Ville Kontinen

OPENNEBULA-BASED IAAS ENVIRONMENT

Creating a sand-box for educational purposes

Bachelor's Thesis
Information Technology

November 2013

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

Date of the bachelor's thesis

29.11.2013

Author(s)

Ville Kontinen

Degree programme and option

Information Technology

Name of the bachelor's thesis

OpenNebula-based IaaS Environment

Abstract

The purpose of this thesis was to create an educational environment in the field of virtualization and cloud computing. The need for this comes from the growth of the mentioned technologies in the modern computing world, and therefore the subjects must be taught to the future engineers. The ultimate goal was to provide a sand-box environment for the usage of virtualization and cloud computing courses taught in Mikkeli University of Applied Sciences. The environment fits into the infrastructure as a service model.

The project was carried out by first examining different technical alternatives in virtualization and combining the cloud. Appropriate tools were selected from these, and then applied in practice to a test server dedicated for this purpose. At certain stages the initial solutions had to be abandoned and alternative methods were taken into use, keeping the original goal in sight.

In the end the environment was created successfully. The system was tested by creating 25 virtual machines generated from a pre-installed image. During the deployment of the virtual machines the total execution time and the responsiveness of the system was constantly monitored, and the environment behaviour was at an acceptable level.

As further research the storage performance of the system could be improved, along with the development of the environment to a complex cloud environment. These improvements could include features such as external monitoring, additional host nodes and automated error recovery.

Subject headings, (keywords)

Linux, virtualization, cloud computing, IaaS, OpenNebula

Pages	Language	URN
37 p. + 5 p. appendices	English	URN:NBN:fi:amk-2013113019167

Remarks, notes on appendices

Tutor	Employer of the bachelor's thesis
Matti Juutilainen	Mikkeli University of Applied Sciences

KUVAILULEHTI

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

Opinnäytetyön päivämäärä

29.11.2013

Tekijä(t)

Ville Kontinen

Koulutusohjelma ja suuntautuminen

Information Technology

Nimeke

OpenNebula-based IaaS Environment

Tiivistelmä

Tämän opinnäytetyön tarkoituksena oli luoda opiskeluympäristö virtualisointia ja pilvilaskentaa varten. Tarve näiden tekniikoiden opiskelulle tulee tietotekniikan viimeaikaisesta kehittymisestä, ja näitä aiheita täytyy opettaa tuleville tietotekniikan insinööreille. Päämääränä oli luoda hiekkalaatikkomainen testiympäristö virtualisoinnin ja pilvilaskennan kursseille Mikkelin Ammattikorkeakoulussa. Tämä ympäristö tunnetaan malliltaan nimellä Infrastructure as a Service, eli infrastruktuuri palveluna.

Projekti toteutettiin tutkimalla ensin erilaisia teknisiä mahdollisuuksia virtualisoinnin ja pilvilaskennan saralla. Näistä valittiin käytettävät työkalut, jonka jälkeen työ suoritettiin käytännössä testipalvelimelle, joka oli varattu tätä tarkoitusta varten. Eräissä työn vaiheissa alkuperäistä suunnitelmaa ei voitu toteuttaa ja vaihtoehtoiset ratkaisut otettiin käyttöön, pitäen kuitenkin projektin alkuperäinen tarkoitusperä mielessä.

Loppujen lopuksi ympäristö luotiin onnistuneesti. Järjestelmä testattiin luomalla 25 virtuaalikonetta etukäteen valmistellusta virtuaalikonepohjasta. Virtuaalikoneiden luomisen aikana järjestelmän kokonaissuoritusaikaa ja käytettävyyttä seurattiin jatkuvasti, ja järjestelmä suoriutui testauksesta hyväksyttävästi.

Jatkoprojekteiksi sopisivat järjestelmän tallennuskapasiteetin suorituskyvyn parantaminen ja järjestelmän muokkaaminen monimutkaisemmaksi pilviympäristöksi. Jälkimmäisen toteuttamiseen voisi käyttää muun muassa ulkoista valvontaa, lisälaskentaresursseja sekä automatisoitua virhetilan korjausta.

Asiasanat (avainsanat)

Linux, virtualisointi, pilvilaskenta, IaaS, OpenNebula

Sivumäärä	Kieli	URN
37 s. + 5 sivua liitteitä	englanti	URN:NBN:fi:amk-2013113019167

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi	Opinnäytetyön toimeksiantaja
Matti Juutilainen	Mikkelin Ammattikorkeakoulu

CONTENTS

1 INTRODUCTION

In the past years the way of providing services for the end-users has changed drastically. The rapid growth of the Internet and the devices used for accessing it has together set new requirements for the service providers: scalability and high availability. Virtualization has developed to answer to these needs, and therefore it must be taught for the future engineers working in the field. In this particular field pure theoretical learning is not enough, as the technology evolves rapidly and the used tools generally have to be learned via personal experiments. This presents the teachers and the existing educational devices a problem: how to provide the students a hands-on experience, when most of the actual computing hardware used in classrooms differs significantly from the devices used in the industry? In practice this manifests as incompatible software, incapability of performing the required tasks and loss of progress in the students' learning.

The goal of this thesis is to answer those needs via providing a virtualization sand-box for educational purposes to suit the needs of the courses taught at Mikkeli University of Applied Sciences. In the end, the result will be used for practical testing and learning in the field of virtualization. The server should meet the performance requirements needed for serving approximately 20 students without significantly hindering their work.

The project advances by first researching and testing out possible performance boosting tools, compiling the actual hardware, installing and tweaking the operating system, installing the virtualization software and the managing tools for it and finally testing the actual performance of the system. If at any stage severe drawbacks or faults are found, an alternative solution is actively sought for and applied, keeping the selected methods suitable for reaching the original goals.

2 VIRTUALIZATION AND CLOUD COMPUTING

The National Institute of Standards and Technology has defined virtualization as follows: “Virtualization is a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, quality of service, and many others.” [1]

The virtualized resources are joined together, rendering them into heterogeneous pools. When viewed externally, it is common that none of the underlying features are seen due to security purposes. This coins the term cloud computing. As virtualization and cloud computing form together a very large entity, this paper focuses only to the key aspects related to the project at hand.

2.1 Virtualization

Virtualization in the modern form has been developed in the 21st century. Earlier, services were handled in the client-server model, where a single server provided multiple end users with the required services. Approximately at the turn of the millennium the hardware used in the server environments was capable of exceeding the performance requirements set by the software. This caused massive server farms to operate under very light loads, generating additional costs via the purchase and upkeep of the devices. The goal was set to be able to operate individual, virtualized operating systems within a single hardware unit, and multiple corporations and communities sought for the answer. [2]

2.1.1 Virtualizing the CPU

The greatest obstacle in achieving this goal was to overcome the privilege ring rule set used in the x86 processor architecture. The x86 architecture utilizes four rings, numbered from 0 to 3, ring 0 being the closest to the hardware and with the greatest command rights over it. Generally the operating system kernel is installed at the ring 0, device drivers to rings 1-2 and the software to the ring 3. Because the operating systems were designed around this setup, virtualization was not possible due the privilege error: the operating system to be virtualized needed to have a way to access

the resources of the system without disturbing the other operating systems present. [2, 17]

As possible solutions different levels of virtualization were developed:

- Application virtualization: the needed software is run on a computer in a pre-set environment which differs from the actual system configuration.
- Operating system (OS) virtualization: a single computer is installed first with a hypervisor (and a possible operating system), and then the hypervisor emulates virtualized environments for other operating systems to use. [2]

Concept logic of these models can be seen from Figure 1.

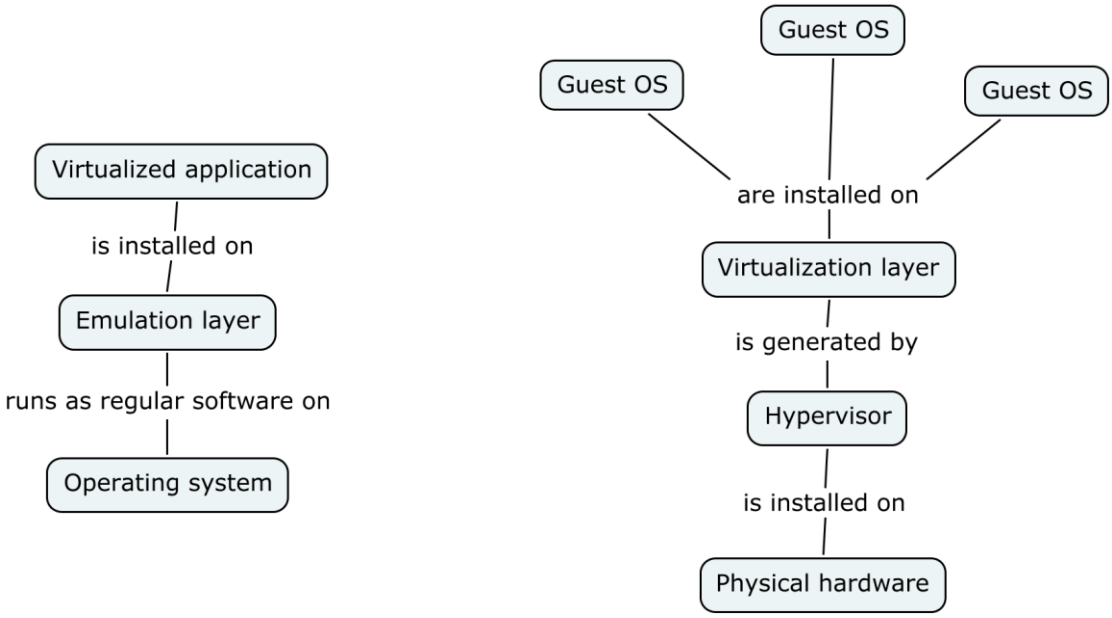

FIGURE 1. Virtualization models

In Figure 1, application virtualization logic is seen on the left side and OS virtualization on the right. The virtualization layer generated by the hypervisor is configured based on which guest operating system will be installed on that environment. [2]

Three different approaches have been developed for achieving operating system virtualization in x86 environments: full virtualization using binary translation, OS

assisted virtualization (also known as paravirtualization) and hardware assisted virtualization. [3]

