UNIVERSITY OF AMSTERDAM

Moving SNE to the Cloud

RP1 Report

Sudesh Jethoe

30-12-2011

*[27]

Summary: In this research the OpenStack framework is used to build a private , public and hybrid cloud for the System and Networking education of the University of Amsterdam. Discussed are the setup used and problems encountered while building it.

Index

1 Introduction

The research explained in this report was conducted for the System and Network Engineering education at the University of Amsterdam. It is part of the course "Research Project 1". This course gives students one month to conduct a research. Because of this time frame, decisions were made that limit the technical scope of this research. The goal of this research is to find out whether or not a cloud framework can be deployed successfully in the System and Network Engineering educational environment. To answer this question, first we explain the background of cloud computing. Then we explain why cloud computing can be advantageous for the System and Network Engineering education. Following, the questions used in this research will be given. Then we will describe the approach we took. Our method will be described after explaining the reasons for selecting a specific cloud framework. This is necessary because the practical part of the research depends on the exact cloud framework used.

After reading this report the reader should be able to:

- Identify the upsides and downsides of different types of cloud environments.
- Have an overview of available cloud frameworks.
- Identify pitfalls which can be encountered when building cloud environments.

2 Background

The cloud is these days often considered as a valuable resource not only in businesses, but also in educational environments. The advantages of improved scalability, agility, consolidation and cost efficiency in comparison to regular server based computing are leveraged already to some extent in universities [1]. In this chapter we describe the development of cloud computing and how its advantages are achieved.

Clustering, Scalability

In classic server based computing a user would log in to a physical machine which runs the required service. An example of a service which such a physical server would host is e-mail. A physical server has a maximum amount of users it is able to serve. The main reason being that a physical machine can only hold a certain amount of physical resources (being disks for storage and processors for computational power).

Due to increasing amounts of users, more and more expensive servers need to be bought or they need to be upgraded. To be able to serve the same amount of users as one big server, a method was developed to cluster smaller (and cheaper) machines together. By using clusters, services could be spread over multiple machines and as a side effect risks of downtime were also mitigated. This mitigation was achieved by making sure that the software running on top of these systems would take care of outages not only of disks or processors, but complete machines in a cluster. As an added effect, these systems had the advantage of being more scalable than classic servers, since servers could be added and removed as needed.

A downside to clusters is that programs written for classical operating systems can not run on a cluster. Since classical applications need to run on physical servers, running non-cluster operating systems, the advantages of cluster computing could only be utilized to a certain extent.

Consolidation, Virtualization

The increasing capabilities of physical servers (more computational power and storage) on the other hand, lead to the development of methods to do the exact opposite of clustering, consolidation. Consolidation being methods to run multiple services on a single server as opposed to running a single service on multiple servers. Although the UNIX time sharing system enabled concurrent usage of systems already in the early stages of server based computing, methods of extended segregation

of processes were devised for added security. Examples of these so called "Sandbox" methods are the BSD jails and Unix chroot environments.

A method developed at a later time, allowed to run not only one but multiple of these classical operating systems not directly on physical hardware, but on top of a so called "hypervisor". This method is known as virtualization and consists of a software layer running on the physical hardware and takes care of the distribution of physical resources amongst the operating systems which run on top of it. Virtualized operating systems are also known as virtual machines (VM's). Virtualization increased security by ensuring services did not run in the same environment and increased options for consolidation, since it became possible to run services running on different operating systems to be hosted on the same physical machine.

Provisioning, Agility

Fully utilizing the capacity of clusters was also an issue. When running services directly on top of physical hardware, it is not possible to plan the capacity of a server on the average amount of users. Servers should be prepared to handle a peak capacity. Unfortunately this means that servers are idle a great deal of their time. Therefore methods were devised to dynamically reconfigure servers to handle a varying set of computational tasks, depending on the demand of these tasks. This is primarily done by a combination of automated installation of operating systems (this is also known as provisioning) and automatic reconfiguration achieved through custom scripting, network booting and/or specialized software. Examples of tools used for automatic configuration management are Puppet and Cfengine. Through the use of provisioning and automated configuration management, it is possible to switch between the tasks servers need to execute relatively fast and makes for increased agility.

