavy workloads. For good networking performance, researchers should use the HPC Center of University of Tartu to do network heavy tasks. Nevertheless, network is a troubling part of proposed

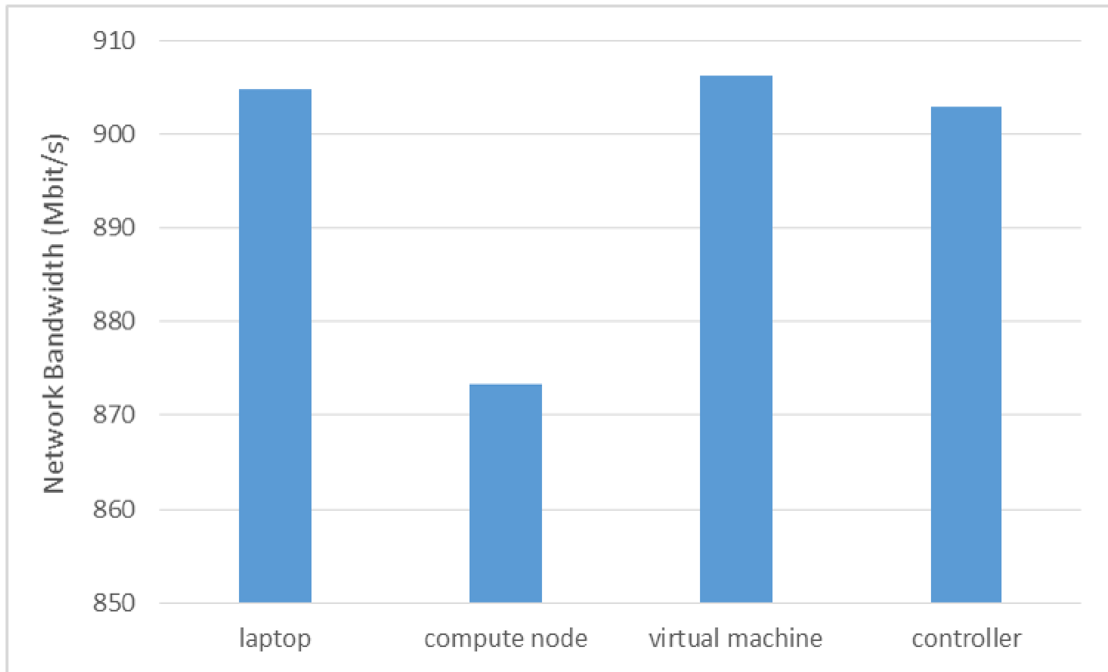

Figure 10: Network Bandwidth from various instances

system.
Input/output for proposed system is completely fine and in general use case, heavy read/writes are not expected on the system. Great performance for input/output is provided by using SSDs.

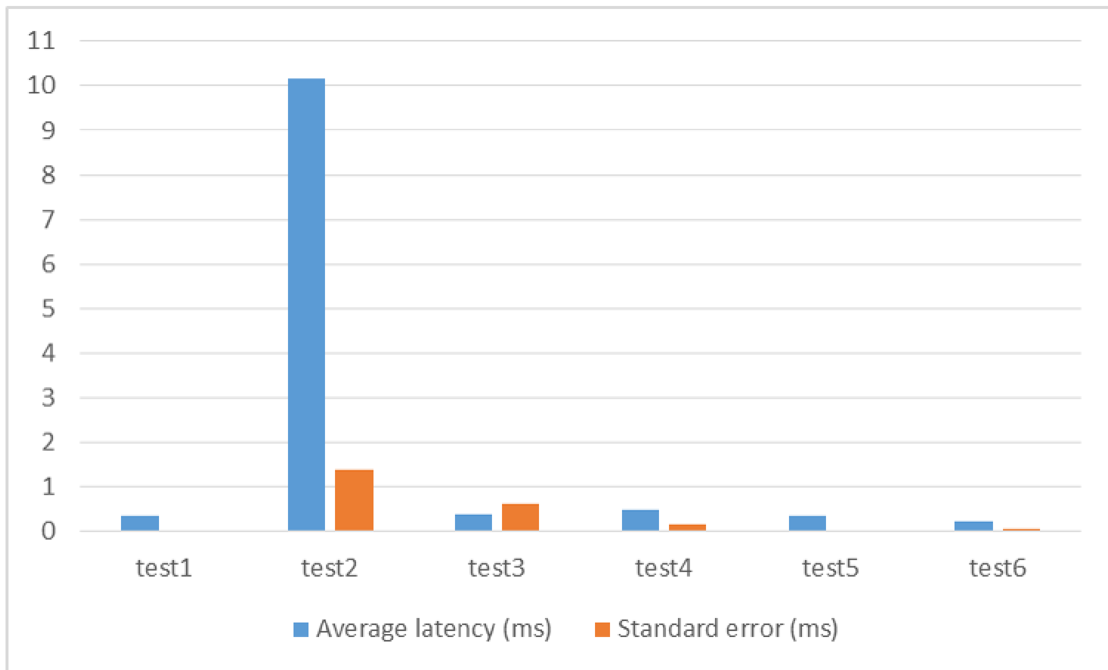

Figure 11: Network latency tests

5 Conclusion

The goal of this thesis was to use existing unused hardware into a good use for Distributed Systems group in the University of Tartu. This goal was achieved by building a private cloud using OpenStack. In OpenStack manuals, it is stated that it requires two network interfaces at least to work: however, this thesis pruposes a design with only one network interface. There appeared to be no resources on how to achieve this task and a solution was proposed to use VLANs to create two virtual networks. They act as a management network and an unbounded network which would be configured by Neutron. The proposed solution does have some pitfalls, like latency drop on the management network, when there is heavy traffic on virtual machines. This, however, is not so critical for a testing/developing private cloud and can be overlooked.
For future work, the testing setup can be upgraded to a better one by using new resources in the future. Here is a list of the main things which can be improved upon:

- Storage - Access to 8TB Storage server will be given and the Distributed Systems group can use it to safely store Glance images and virtual machine disk files, like qcow2.

- Diskless setup - There is ongoing work to achieve diskless boot, so if we set up RAID on the controller node and put compute node filesystems on controller, then compute nodes will have NFS root file system and will boot with persistant configurations. That means it would not be necessary to use one SSD per node and gives better scaling. This is the current thesis topic of a fellow student [19].

- Management - Using configuration management tools it is possible to automate provisioning of OpenStack components and virtual machines. As a result, there will be no necessity in doing all configuration steps manually next time. This is the current thesis topic of a fellow student [20].

- Custom instances - Creation of custom images has to be researched. This is due to the fact that System Administration course requires that each student makes their instance from nothing. This requires more research into what exactly is needed for System Administration course.

- Network performance after upgrading system - Right now the system is working fine, but how will network perform if additional services like management will be running. Additional overhead will be caused by the fact that instances will be stored on storage servers and if compute node root file systems are on some other node in RAID configuration. This will require testing.

- APU GPU part - Additional power could be added by GPU which is currently eating little RAM, but it is not used. Research into utilizing GPUs will be needed.

References

[1] "International Supercomputing 2015 conference Student Cluster Competition - Team Tartu." http://www.hpcadvisorycouncil.com/events/2015/isc15-student-cluster-competition/team_tartu.php. Accessed: 2016-05-10.

[2] V. Kadakas, "Establishing Scientific Computing Clouds on Limited Resources using OpenStack." Bachelor thesis, 2013. University of Tartu.

[3] A. Trukits, "The Cost of Virtualization for Scientific Computing." Bachelor thesis, 2013. University of Tartu.

[4] J. Y. S. Moussa Taifi, Abdallah Khreishah, "Building a private hpc cloud for compute and data-intensive applications," *International Journal on Cloud Computing: Services and Architecture (IJCCSA)*, vol. 3, April 2013.

[5] H. N. Palit, X. Li, S. Lu, L. C. Larsen, and J. A. Setia, "Evaluating hardware-assisted virtualization for deploying hpc-as-a-service," in *Proceedings of the 7th International Workshop on Virtualization Technologies in Distributed Computing*, VTDC '13, (New York, NY, USA), pp. 11–20, ACM, 2013.

[6] A. G. Carlyle, S. L. Harrell, and P. M. Smith, "Cost-effective hpc: The community or the cloud?," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pp. 169–176, Nov 2010.

[7] "Official OpenStack Mitaka manual." http://docs.openstack.org/. Accessed: 2016-05-10.

[8] "CloudStack." https://cloudstack.apache.org/. Accessed: 2016-05-10.

[9] "Eucalyptus." http://docs.hpcloud.com/eucalyptus/4.2.2/#install-guide/intro.html. Accessed: 2016-05-10.

[10] "OpenStack hypervisors." http://docs.openstack.org/mitaka/config-reference/compute/hypervisors.html. Accessed: 2016-05-10.

[11] "Xen And Xen Server." https://wiki.openstack.org/wiki/XenServer/XenAndXenServer. Accessed: 2016-05-10.

[12] "XCP Overview." http://wiki.xen.org/wiki/XCP_Overview. Accessed: 2016-05-10.

[13] "Ubuntu Cloud page." http://www.ubuntu.com/cloud. Accessed: 2016-05-12.

[14] "OpenStack Dev Ops Manual." http://docs.openstack.org/openstack-ops/openstack-ops-manual.pdf. Accessed: 2016-05-10.

[15] "OSI Layer description by Microsoft." https://support.microsoft.com/en-us/kb/103884. Accessed: 2016-05-10.

[16] "Mitaka Ubuntu Overview." http://docs.openstack.org/mitaka/install-guide-ubuntu/overview.html. Accessed: 2016-05-10.

[17] "Author's github." https://github.com/crypotex/bscthesis. Accessed: 2016-05-12.

[18] "hdparm manual." https://www.cl.cam.ac.uk/cgi-bin/manpage?8+hdparm. Accessed: 2016-05-12.

[19] A. Martoja, "Fault-tolerant networking using linux based systems and consisting of used hardware." Bachelor thesis, 2016. Working title.

[20] D. Tšumak, "Large-Scale Provisioning and Configuration Management." Bachelor thesis, 2016.

Non-exclusive licence to reproduce thesis and make thesis public

I, Andre Tättar (date of birth: 13th of July 1993),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

1.1 reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2 make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

Building and Configuring a Custom Private Cloud Using Consumer Hardware supervised by Artjom Lind

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 12.05.2016