In a binary translated virtualization the guest OS is not allowed direct access on the x86 ring 0, but it is positioned on the ring 1. All the requests coming from the OS are translated run-time for the hardware, which the guest OS is not aware of. This creates some overhead in the guest OS performance. All the user applications have direct access to the hardware, meaning they suffer from less overhead caused by the virtualization. None of the software installed in this matter requires additional configuration or modification because of the virtualization and the environment sets no restrictions on which operating systems can be virtualized. [3]

Paravirtualization means that the guest operating system has been modified to partially understand the ongoing virtualization and support the environment it has been installed into. In certain cases this setup performs the best, but it rules out all closed-source operating systems due to its nature. [3]

Hardware assisted virtualization is the latest development in this field, as it has been available since circa 2006. The processor has been pre-programmed by the manufacturer to contain an instruction set suitable for virtualization; a root privilege level has been added between ring 0 and the hardware. This allows all types of guest operating systems to be installed as long as the hardware and selected hypervisor support the feature. The hypervisor is installed to the root ring, and therefore it supersedes the rights given for any operating system installed at ring 0. This feature does not affect any existing operating systems, as it is fully backwards compatible. [3]

2.1.2 Virtualizing memory, devices and I/O

A computer does not run solely on a processor, so therefore the remaining components had to be virtualized to achieve a full environment. Modern x86 CPUs have already contained memory management units (MMU) in order for the basic operating system to access the RAM as efficiently as possible. For achieving a virtualized environment, the MMU is virtualized so that the guest operating systems access the hardware memory through two sets of management units: first through the virtualized set and

then by the hypervisor to the actual hardware. The logical behavior of the memory management can be seen in Figure 2. [3]

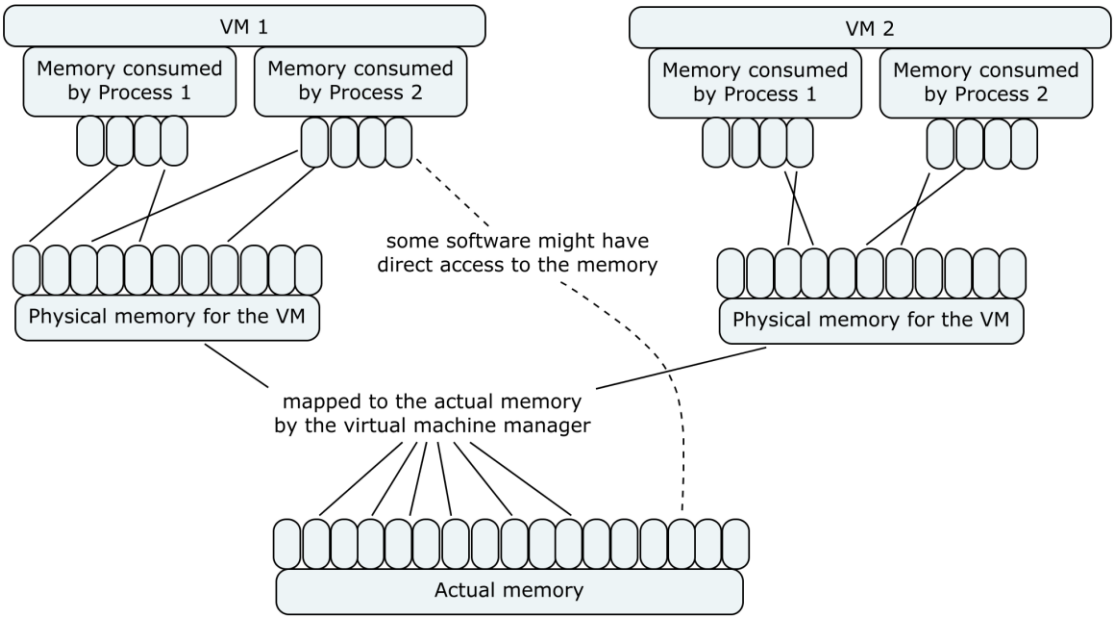

FIGURE 2. Memory management under a hypervisor

The last components to be virtualized are any externally connected devices, such as hard drives, user I/O devices and network interface cards (NICs). The hypervisor provides the guest operating systems with emulated devices. The emulated devices are selected from well known and supported devices to achieve standardization between different hypervisors to allow guest OS portability. [3]

2.1.3 Nested virtualization

A central feature required by the system at hand is called nested virtualization. This concept utilizes multiple hypervisors installed inside a single server. This model can be utilized in multiple different solutions in the IaaS field, such as providing the end-users the capability to further virtualize the environment provided to them and to enable live migration of any hypervisor. [19]

In practice this can be achieved in two different methods in x86 architecture systems, depending on the hardware utilized in the particular system: either via multi-level or through single-level architectural support for nested virtualization. In the former model the hypervisor closer to the hardware handles the requests originating from the

hypervisors installed on top of it in the stack. This causes overhead in the system, as the logical ladder is descended step by step towards the hardware. The logic is presented in Figure 3. [19]

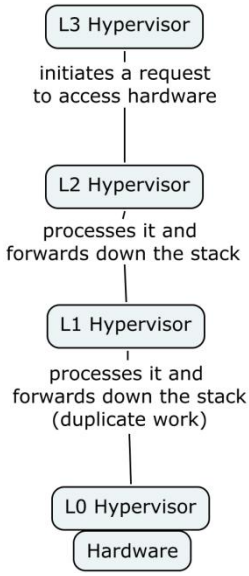

FIGURE 3. Multi-level architectural support operation logic

In the single-level architecture any hypervisor based requests are dropped to the lowest hypervisor in the stack. This requires hardware support from the system, and currently both AMD and Intel support the feature. The logical layout of the installed hypervisors is multiplexed in this model. The multiplexing logic and the order of requests are shown in Figure 4. [19]

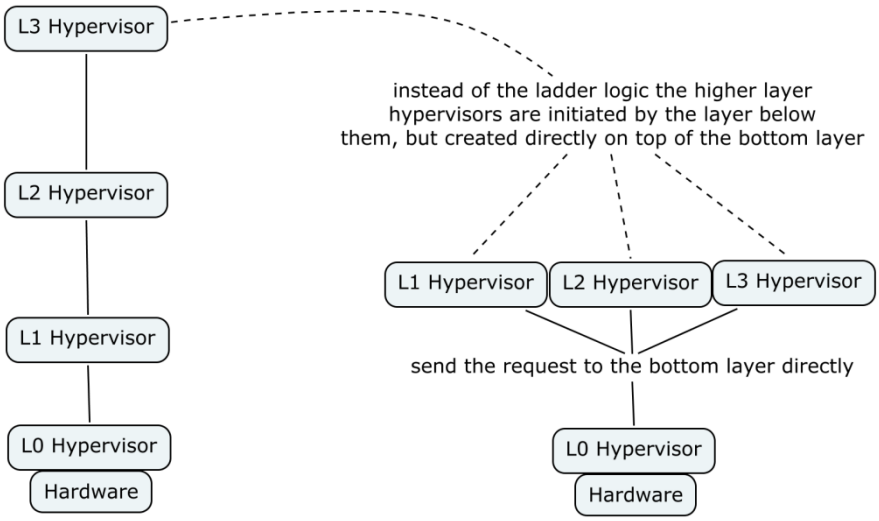

FIGURE 4. Single-level architectural support operation logic

Nested virtualization is present in at least VMware products and Kernel-based Virtual Machine (KVM). In the latter it has to be separately enabled via loading the CPU architecture specific module to the system and configuring the virtual machines on top of the bottom hypervisor accordingly.

2.2 Cloud computing models

The cloud ideology has been used for describing e.g. networking models, such as the Internet, since the development in computing reached too complex models to be drawn in technical documents. The cloud represents a setup created for the desired purpose, which consists of multiple devices and services, but appears as a unified resource to the user. Modern day need for creating cloud computing as we know it aroused from the rapid development and spread of the Internet. In the past 10 years the amount of users accessing the Internet has almost tripled, which has caused the services provided via the network to be extremely agile in the terms of availability and load handling. [5]

In practice these needs are met via cloud computing, which is understood these days as distributed computing resources working together over a network. Virtualization plays a key role in cloud computing, as it allows the installation of multiple lightweight services into a single physical device in a secure manner. [4, 5]

Cloud computing is divided into three major service models:

- Software as a Service (SaaS): The end users have access to a virtualized application, which is provided simultaneously to multiple users originating from different locations and source devices. The end users have no power over the architecture and tools used for generating the service, and limited capabilities on modifying the service behavior. The main goal using SaaS is to provide similar services as a traditional desktop program does.
- Platform as a Service (PaaS): The end users are provided with a pre-set environment with hidden underlying characteristics. The users only have control over their own components in the system. In this scenario the goal is for the users to utilize their own programming work (applications), much like in a regular operating system environment.

- Infrastructure as a Service (IaaS): In this scenario the end users are provided with real computing resources. The users have free choice how to utilize them, such as run different operating systems and applications. The users have no control or knowledge how the resources are gathered, merely how much is at their disposal and what they choose to do with them. [6]

The categories can be seen as a pyramid-like structure, as seen in Figure 5, starting the list from the top of the figure. In this thesis IaaS plays the key role, as it is a model which meets the requirements for providing the educational sessions with the needed computing power and compatibility.

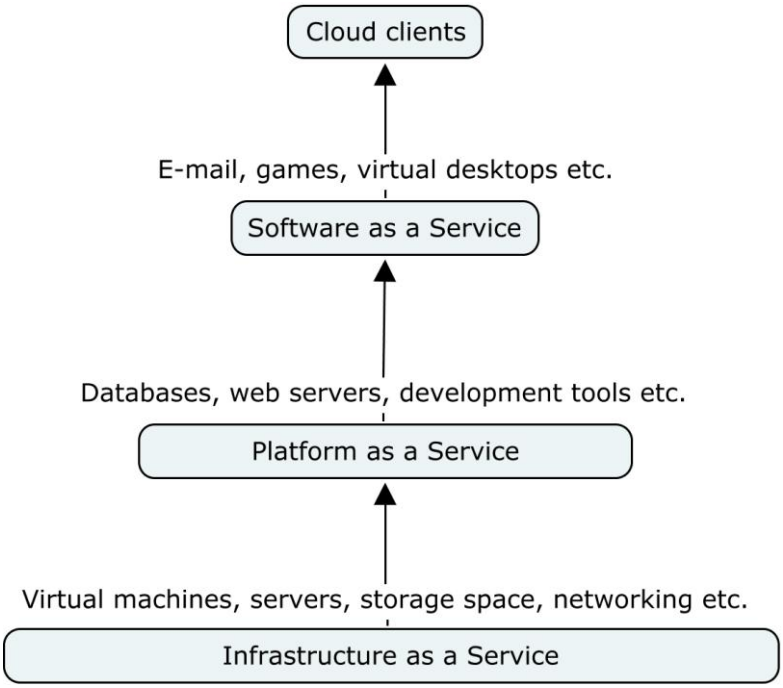

FIGURE 5. Cloud model stack

Generally any deployed cloud can be recognized from the following characteristics:

- Resource provisioning on-demand: the cloud does not have to operate at its full capacity at all times – the resources are brought available when needed.
- In enterprise solutions paying per usage: the cost of the service is related to the actual usage level of it, instead of providing a service with a fixed sum even when it is heavily underutilized.