Putting it all together: Clouds

By combining methods for clustering, virtualization, provisioning and configuration management a flexible, agile clustered as well as consolidated environment can be built. This is what is generally called a cloud. A cloud has the scale advantage of a clustered environment and also the advantage of virtualization, which enables it to run classical applications on classical operating systems. Besides, the combined effect of provisioning and virtualization enables full utilization of available hardware, without the need for downtime. In effect it can be seen as a "best of all worlds" of all of the forms of computation described above.

The necessary hypervisors, configuration- and networking software, can be tied together using custom scripts. However, frameworks exist which integrate this functionality to any certain extent. We will discuss the various properties of these frameworks in chapter 4.1 "General cloud design".

Iaas, Paas, Saas

All the discussed forms of computing can be offered as a hosted service. Depending on the extent of access offered to the system, there is a distinction in three forms, namely IaaS, PaaS and SaaS.

- SaaS: "software as a service" and is used to indicate hosted software solutions ready to be used by end users. Examples are web applications.
- PaaS: "platform as a service", indicates a set of programming interfaces, which enable developers to build SaaS applications on top of a cloud or cluster. Examples are Google App Engine and Windows Azure.
- IaaS: "infrastructure as a service", providers of this type of service offer the possibility to manage and host your own operating systems and software in a hosted virtualized environment. The biggest public service providers of this type of service are currently Amazon and Rackspace.

Public/Private/Hybrid Cloud

Cloud platforms hosted by public service providers are known as “public clouds”.
It is also possible to host clouds on a private infrastructure. It can be done either by combining applications with custom scripting or by using specialized frameworks. Clouds hosted on a private infrastructure are also known as “private clouds”.

It is also possible to mix public and private infrastructures together. The extents to which infrastructures are tied together vary. It can be only a connection through which data automatically flows from a company LAN to a SaaS provider for processing. It can be an extra set of virtual webservers, to add additional capacity to an existing webserving farm. A cluster of virtual servers of which a part run in the public cloud and another part runs in the private cloud. Clouds which are mixed are called hybrid clouds. Public cloud service providers, often provide additional services to build hybrids. Amazon, for example, provides its “Virtual Private Cloud” (VPC) services. This service allows cloud builders to connect clouds with an IPsec tunnel, offering the possibility of a shared IP-space.

3 Clouds in the SNE environment

Now that a common ground in cloud computing is established, it is time to discuss its relevance to the *System and Network Engineering* (SNE) education. We will do this by discussing the advantages and requirements of clouds to the SNE education.

3.1 Advantages of clouds to SNE

Currently the education provides all students with their own physical server to run the required experiments on. Unfortunately this solution is not very flexible, expensive and limits the number of students being able to participate in the education. Moving to a cloud environment can be beneficial to the education. By pooling the physical resources and building a (private) cloud infrastructure, the following advantages can be achieved:

- Consolidation, more students can be hosted on the same amount of hardware
- Flexibility, student VM's can more easily be migrated to other hardware. This can be beneficial when there are for example hardware defects or a temporary demand for more processing power.

Attaching this private infrastructure to a public cloud environment might also be advantageous. It can add the ability of the cloud environment to scale out, without adding additional hardware to the private environment. This allows for the hosting of additional VM's and allows more students to take part in the education. Although we could also investigate the possibility of running the complete environment exclusively in a public cloud without making use of a private environment, we want to fully utilize existing infrastructure.

How do we want to realize these advantages

Although it is possible to build a (private) cloud manually by using custom scripts and hypervisors, we want to make use of a framework. We expect it takes less time to use an existing framework than manually "build a cloud". We also want to find out, whether a framework can actually provide what the education needs. After we have built the private cloud, we want to connect to a public cloud service provider, in order to create a hybrid. Due to the type of services the education needs to offer, the only type of cloud service provider which can be used is one that provides "Infrastructure as a Service".

Requirements

In order to determine whether the solution can be used within the education, we have determined that the solution should at least be able to let students:

- Set up their own Linux and Windows (virtual) machines.
- Set up their own (virtual) networks and routes.
- Delegate IP-space
- Run Internet services like DNS and e-mail.
- Secure and administer these environments

3.2 Research Questions

Considering these requirements, our research questions will be:

- Is it possible to execute SNE-education practica in the {public/private/hybrid} cloud?
- Can connectivity be achieved on a level which allows the creation of custom layer 3 (IP) networks by users of this environment?
- Can full cloud transparency be achieved?