- Quality guarantees: quality of service (QOS) is guaranteed by the provider – the cloud is available and capable to perform as agreed on.
- Scalability: if needed, the cloud can be scaled up or down in the provider side with ease to suit the current needs. [18]

Infrastructure as a Service is generally conceived as the most complex service model to operate from the three service models – this is mainly due to the fact that it can contain the other models and is capable to suit their needs, not vice versa. Setting up an environment based on the model requires knowledge from the computer hardware operating principles, operating systems, security aspects, both internal software threats and external, malicious user threats; and complex networking, as the end result should seamlessly provide the user resources similar to a regular server, only in larger capacity. Achieving this generally requires in-depth understanding of the underlying technology. [7, 18]

Any cloud can be deployed based on the following models:

- Private cloud: the cloud is accessible only to the members of a single organization. It can be managed by the organization solely, a third party or both of them in collaboration. It may exist in- or outside the organization premises.
- Community cloud: the cloud is accessible for a joint community of organizations. With this distinction it bears the same characteristics as a private cloud.
- Public cloud: the cloud is accessible to the general public. It may be operated by any party, and resides in the provider's premises.
- Hybrid cloud: the cloud is a mixture composed from any of the aforementioned models, which remain unique but are bound together for achieving capacity or capability benefits when needed. [1, 7]

The cloud deployment models and their interactivity can be seen from Figure 6.

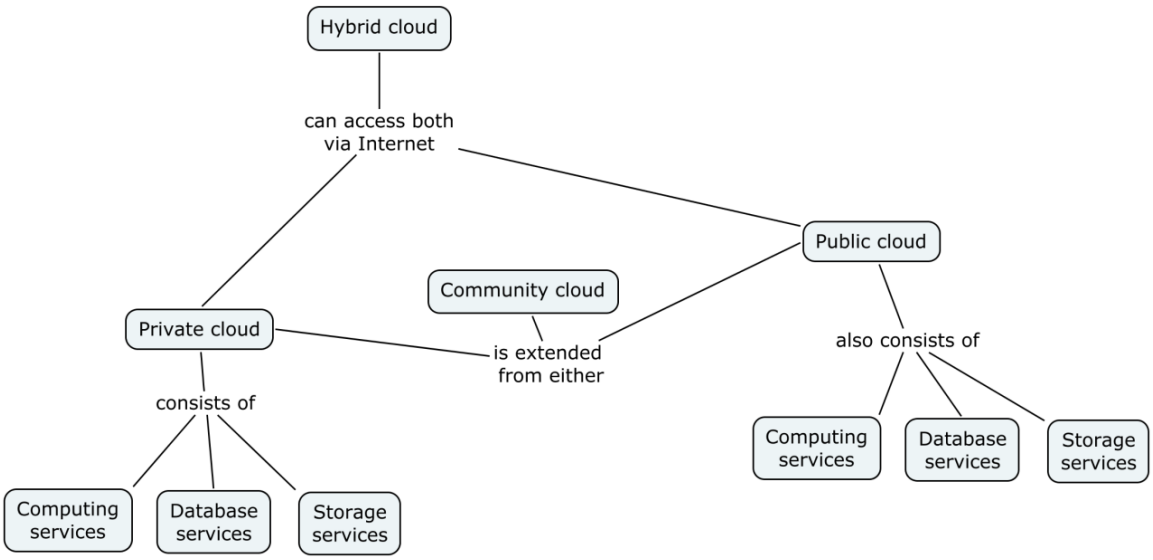

FIGURE 6. Cloud deployment models

It is worth noting that a community cloud can also be private in the sense of who is it targeted to, and can be considered also as either private or public cloud. The main difference in the cloud models is the availability: whether the services are accessible completely, partially or fully by the general public. [7]

2.3 Cloud deployment options

Multiple major players and joint open source collaborations in the information technology field have developed their own techniques on how to virtualize resources. In this thesis the focus lies on different methods used for virtualizing the hardware of a computer, and then used for generating an IaaS-type cloud. The software used directly on top of the hardware is a hypervisor. The currently notable hypervisors are

- VMware ESXi (these days labeled as vSphere), created and maintained by a pioneer in the virtualization field, VMware.
- Xen, an open-source project and the foundation for many other projects, both commercial and open source.
- KVM, a virtualization subsystem included in the Linux kernel since version 2.6.20. [8]
- Microsoft has developed its own known as Hyper-V. Hyper-V is utilized in Windows Server environments, and is included in the operating system since Windows Server 2008. [9]

Virtualization as itself could be achieved via using the aforementioned hypervisors without any additions, but the hypervisors lack the capability to join resources together to form an effective IaaS cloud. Different hypervisor producers have created their own toolkits for controlling multiple separate physical servers containing the hypervisor at once. These tools provide the server farm the true nature of cloud computing: automated service, pooled resources, on-demand provisioning of resources and automated resource utilization monitoring. [6]

As professional tool examples, VMware's vCenter and Microsoft's System Center Virtual Machine Manager (SCVMM) are capable of managing hundreds of host nodes together as a cloud and utilize their resources to the full extent. The vCenter is an independent server application installed preferably on a standalone server, and the host nodes communicate via a network to form a scalable, fault tolerant cloud. The SCVMM is installed on top of a Windows Server and is then used to configure host nodes in the network. An open-source alternative capable of performing the same tasks is OpenNebula. OpenNebula is operated by default via a command line interface (CLI), but it can be installed alongside with Sunstone, which provides a graphical user interface (GUI) for the system. Essentially both of these toolkits, vCenter and OpenNebula, provide means of configuring and monitoring multiple hosting nodes simultaneously. It is worth noting that both are tied to Linux environments: vCenter is, in the end, a very specifically designed Linux distribution flavor and OpenNebula a downloadable toolkit for certain Linux distributions (namely Debian, openSUSE, Ubuntu and CentOS). [10, 11, 12]

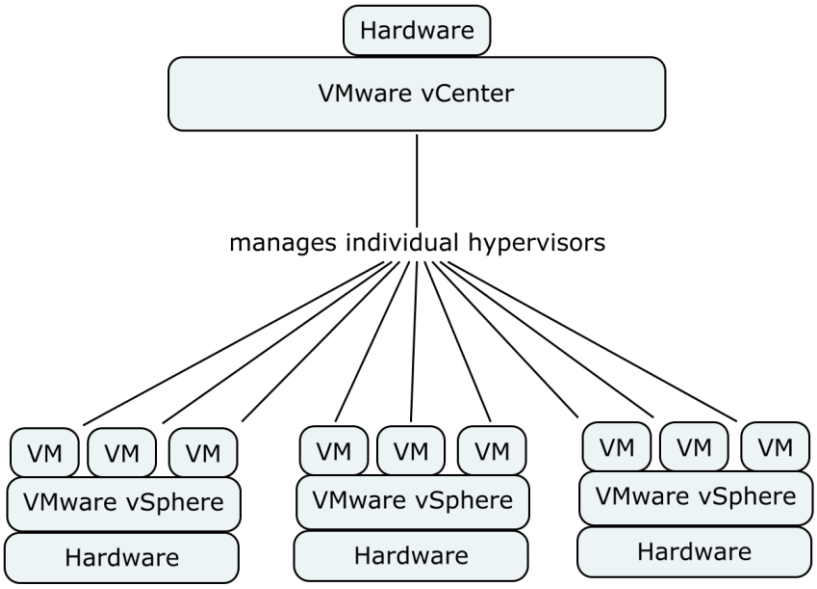

FIGURE 7. VMware vCenter logical diagram

In Figure 7 the logic behind the centralized toolkits is shown. Multiple host nodes are collected together into the management software, so that the resources are available as a full scale cloud. OpenNebula and SCVMM operate in the same logic.

3 DESIGNING A CLOUD ENVIRONMENT

In this chapter the general key points in designing a cloud environment are briefly mentioned and the selected hard- and software tools are presented. One must keep in mind the needs of the system at hand when choosing any devices to be attached to it, because the hardware may either permit or forbid essential operations in the finalized environment.

3.1 Key consideration points

Cloud computing sets certain minimum limits for the performance of the cloud. Virtual machines (VMs) can theoretically consume as much processing power as a regular server or PC. When multiple VMs are installed onto a cloud platform, the hardware components need to match up to the need at hand.

In the following are the key components in the hardware along with factors that matter to the performance of the system in overall:

- Central processing unit (CPU): the servers hosting the VMs must have support for virtualization. This is the only aspect required from them, but logically the performance of the CPU(s) affects greatly the overall performance of the system.
- Random access memory (RAM): bearing in mind that the server will host multiple VMs and each of them have the same capabilities of traditional servers, the total amount and type of RAM has to meet the minimum limits for the desired operations. Preferably the total amount of RAM installed should exceed that limit to avoid possible performance sufferings.
- Long term storage: all the data must be stored in some type of an environment, whether it is direct storage or network file system (NFS). Redundant Array of Independent Disks (RAID) is an essential technology here, no matter how the array is connected to the server itself. RAID combines multiple disks as one logical storage unit with potential performance and failure advantages. The storage is generally conceived as the first possible bottleneck in virtualized environments, so the read and write performance of the array has to be planned properly. The developments in solid state disks (SSD) durability have added them to possible components in the system alongside the traditional hard drives (HDD).
- Networking: as the servers connected to the cloud communicate over a network, the individual units and the networking devices connecting them must have sufficient networking capabilities. This is especially true when a NFS of any type is used. [11]

In addition to this, software can cause limitations to what can be achieved in the system. As a few points worth mentioning, file and operating system architectures may cause unwanted effects in daily operations. Different file systems are designed for different purposes and the technical advancements during that time affect the characteristics of them: e.g. maximum file system size, maximum and minimum file size, capabilities to recover from sudden power losses and stored metadata (such as access rights). Operating system architecture affects the performance of the system via its capabilities such as resource allocation (both CPU and RAM) and multi-tasking.