With full cloud transparency we mean virtual machines behave the same whether they are in the public or private parts of a hybrid cloud. They should be able to operate in a shared IP-space and not need modifications to run in another environment. Having this ability greatly simplifies the management of a hybrid environment.

4 Cloud design and the choice for OpenStack

The goal of this section is to discuss cloud design and make a choice from available cloud frameworks. We will do this by first looking at general cloud design. After that we will give further detail to the selected cloud-framework for the private cloud. Finally we will show the available options for connecting this private cloud to a public cloud.

4.1 General cloud design

Clouds consist of a few main components (figure 1). At the base of the system is the hypervisor (Xen, KVM, VMware) running directly on a physical server. Hypervisors provide an abstraction layer which enables the dynamic allocation of physical resources. These physical resources are accessed via virtual machines (VM's). To distribute VM's amongst the nodes, the hypervisors on the nodes need to be managed.

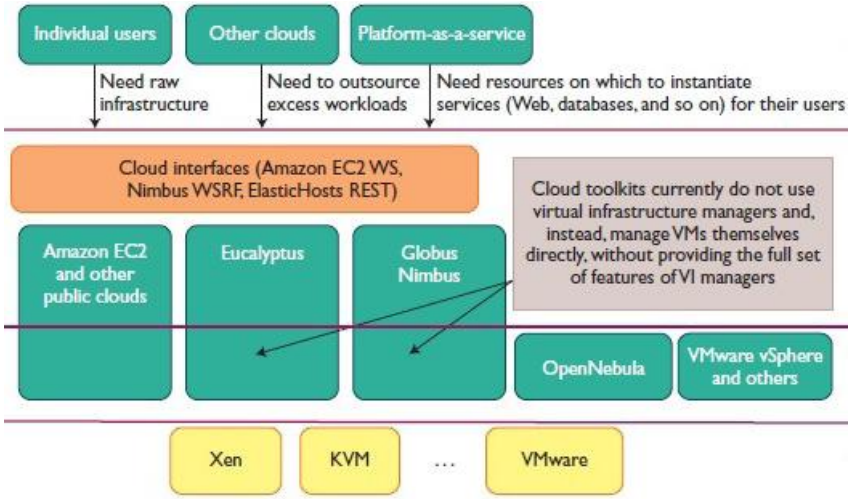

Figure 1: The cloud ecosystem for building clouds [5]

Virtual infrastructure managers

A "virtual infrastructure manager" (VIM) is more hypervisor oriented and focusses on giving administrators more control of launching and migrating VM's [6]. An example of how this control is achieved is by providing a direct interface to the hypervisor through an API (libvirt [9]). Examples of VIM's are OpenNebula [6] and VMware vSphere [7].

Cloud frameworks

The distribution can also be managed by making use of a cloud framework. Cloud frameworks focus more on interfacing with users and at the same time try to decrease the management complexity of a cloud infrastructure. This is achieved by providing access to the resources via more generic Amazon EC2 and S3 compatible API's. Besides, they offer options for users to directly access isolated parts of the cloud, for example by using SSH. Examples of cloud frameworks are VMware's vCloud [8], Eucalyptus [9] and OpenStack [4].

Since cloud frameworks are focused on exposing cloud resources to users, rather than being built to serve administrators as with virtual infrastructure managers, only a cloud framework will be considered. In this way the necessity to build an interface on top of a VIM can be avoided.

VMware's vCloud is a cloud framework, however it is proprietary and, therefore, it is not a suitable candidate to be used in environment like that of the SNE-education, which focuses on the use of open -standards, -software and -security [10]. Since the development of cloud frameworks is still in

its early stages, another important consideration is the active development base. Recently though, the major Linux software vendors Red Hat and Canonical announced support for OpenStack, with their respective distributions Fedora and Ubuntu. Where Ubuntu used to ship Eucalyptus as a main component in their "cloud foundation technology" [12], they switched to OpenStack to be their leading cloud platform. Besides these two companies, also Citrix, Dell, Intel, NASA and Cisco have opted for OpenStack as the default cloud platform. This can give OpenStack an advantage in the long run.