As mentioned earlier, storage can cause performance penalties on larger deployments. Additionally storage modules have to be designed to avoid a single point of failure. Several different techniques have been developed to overcome these issues, and just to name a few:

- Shared file systems, which allow clustered servers to access data over a network.
- Storage area network (SAN) which is created in a local area network (LAN), allowing storage units to be directly connected to the network.
- Two storage related networking standards:
    - Fibre Channel (FC) standard, which allows communication in extremely high speeds, generally through optic fiber network.
    - iSCSI, which loses in performance to FC but can operate through almost any transmission medium. [15, 16]

The general logic in larger deployments is to have one or multiple storage modules present in the whole cloud. The connecting links to the module have to be high-speed and aggregated, and in critical cases the module has to be duplicated to another physical location to prevent data loss in disaster events. On top of this, each node accessing the storage can have some type of local storage to use as a cache before serving the client and to store the OS on. [15]

The environment created in this project is not complex enough to require a full sized SAN, but the performance of the storage needs still attention. Therefore the system should utilize some sort of a method in order to boost the performance of the disk array. Given the hardware available for usage, the system could be built with a fast disk drive installed as a cache between the large storage unit and the RAM. There are at least two solutions available for this: bcache and flashcache. Both utilize preferably SSDs as the cache before committing the actual input/output (I/O) operations to the storage array. [13]

As an alternative approach for achieving better results from HDDs the journal of the file system can be moved to a faster drive. A journaling file system creates a journal consisting of all the writes performed to the file system and updates it constantly during usage of the system. In large disk partitions the size of the journal grows to be

quite large, and accessing for detecting unallocated disk space delays the disk operations. Certain file systems allow the movement of the journal away from the partition it is written about, permitting it to reside on a faster disk for better performance. The file system chosen to be used in this environment was ext4, so this alternative is a viable possibility. As a performance example, the external journal generated from a ext4 file system can theoretically boost the writing speeds up to three times when compared to a regular ext4 file system. [14]

In this project the main tasks to be performed were generic implementation level jobs done on a fresh system. These include the installation and initial configuration of operating systems, performing basic to medium level configurations and testing other systems in nested virtualization. The simultaneous user load was estimated to be approximately 20-25 students. The majority of the tasks performed on the system would therefore relate to I/O operations in the storage. Networking and pure CPU performance are not considered essential in this project, since the environment would not be used to host any long term services besides the mentioned.

3.2 Selected setup

The selected server hardware layout is the following:

- Motherboard: Asus Z9PE-D8 WS
- CPU: 2 x Intel(R) Xeon(R) CPU E5-2620 @ 2.00GHz
- RAM: 4 x DDR3-DIMM Kingston 16GB @ 1333MHz
- Storage: 6 x Western Digital SATA3.0 3 TB HDD, OCZ RevoDrive3 PCI-E 240 GB SSD
- Networking: 2 x integrated Intel® 82574L Gigabit LAN Controller.

The operating system was chosen to be Ubuntu Server 12.04.3 LTS, due to the open source nature and sufficient support for meeting the set criteria of the project. This selection was made mainly based on the extensive support available for the system and general compatibility with the rest of the chosen software. A viable alternative would have been to use Debian, which Ubuntu is built on.

OpenNebula toolkit was chosen to be used for controlling the resources to form a cloud. An alternative open-source option for OpenNebula could have been Xen Cloud Platform, which is supported by both of the considered operating systems. OpenNebula was selected because of its flexible modularity.

The ultimate goal is to allow students to experiment in any possible way, and the software should not limit the different exercises and tasks the system is capable of performing. One key aspect in relation to this is nested virtualization. This requirement roots to the fact that the whole environment is already virtualized when students enter it; if nested virtualization is not available, the students cannot perform any tasks related to virtualization since the tools are not present. The base system cannot be allowed to be used by students, since the possibility of a critical system failure induced by them is very present, and would effectively ruin the whole learning environment.

It was decided that OpenNebula would be installed with Sunstone, a graphical user interface (GUI) for the ease of use. VMware products were considered as a real possibility, but as the products are essentially for enterprise usage, the thought was abandoned. The main reason for not considering Hyper-V from the Microsoft family for this project was the lack of support for nested virtualization. [11]

The selected motherboard supports different RAID levels, but not all of them are supported in Linux environments. This ruled out the usage of this feature, and software RAID generated by the OS was selected. Performance-wise an independent RAID controller would have been a secure choice, but budget restrictions ruled this option out. From the tools mentioned in the previous chapter bcache was selected to be the primary option for boosting the system and as an alternative, the utilization of an external journal. [13]

The system itself would be installed as a standalone unit, with the OpenNebula front-end (the managing toolkit) existing on the same hardware as the hypervisor and the VMs. While this does not match the true logic of a cloud, the door is left open for extending the cloud as OpenNebula has the capability of accepting heterogeneous resources. This could mean combining external student projects based on different tools to the existing cloud.

4 OPENNEBULA

The roots of OpenNebula trace back to a research project based in Madrid, started in 2005. The project was released as open-source in 2008, and is now open for development via a community. OpenNebula has been developed from the start in a fully open ideology: it has been created around the ideas of interoperability and compatibility between all possible components. Because of this, OpenNebula does not utilize a single hypervisor and has no specific requirements for the environment it is installed to. [11]

Additionally OpenNebula differs from the other open-source cloud toolkits due its usage of XML remote procedure calls (RPCs), which can be used to access the application programming interface directly. OpenNebula also utilizes Cloud Computing Interface (OCCI) and support to Amazon Elastic Cloud Compute (EC2) in order to expand the resources connected to it in order to form a hybrid cloud. [11, 20]

As stated, OpenNebula as itself is installed as a front-end and offers no virtualization features. The front-end manages the resources connected to it, and it utilizes secure shell (SSH) access in order to communicate to the attached servers. While theoretically this allows geographically very distributed systems, the method it is utilized in provides a challenge from the security viewpoint: the front-end requires SSH access without passwords to the host nodes. If the cloud is deployed over public networks, but is intended as a private cloud, this has to be noted by the network administrators. [20]

Another point worth considering because of the operational logic in the environment is the possibility of a human error. As OpenNebula allows all configurations to be set in the system, the user has to pay attention to avoid possible configuration mismatches and errors. [20]

As mentioned earlier, OpenNebula provides two ways for users to interact in it: either through the command line interface (CLI) or through the graphical user interface (GUI) named as Sunstone. Sunstone follows the modular design of OpenNebula, and utilizes Javascript-based plugins to access the application programming interface (API) provided by OpenNebula. The plugins are loaded into the Sunstone client via

settings provided in YAML format. The administrator can freely create and modify the plugins based on the needs at hand. [11]

It is worth noting that the output behavior of the CLI commands is customizable, and can be achieved via modifying the configuration files found in /etc/one/cli/. This allows the user to modify the information obtained through the CLI to meet the desired needs. Similar behavior to this can be achieved in Sunstone through modifying the plugins and their settings to alter the output they provide to the client. [11]

OpenNebula uses websockets in order to provide a desktop environment from an installed virtual machine. In practice this is achieved by utilizing virtual network computing (VNC). OpenNebula is installed with the software noVNC, which provides the virtual machine an active VNC server session assigned in a random port. The user can then either connect to the session with any external VNC viewer software or access the built-in VNC viewer by noVNC through the GUI Sunstone. Although the latter operates theoretically, it has certain issues in higher resolution screen sizes (some icons appear with vertical lines and the screen ratio is not correct). Using an external viewer is therefore recommended, if the virtual machine cannot be connected otherwise, e.g. SSH or remote desktop connection. The port for connecting to the VNC session for each virtual machine can be found from the detailed virtual machine information. An important feature of VNC is that it is not a secure protocol in overall and by default OpenNebula VNC sessions are unencrypted.

The operational logic of OpenNebula and associated components is presented in Figure 8.

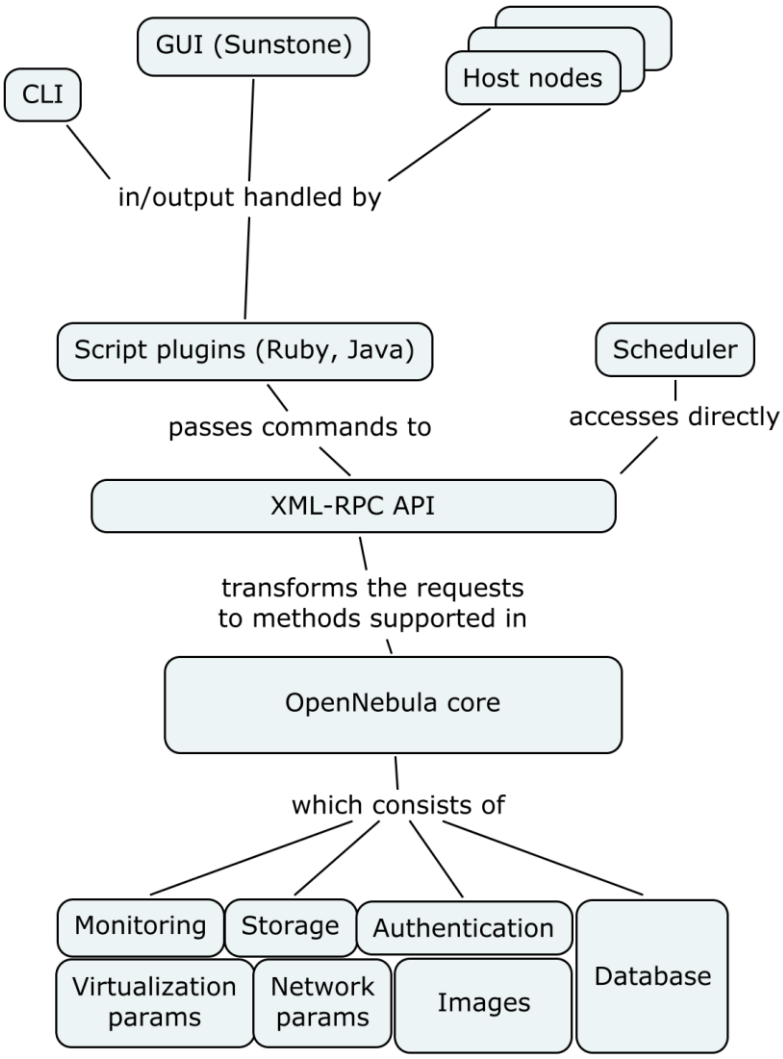

FIGURE 8. Operational logic of OpenNebula

The underlying components listed in Figure 8 provide the core installation with the drivers needed for operating a cloud. As stated before, the software performs no virtualization or VM hosting on its own – it merely utilizes the hardware resources provided by the host nodes. The listed components define how OpenNebula utilizes these resources.

Logically the storage contains all the other elements listed, but it can be expanded and divided depending on the purpose. The majority of the storage is spent on the virtual machine images, both installation disc and actual VM specific images. The database contains all the data and settings related to the software, items such as the usernames and passwords. The virtualization and networking parameters are provided to the specific host nodes and virtual machines when needed, and the networking parameters

are also utilized when performing VM routing. The community has provided OpenNebula with a fully compatible virtual router, which is available as an add-on from the OpenNebula Marketplace. [11]

The monitoring component performs the actual resource state monitoring, and this utility can be configured with hooks. The hooks offer the system user-defined actions based on resource state, e.g. a host node not responding the monitoring poll would trigger an e-mail alert. Finally, the authentication component ensures that all actions performed within the system are correctly authorized via methods such as encrypted passwords, SSH RSA key pairs or Active Directory. [11]

In conclusion, OpenNebula and Sunstone provide the administrator very extensive configuration possibilities. The environment is extremely flexible, and because of this the administrator has to pay attention to the actions performed in the system. [20]

5 INSTALLATION AND CONFIGURATION

The system was installed multiple times using the throw-away prototype model. The logic for this prototyping model comes from coding projects. In some cases the initial design is anticipated to change during the creation process, so the project is handled on a throw-away logic: after an initial draft has been made and the first problems encountered and solved, the whole project is then started again from the absolute beginning. The reason to act in this manner is to avoid unnecessary components ending in the finalized project. In this project the failure of implementing bcache as intended serves as an example: if a component does not operate as intended, the safest way is to fully remove it from the system to avoid any further possible problems originating from it. Because of these obstacles the system had to be reconfigured from the base foundations, and reverting configurations to previous settings was impractical in some situations.

5.1 The operating system

The base system was assembled from the parts without major problems and the operating system was installed with ease. The installation of the OS was performed

according to the guidelines given in the OpenNebula documentation, namely by creating the default user account for “oneadmin” and selecting the OpenSSH package immediately, rather than configuring these into the system after the installation. The OS was installed in English with Finnish localization. During the installation phase, the system was configured to utilize software RAID 5, using all the HDDs as active devices for achieving maximum disk capacity. The total capacity for the RAID is 15 TB.

After several tries the SSD proved to be unusable for storing the operating system itself, as the manufacturer does not provide any support for Linux to this date. A workaround patch has been provided via open-source projects, making the SSD partially usable as additional storage or swap, but Linux distributions are unable to boot on any partitions configured to the SSD. Additionally, the patch caused Linux to understand the 240 GB disk as two separate 120 GB disks. Because of these facts achieving full utilization from the SSD proved impossible. The first recognized partition from SSD disk was configured to contain the boot loader GRUB in order to avoid the absence of the boot loader in case one of the HDDs broke down. The remainder was configured as swap space for the operating system. The second partition was configured to contain a 5 GB spare space, formatted in ext4 file system. The purpose of this partition is explained in the next chapter. The full disk usage scheme can be found from Table 1.

TABLE 1. Disk usage scheme

	Device	Assigned partitions
SATA disks 1-6	/dev/sda-f	3 TB each, used as parts of RAID
RAID 5	/dev/md0	15 TB formatted as ext4 for /
SSD, first partition	/dev/sdg	500 MB formatted as ext4 for GRUB, 119.5GB for swap
SSD, second partition	/dev/sdh	5 GB formatted as ext4 for external journal space

5.2 Boosting the hard disk performance

The next goal was to achieve best possible results from the RAID. A few tools were considered in the planning phase, but due to some unforeseen factors the ultimate solution differs from these.

5.2.1 Bcache

As mentioned before, the original plan was to utilize Bcache. Bcache consisted of an external patch for the Linux kernel and a separate toolkit for managing the behavior of the software. The software was implemented to the Linux kernel in version 3.10 stable. The general logic in the software is to utilize a faster hard drive as a block layer cache in front of a larger, slower storage. It has no restrictions related to the used file system. In practice it the used drives are configured to appear as a Bcache device and the original devices are not accessed directly. The Bcache device appears as a regular disk in the OS under the identity of /dev/bcacheN, and is then available to any desired usage. The drawback in the design is that the used devices must be formatted blank before they are available, which requires the operating system being installed on a separate partition.

During the initial installation time Bcache was installed to the system successfully, utilizing the second partition of the SSD as the caching partition and the majority of the HDDs as the storing device. There were two separate RAID 5s installed, one for the Bcache and one for the OS. The operating system was able to fully work with the software, but for unknown reasons OpenNebula refused to read any VM images or network configurations stored on the cached devices. There were no external error messages, regular data read-write operations on the cache were successful but nevertheless the cache did not operate properly in the environment in the end. Because of this, Bcache was removed from the system and the disk partitions were revised. After these operations OpenNebula started operating as intended.

5.2.2 External file system journal

After Bcache failed to operate properly, this solution was selected as the next best option.

The configuration of the external journal is relatively simple and straightforward: the original journal for the file system is deleted, a dedicated partition for storing the external journal is created and the journal is recreated to that partition. After this the original file system is then formatted based on these settings, and set to be mounted to the system using asynchronous writes.

After modifying the file system in this manner the performance was tested by using sequential reads and writes. Since the OS was installed physically on to the same devices as the designated larger storage space, and software RAID was implemented on both partitions, the ext4 with the external journal did not achieve significantly better performance. After exhaustive testing the benefit on write speeds appeared to hover around 5%. Due to this fact the external journal was abandoned as a solution and the system was reinstalled on only one RAID partition, as seen on Table 1.

5.3 Installing OpenNebula and Sunstone

The installation of OpenNebula and Sunstone are discussed in this chapter. OpenNebula front-end was installed onto the same platform as the host. In the shown commands, rows starting with the number sign require the command to be input as the root user and rows starting with a dollar sign should be input as a regular user. It is assumed here that the operating system is installed with the user “oneadmin” and that there is a SSH server running in the system.

The most important step is to ensure that the CPU used in the system supports hardware assisted virtualization. After this, the system is updated to the latest software versions. If the kernel is updated in this process, which is very likely, the system should be rebooted before processing. The system should then be installed with the desired hypervisor; in this project KVM was installed. During the same installation the Libvirt toolkit, virtual bridge tools and QEMU toolkits should be installed.

```
$ egrep ’(vmx|svm)’ /proc/cpuinfo|wc –l
$ sudo apt-get update && upgrade
$ sudo apt-get install bridge-utils kvm libvirtd qemu-kvm
```

The next step after this is to add the OpenNebula repositories into the system. This is done by first obtaining the repository key from the website, then adding it to the trusted keys. After this the download address specific to the operating system version is added to the repository list. The software can be installed from alternative sources, but the preferred method is to use the repositories provided by the developers.

```
# wget http://opennebula.org/repo/Debian/repo.key
# apt-key add repo.key
# echo "deb http://downloads.opennebula.org/repo/Ubuntu/12.04 stable opennebula" >
/etc/apt/sources.list.d/opennebula.list
```

The OpenNebula and Sunstone are then installed. The system will very likely ask that should the software be installed from the added repository, in which the answer is yes. It is advisable ensure after the installation that both of the daemons are not running to avoid possible configuration mismatches, as the default settings do not provide full connectivity to the setup.

```
$ sudo apt-get update
$ sudo apt-get install opennebula sunstone-server
$ one stop
$ sunstone stop
```

The next step is to utilize a script provided in the installation package, which automatically detects the used OS and installed packages and then installs the rest of the needed items. These include ruby gems, sqlite3 development, mysql client, curl development, libxml2 and libxslt libraries, gcc and g++ and make, if not present.

```
# /usr/share/one/install_gems
```

After this SSH keys are generated and added to the authorized keys to allow passwordless SSH. It is worth testing the SSH connection via trying to connect from the system to itself. If there are any problems achieving this connection, the configuration of the network and the SSH server should be double checked. Then the

user oneadmin is added with a password to the OpenNebula authentication file, and the file is read protected from others.

```
$ ssh-keygen
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ ssh oneadmin@onehost
$ echo "oneadmin:password" > ~/.one/one_auth
$ chmod 600 ~/.one/one_auth
```

Then it is ensured that the user oneadmin is included in the groups kvm and libvirtd. After this the libvirt configurations should be changed to allow live migrations via TCP. This is done by making the listed modifications to the stated configuration files:

```
$ sudo adduser oneadmin libvirtd
$ sudo adduser oneadmin kvm

$ sudo nano /etc/default/libvirt-bin
libvirtd_opts=”-d –l”
$ sudo nano /etc/libvirt/libvirtd.conf
listen_tcp = 1
```

Now the OpenNebula daemon is ready for use, and the initial success of the installation can be verified by checking the list of running virtual machines. If the installation was successful, the monitoring headers appear without any installed guests. If any error messages are printed, the log files for OpenNebula are located under /var/log/one/oned.log and /var/log/one/sched.log. The daemon should always be started with the user oneadmin.

```
$ one start
$ onevm list
```

After this it is time to configure the Sunstone in order to access the GUI via a browser. The configuration file for it is located in /etc/one/sunstone-server.conf. Most of the default configurations are suitable, but a few points need to be noted and/or modified:

- :host – traffic originating from which IPv4 addresses is allowed to access the system, entering 0.0.0.0 will allow any, and
- :port – which port the Sunstone is listening to, defaulting to 9869 and cannot be set into any system reserved ports (such as 80 and 8080). In this environment the port was set to 12000.