4.2 OpenStack design

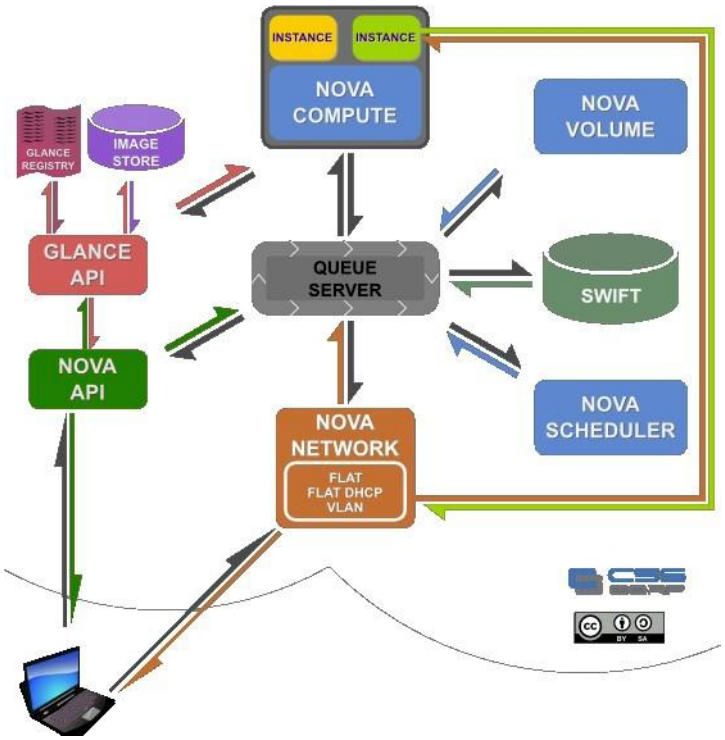

Figure 2: OpenStack architecture [8] with components: Nova (Compute), Swift (Storage), Glance (Provisioning), Environments are secured by setting up iptable firewalls.

The OpenStack cloud framework consists of three main components which itself are split in smaller parts that can run independently. The components manage computational power, storage and the provisioning of virtual machines to physical nodes.

4.2.1 OpenStack Nova (compute) components

Nova is the core of the OpenStack framework. It consists itself of parts, which are responsible for interfacing and managing the creation, removal and distribution of instances among the physical nodes in the cloud. A fully operating cloud can be run using only this component. It consists of the following parts:

- Nova-API: The nova-API is the component which is used to interact with the infrastructure from the outside. This component is also used by client programs which can be run from a local desktop. The API allows users amongst others to; upload VM's, start- and stop VM's, add persistent storage and view the status of servers in the cloud environment.

- Nova-volume: The nova-volume component is used for attaching persistent storage volumes to instances. This is useful since spawned instances are not persistent themselves. Nodes running the nova-volume component take care of the creation and attaching and detaching of volumes to instances. Volumes are stored either locally on the node or using a specialized

component, like Swift. Swift is the distributed storage system for OpenStack and will be discussed later on in this chapter.

- Nova-scheduler: The nova-scheduler distributes the API-calls among the various nodes, it does this while taking into account factors as load, memory, zone availability and processor architecture.

- Nova-compute: Nodes running the nova-compute component are the ones actually hosting the virtual machines. They manage the lifecycle of an instance. Instances are manipulated through an API which enables the framework to boot, reboot and terminate instances, attach and detach volumes and retrieve the consoles output. The framework is hypervisor agnostic and currently supports the following hypervisors; KVM, UML, XEN, Hyper-V and QEMU.

- Nova-network: The nova-network component manages the traffic flow between the various nodes in the network. It creates the necessary networking configuration (addresses, iptables, bridging) when launching instances, applies the configured security groups and, if applicable, VLAN's. A security group is collection of rules which apply to a set of VM's which run in the same subnetwork. It can be used, for example to allow access to port 80 on a collection of webservers.