The Sunstone GUI is ready to be started, which is done by the command "sunstone-server start". In case of any errors, log outputs are found in /var/log/one/sunstone.log and /var/log/one/sunstone.error.

It is worth noting that after a reboot the daemons start simultaneously, which may cause Sunstone time out the connection to OpenNebula. This appears as the services running normally, but Sunstone reports of being unable to authenticate the log-in. This can be fixed via starting the OpenNebula before Sunstone and waiting several seconds before initiating Sunstone.

5.4 Configuring the environment for usage

This chapter is about how to configure OpenNebula via the different paths (CLI and GUI). All operations provided by OpenNebula can be configured via Sunstone, but it is advisable to learn at least the procedures in both manners. If Sunstone is facing errors of any kind, the CLI can be used to recover from them.

5.4.1 Adding hosts and clusters

Before OpenNebula can be utilized for operating any virtual machines, it must be configured with at least one host node. As the installation logic separates the front-end from the host nodes, the system does not initially have any computing resources at its disposal. If needed, different clusters can be created and hosts added to them to form separated entities in the cloud. The usage of clusters is fully optional, and all the added resources default to cluster "None" and therefore operate together by default.

In the CLI the following syntax is used for adding a host and then checking its state:

```
$ onehost create hostname im_mad vmm_mad tm_mad vnm_mad
$ onehost list
```

In the syntax, the im_mad and vmm_mad stand for which hypervisor toolkit to revert in controlling VM images and the actual VMs (as an example, using KVM would require setting im_kvm and vmm_kvm). The following two parameters are related to image transfer protocols and the virtual network management driver. Values such as tm_ssh for utilizing SSH in image transfer and dummy for the virtual network can be used.

In Sunstone, hosts are added via the menu items Infrastructure -> Hosts and clicking Create. Figure 9 illustrates the host managing interface in the GUI.

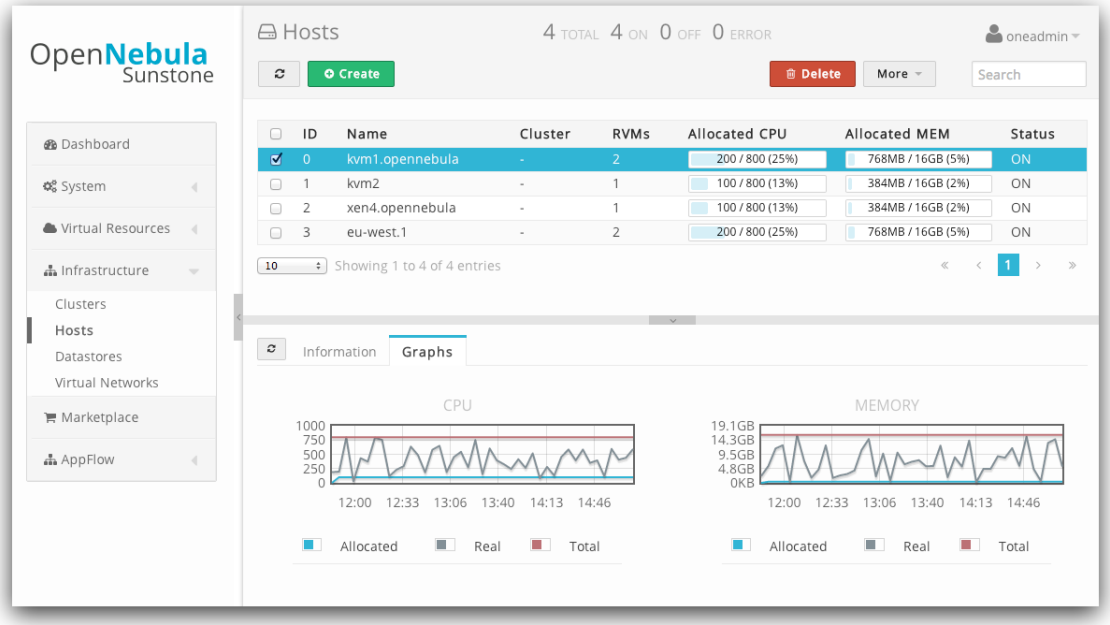

FIGURE 9. Adding and monitoring hosts in Sunstone

In the following example a cluster is added via the following command, then all available clusters are listed, a host is added to the created cluster and the details of the cluster are printed out. One host can be configured into a single cluster only, and should be deleted from it with the command on last line, if needed.

```
$ onecluster create cluster-name
$ onecluster list
$ onecluster addhost cluster-name hostname
$ onecluster show cluster-name
$ onecluster delhost hostname
```

Clusters can be configured to contain any virtual infrastructure resources. These include the mentioned hosts, virtual networks and datastores. In the following are the commands how to add and remove a virtual network and a datastore to and from a cluster:

```
$ onecluster addvnet cluster-name network-name
$ onecluster delvnet cluster-name network-name
$ onecluster adddatastore cluster-name storage-name
$ onecluster deldatastore cluster-name storage-name
```

Clusters are maintained in Sunstone from Infrastructure -> Clusters, Create. Figure 10 shows the GUI in the current version during this step.

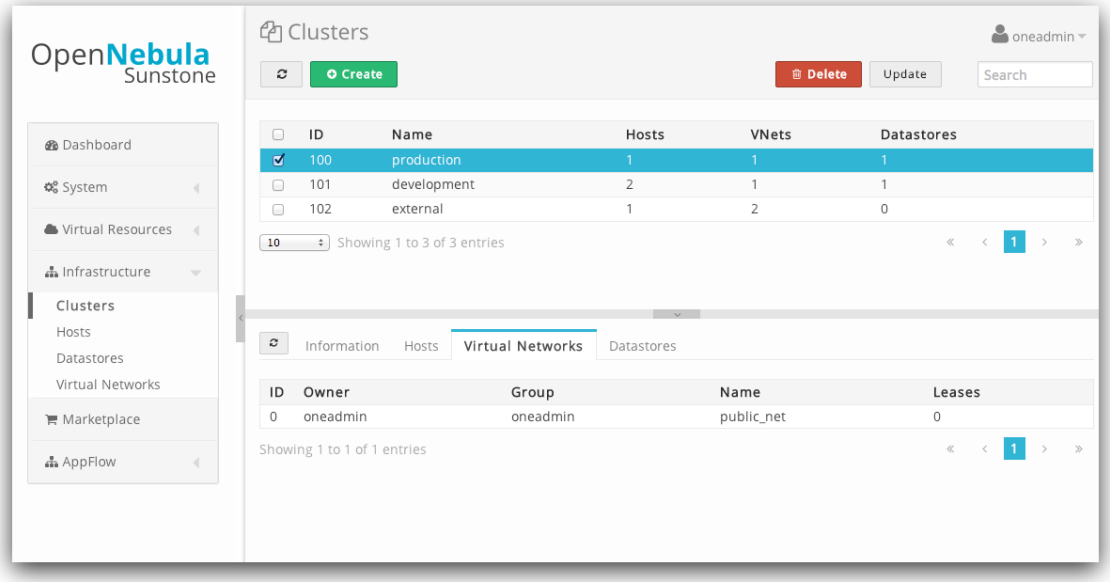

FIGURE 10. Managing clusters in Sunstone

It is worth noting here that the usage of clusters in simple environments is not necessary, and the environment will operate in the default cluster without any

attention paid to the clustering aspect. In case the administrator wants to separate and dedicate resources in more complex environments, the clustering comes to play.

5.4.2 Virtual networks and datastores

The next step is to add a virtual network for the VMs to communicate through. A virtual bridge must be configured using the tools provided in the bridge-utils package, which was added during the installation of OpenNebula. The virtual interfaces are to be configured via those tools, and then OpenNebula networks are added to utilize these networks.

In the command line interface the user must first create a sufficient network layout template to a blank file. Two examples for separated LANs can be found from Appendices 1 and 2. We will assume that the files are saved as blue-lan and red-lan. After the creation of those files the networks can be initiated with the following commands:

```
$ onevnet create blue-lan
$ onevnet create red-lan
$ onevnet list
```

It is worth noting that the creation of virtual LANs is faster via the GUI, as it requires no externally created templates. The networks are added from Infrastructure -> Virtual Networks, Create. The configuration window shown in Figure 11 opens up, and the network parameters are set there.

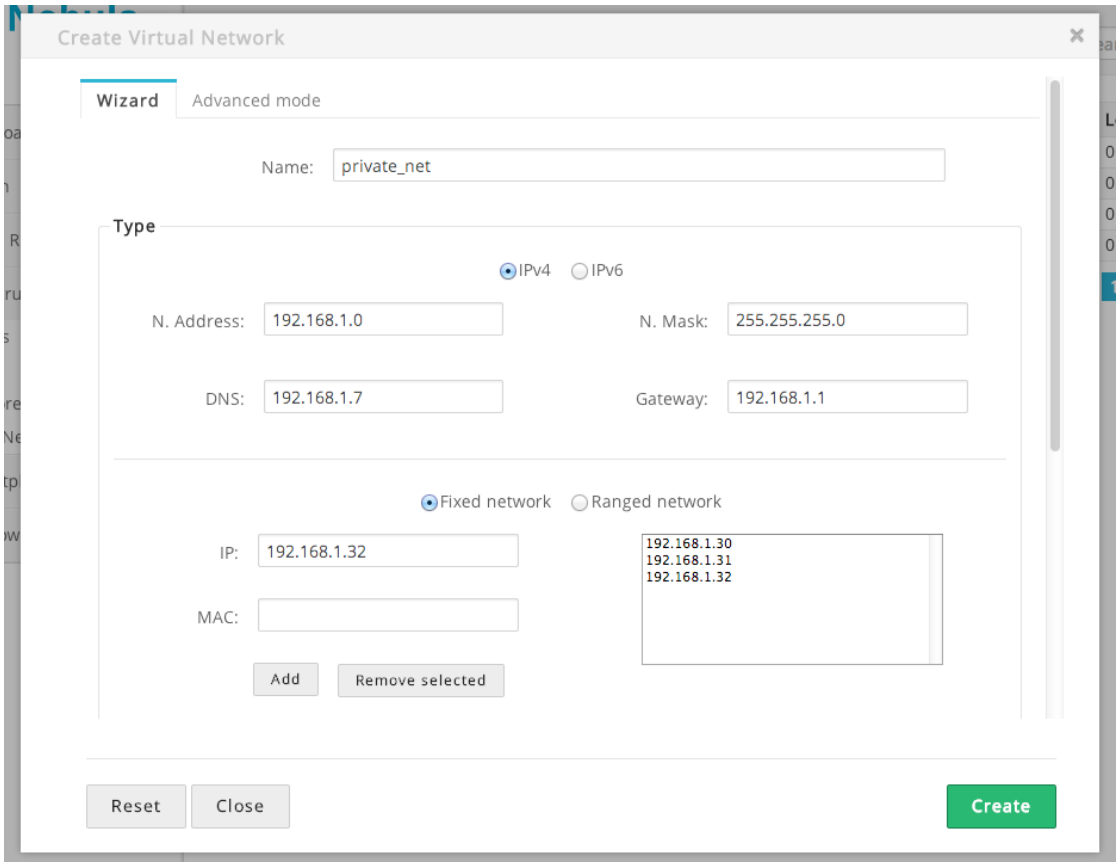

FIGURE 11. Adding virtual LANs via Sunstone

Datastores are used for storing the virtual machine images and other files (such as kernels and configuration files) inside the cloud. By default the OpenNebula installation utilizes the disk space allocated on the front-end in /var/lib/one as the primary datastore. Any additional datastores can be added and configured in very complex ways, and it is advisable to consult the software manuals for these settings. The current datastores can be viewed with the CLI command “onedatastore list”, or through the GUI under Infrastructure -> Datastores.

5.4.3 Virtual machines, templates and images

OpenNebula creates virtual machines based on templates. No single VM can be created without initiating a template for that layout first. The creation of templates and adding images via the CLI requires the user to set relatively detailed parameters.

Templates can be initiated through a pre-configured file, see Appendix 3, or by entering the command arguments one by one. The main commands for creating a

template, viewing all the templates, the removal of one of them and two full creation commands are as follows:

```
$ onetemplate create <params>
$ onetemplate list
$ onetemplate delete template-id

$ onetemplate create test-template
$ onetemplate create --name test-vm --memory 512 --cpu 1 --disk "Ubuntu Linux" --nic
Public --vnc
```

The parameters required for creating the image can be found from Appendix 4. The main image manipulation commands are the following:

```
$ oneimage create <params>
$ oneimage list
$ oneimage clone image-id new-filename
$ oneimage delete image-id
```

Initially, the image is created in to the system. Then all the present images are listed. The clone-argument is provided when an existing image is copied as another image, for example when versioning different solutions. Finally, the image can be deleted using its unique identification number. A working example of the oneimage command can be found in the next caption for reference in usage.

It is highly advisable to use the GUI in the creation of templates and adding images after familiarizing with the required settings. The settings are found under Virtual Resources -> Templates and Images.

Another goal for the project was the capability of using pre-configured VM images. This is a very handy feature, as it removes the need for the administrator to install each VM separately, and create blank pre-set environments for mass deployment. OpenNebula provides certain images via a marketplace, but these are generic images intended for all sorts of purposes. For some reason OpenNebula was able to read the images generated with other virtualization software, but KVM refused then to boot the

actual operating system from the image. This triggered no apparent error messages in the logs, and the only output was the statement of a missing hard drive in the virtual machine boot up. Since the default terminal Linux provided from the OpenNebula marketplace functioned perfectly, the pre-installed image was probably in a wrong format. Because of this situation, the disk image format had to be altered before submitting it to the system.

After a bit of research, the most reasonable solution for achieving this goal seemed to be the usage of Oracle VM VirtualBox on a regular desktop PC for the creation of the image. The selection was made based on the fact that the software is free to use and it is provided in both platforms used in this process (Windows-Linux). Therefore the migration from the Windows environment to the Linux should go as smoothly as possible, as the same software is used in both ends.

The desired virtual machine is created using the VirtualBox, and saved in the desired state. The image is then transferred to the server via for example secure copy (SCP) and converted to a usable form by using the tools provided in the Linux version of the software. After that, the OpenNebula image importing tools can fully utilize it and it can be run under any desired hypervisor. There is a notable feature in OpenNebula: the software refuses to import images from certain folders, such as the user home folder and the datastore base folder. This feature is considered to be a security improvement, although it is not included in the official documentation, merely in the update logs and discussion boards of the project. In the following is an example on how to convert and import a pre-installed image to the system.

```
$ VBoxManage clonehd source_image.vdi target.raw --format raw
$ mv target.raw /home/oneadmin/images
$ oneimage create --datastore default --name filename --path
/home/oneadmin/images/target.raw --description "free word description" --driver raw
--prefix hd
```

On the first line, the VBoxManage tool is used to copy and reformat the selected source image, in this case source_image.vdi, to another form. In this case the destination file is named target.raw, and the modifier –format raw specifies the file type. After this, the image is moved to be safe folder according to OpenNebulas

policies, and on the third line the OpenNebula image tools are used for creating a brand new image to the selected datastore from the generated raw image, specifying the used image driver as raw and the image being a hard disk image. After OpenNebula has processed the data, the image will appear in the datastore under the selected filename. Initially the image will be in a LOCKED state, and if the importing process ends without errors, the state will change to READY.

After all these settings have been provided, a virtual machine can be instantiated. This is done with the following commands in the CLI:

```
$ onetemplate instantiate template-name --name vm-name
$ onevm deploy host-id vm-id
$ onevm list
```

On the first line the created template is instantiated to a virtual machine with the selected name. OpenNebula will eventually deploy the instantiated machine, if there are resources available, but the user can force the deployment with the command on the second row. Using the list modifier the system outputs the complete list of all virtual machines and their states.

In the Sunstone GUI the virtual machines can be either instantiated through the Templates view via the button Instantiate, or through Virtual Resources -> Virtual Machines, Create. The virtual machine main view is presented in Figure 12.

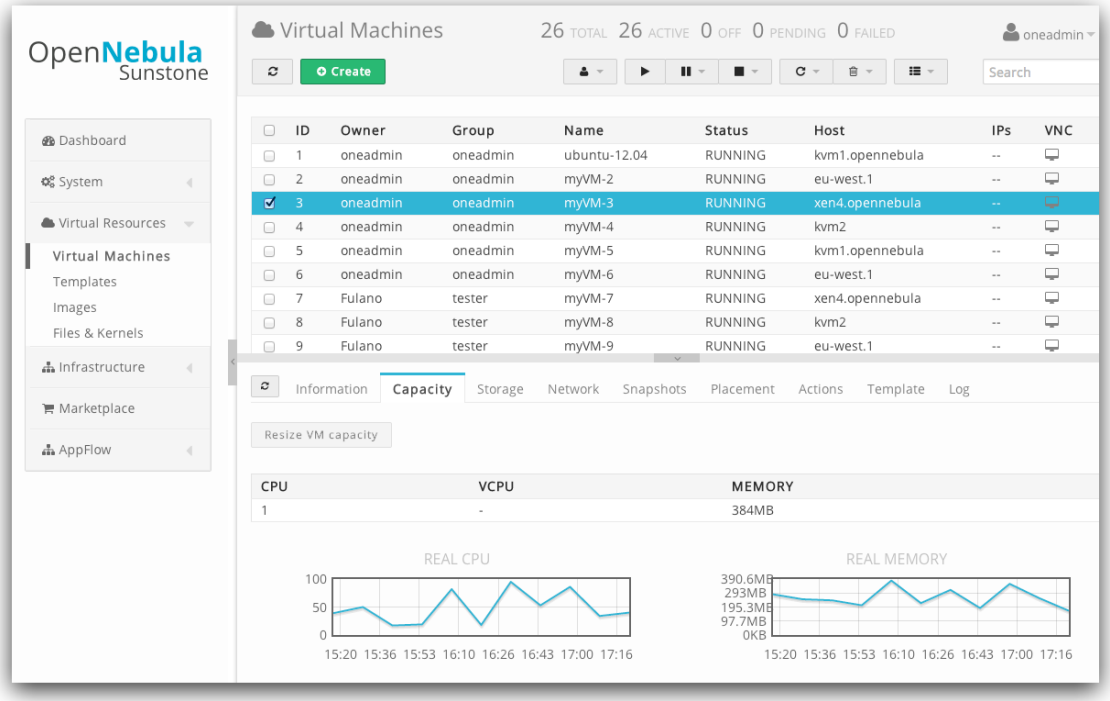

FIGURE 12. Virtual machines main view

Assuming that the virtual machine is deployed correctly, a screen icon appears on to the right side column under VNC after the VM has booted. Clicking this icon opens up a VNC connection to the VM, allowing direct usage through the browser.

5.5 Environment testing

After the environment installation was completed, a pre-installed virtual machine image was created on an external PC and then added to the system. The VM was configured to have 1 GB of RAM and have a 32-bit x86 processor, 8 GB hard drive with Ubuntu 12.04.3 LTS desktop version installed. The virtual image was left in a saved state of the OS being on in the desktop, scanning for updates.

25 virtual machines were initiated from a template created with the mentioned image and a network with NAT and access to the Internet. The virtual machines were then deployed on to the server, and the overall system response and VM deployment time followed constantly. It turned out that OpenNebula allocates almost all of the physical RAM at this point, even though it does not apparently use it all – out of the 64 GBs installed, only 200 MB were free during the initial deployment, even though the 25 machines consumed only a total of 22 GBs. The RAM was not utilized fully, as swap

space was not occupied by the hosting OS almost at all. After time had passed and the virtual machines were left running idle, the real consumed RAM dropped slowly down to the actual usage levels, from 64 GB down to circa 15 GB.

The environment slowed down significantly, and some of the OpenNebula functions came unavailable. Even though this happened, the already deployed VMs were available without any problems and could be used normally. The operating system was responsive and the environment was capable of rolling out a virtual machine in average 2-3 minutes each (total deployment time for 25 virtual machines was 1hrs3min). OpenNebula deployed the VMs in a queued form, allocating the host resources piece by piece, booting the VM up and then proceeding to the next VM in the queue. It does not over allocate resources without separate user interference; if any of the hardware limits (CPU or RAM) are met, the rest of the VMs are left in a PENDING state. This can be overridden by setting manually the VMs to be deployed on the desired host. After this OpenNebula shows the allocated host resources according to their correct percentage.

Another interesting point is the disk usage. As the VMs were created from a single raw image sized at 8 GB, and no other disks were allocated to the VMs, OpenNebula created single unmarked instances of this template image for each individual VM. This became apparent as the total disk usage increased, but no actual additional image files were added to the datastore or could be located from the host file system.

6 CONCLUSIONS

The original goal for this whole project was to create a virtualization environment suitable for educational purposes. The key requirements revolved around the performance of the system under constant usage and the ease of virtual machine deployment. To meet these goals, different approaches in the fields of virtualization and cloud computing were studied and then applied in practice.

Two major obstacles can be identified preventing the progress in the project: the SSD utilization and the pre-installed virtual machine image submission. Since the SSD manufacturer does not provide adequate support for the hardware in Linux

environments the original design goals for using it as a cache and/or storing the operating system on it could not be met. The second matter should not theoretically even exist: as the virtualization software providers have agreed to utilize standardized virtual machine and image formats, the migration between any systems should go seamlessly. Reverting to flat raw images should not be required, but fortunately alternative tools to reach the original goal are available.

In overall I would estimate that the project was successful. As it could be anticipated beforehand, there were minor and major obstacles in the process, which were overcome in the end, one way or another. The designed system is fully operational and is usable for its purpose “as-is”, but the selected tools still allow further development in the subject. In the following a few points related to the execution of the project and possible future developments are listed.

The first thing is the utilization of the SSD in the system to a larger extent. As the selected operating system is open-source, the community might develop updated drivers for the device, allowing the operating system to boot from the disk. In that scenario the external journaling method would become a viable solution. These hindrances prove a valuable point, though: the hardware used for any environments should be picked with extreme consideration to avoid unwanted surprises like this.

As OpenNebula is capable of utilizing resources provided by multiple hypervisors, it would be possible to add at least temporarily additional hosts to the cloud. These could be installed with for example VMware ESXi, as it is relatively stable and easy to install and add it to the existing cloud. After this features such as live migrations, high availability and load balancing could be tested extensively within the cloud. To take further advantage of the environment OpenNebula Zones (oZones) could be applied. This would require additional hardware resources, as oZones is a component designed for monitoring and controlling multiple separate OpenNebula clouds. This would permit the development of a hybrid cloud.

Another feature worth testing in OpenNebula is the automated hooks for controlling all resources. OpenNebula allows the user to configure actions based on resource states (such as a VM resulting in ERR state) to automate the cloud functions. As enterprise solutions should be capable of autonomous actions, experimenting with

these hooks could be worth valuable experience in the operational logic of a cloud. In addition to this, monitoring software, such as Ganglia, could be connected to the cloud to provide long-term statistics.

In the end it has to be noted that since the IT field develops rapidly and the majority of tools used in this project are based in open-source, the knowledge gained from here might not apply after a few years. The base logic will probably stay the same for a while, but the specific tools used for achieving better results develop constantly. In order for the university staff to maintain a sufficient level of education for the students, the end result of this project could be revised when the methods used here are becoming outdated.

BIBLIOGRAPHY

[1] Amit Singh. An Introduction to Virtualization. Website. 2004. http://www.kernelthread.com/publications/virtualization/. Referred 13.11.2013.

[2] Charles David Graziano. A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds project, PDF Document. 2011. http://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=3243&context=etd. Referred 18.11.2013.

[3] VMware Inc. Understanding Full Virtualization, Paravirtualization, and Hardware Assist. PDF Document. 2007. http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf. Referred 18.11.2013.

[4] Michael Armbrust, Armando Fox et al. A view of cloud computing. PDF document. 2010. http://delivery.acm.org/10.1145/1730000/1721672/p50-armbrust.pdf. Referred 18.11.2013.

[5] International Telecommunication Union. Key 2006-2013 ICT data for the world. Excel document. 2013. http://www.itu.int/en/ITU-D/Statistics/Documents/statistics/2013/ITU_Key_2005-2013_ICT_data.xls. Referred 18.11.2013.

[6] National Institute of Standards and Technology. The NIST Definition of Cloud Computing. PDF Document. 2011. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. Referred 13.11.2013.

[7] Stephen R. Smoot, Nam K. Tan. Private Cloud Computing. 2012. USA. Morgan Kaufmann Publishers.

[8] Rajkumar Buyya, James Broberg and Andrzej M. Goscinski. Cloud Computing: Principles and Paradigms. 2011. Australia. John Wiley & Sons, Inc.

[9] Microsoft Corporation. Windows Server 2012 R2 Datasheet. PDF Document. 2013. http://download.microsoft.com/download/D/2/C/D2CDA5BA-E440-4A50-A418-5362291156C1/Windows_Server_2012_R2_Datasheet.pdf. Referred 13.11.2013.

[10] VMware, Inc. VMware vCenter Server Datasheet. PDF Document. 2012. http://www.vmware.com/files/pdf/products/vCenter/VMware-vCenter-Server-Datasheet.pdf. Referred 18.11.2013.

[11] Giovanni Toraldo. OpenNebula 3 Cloud Computing. 2012. United Kingdom. Packt Publishing Ltd.

[12] Microsoft Corporation. Unified management for the Cloud OS. PDF Document. 2013. http://download.microsoft.com/download/7/7/2/7721670F-DEF0-40D3-9771-43146DED5132/System_Center_2012%20R2_Overview_White_Paper.pdf. Referred 19.11.2013.

[13] AccelCloud Services. Linux flashcache and bcache performance testing. Website. 2012. http://www.accelcloud.com/2012/04/18/linux-flashcache-and-bcache-performance-testing/. Referred 13.11.2013

[14] Libre Solutions Pty Ltd. ext4: using external journal to optimize performance. Website. 2013. http://www.raid6.com.au/posts/fs_ext4_external_journal/. Referred 16.11.2013.

[15] Steve Soltis, Grant Erickson et al. The Design and Performance of a Shared Disk File System for IRIX. PDF Document. 1998. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=F6A956711348D113C2E84DFF90D0560F?doi=10.1.1.3.9932&rep=rep1&type=pdf. Referred 20.11.2013.

[16] Internet Engineering Task Force. Internet Small Computer Systems Interface (iSCSI). Website. 2004. http://tools.ietf.org/html/rfc3720. Referred 20.11.2013.

[17] Paul Barham, Boris Dragovic et al. Xen and the Art of Virtualization. PDF Document. 2003. http://www.cl.cam.ac.uk/research/srg/netos/papers/2003-xensosp.pdf. Referred 20.11.2013.

[18] Luis M. Vaquero. EduCloud: PaaS versus IaaS Cloud Usage for an Advanced Computer Science Course. PDF Document. 2010. http://jpinfotech.org/wp-content/plugins/infotech/file/upload/pdf/3741EduCloud-PaaS-versus-IaaS-Cloud-Usage-for-an-Advanced-Computer-Science-Course-pdf.pdf. Referred 20.11.2013.

[19] Muli Ben-Yehuda, Michael D. Day et al. The Turtles Project: Design and Implementation of Nested Virtualization. PDF Document. 2010. https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Ben-Yehuda.pdf. Referred 21.11.2013.

[20] Peter Sempolinski, Douglas Thain. A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. PDF Document. 2010. http://www.cse.nd.edu/~ccl/research/papers/psempoli-cloudcom.pdf. Referred 21.11.2013.

APPENDICES

Appendix 1. Template file for a fixed virtual LAN

```
NAME    = "Blue LAN"
TYPE    = FIXED
BRIDGE  = vibr1
LEASES  = [IP=192.168.100.1]
LEASES  = [IP=192.168.100.2, MAC=50:20:20:ff:ff:ff]
LEASES  = [IP=192.168.100.3]
LEASES  = [IP=192.168.100.4]
GATEWAY = 192.168.100.1
DNS     = 192.168.100.1
LOAD_BALANCER = 192.168.100.4
```

Appendix 2. Template for a ranged virtual LAN

```
NAME    = "Red LAN"
TYPE    = RANGED
BRIDGE  = vibr0
NETWORK_SIZE    = C
NETWORK_ADDRESS = 192.168.122.0
GATEWAY = 192.168.122.1
DNS     = 192.168.122.1
LOAD_BALANCER = 192.168.122.3
```

Appendix 3. Virtual machine template creation template

```
NAME   = test-vm
MEMORY = 512
CPU    = 1
DISK = [ IMAGE  = "Ubuntu Linux" ]
DISK = [ TYPE     = swap, SIZE     = 1024 ]
NIC = [ NETWORK = "Public", NETWORK_UNAME="oneadmin" ]
GRAPHICS = [ TYPE    = "vnc", LISTEN  = "0.0.0.0" ]
```

Appendix 4. Image creation parameters

-d, --datastore id|name	Datastore where the image will be put
--name name	User selected name for the image
--description description	Free word description of the image
--type type	Type of the new image: OS, CDROM, DATABLOCK, KERNEL, RAMDISK, CONTEXT
--persistent	If the image is to be stored as persistent
--prefix prefix	Device prefix for the disk (eg. hd, sd, xvd or vd)
--target target	Device the disk will be attached to
--path path	Path of the image file
--driver driver	Driver to be used by image (raw, qcow2, tap:aio)
--disk_type disk_type	Type of the image (BLOCK, CDROM, RBD or FILE)
--source source	Source to be used. Useful for not file-based images, such as web-based images
--size size	Size in MB. Used for only for DATABLOCK type

Appendix 5. Virtual machine creation parameters

--name name	Name of the new virtual machine
--cpu cpu	A number value of assigned CPU capacity
--vcpu vcpu	How many virtual CPUs are assigned
--arch arch	CPU architecture of VM, such as: i386 or x86_64
--memory memory	Amount of assigned RAM, megabytes by default
--disk image0,image1	Disk images to be attached
--nic network0,network1	Attached networks
--vnc	To add VNC server to the VM
--boot device	Selecting boot device (hd|fd|cdrom|network)
-m, --multiple x	The total count of VMs to instantiate