In a setup where the nova-network component is running on a central node in the network, each VM running on a compute node is connected to this component through a virtual bridge. In this way, each VM can get an address directly from the nova-network component, see the figure below.

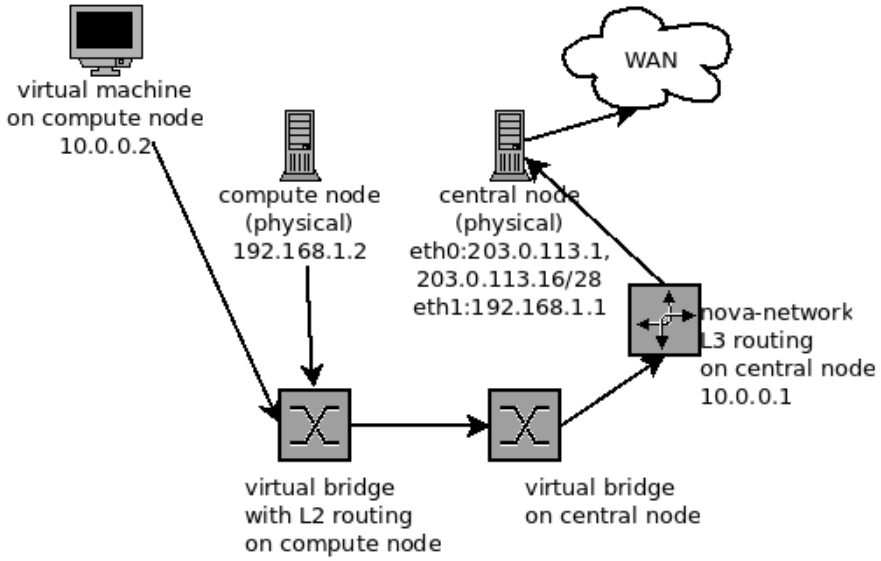

Figure 3: Traffic flow to WAN in OpenStack [14].

In this figure the traffic flow from a VM to the internet is shown. The VM has a private address in a range defined for the group this VM belongs to (in this case, the 10.0.0.0/24 range). It runs on a compute node which is in the same subnet as the central node (192.168.1.0/24). The compute node and the central node can be connected either directly or via a switch.
Traffic from the VM is routed through dynamically created layer 2 routing rules on the compute node. The nova-compute component running on this node creates these rules.

On the central node the traffic flow is managed by the nova-network component. It creates a virtual bridge with the necessary rules to receive and send traffic from the VM's. The nova-network

component also ensures the creation of layer 3 routing rules when VM's need to be reachable from the WAN. This is done by network address translation (NAT). In figure 3 the central node was assigned the public address 203.0.113.1 and the subnet 203.0.113.16/28. Addresses from this subnet can be delegated to the VM's through the nova-network component running on the central node. In this figure the network component can create a mapping between the 10.0.0.2 address assigned to the running VM and 203.0.113.20, an address from the public range assigned to the central node. Although OpenStack also supports IPv6 it is not yet fully developed and is therefore not considered in this research.

Currently three network models are supported in OpenStack:

VLAN-networking

Packets originating from VM's are tagged with a VLAN-id by the nova-compute component when leaving the compute node and are sent to the node running the nova-network component. The nova-network component runs a dnsmasq service for each VLAN. The dnsmasq program acts in this case as a DHCP-server. In this way all instances within the same VLAN are in the same subnet. To be able to use this kind of setup, the switches connecting the internal network should support VLAN-tagged ethernet frames.

Flat-networking

Ethernet frames originating from instances running in the cloud are routed to the central node through ebtables [14] and get an address directly from the nova-network component.

DHCP-networking

Ethernet frames originating from instances running on compute nodes are routed to the central node through ebtables and get an address from a dnsmasq service running on the central node.

Initially all instances get a single IP-address, assigned through one of these methods. Optionally additional addresses can be assigned to an instance. These are however not assigned to the instances directly, but network address translation is used at the central node to map these addresses to the private IP-addresses of the instances (figure 3). Unfortunately this means that, without adding extra routing rules in the instances, it will not be possible to delegate public IP-space within instances, when using flat- or DHCP-networking.

4.2.2 Swift (storage)

Swift provides a distributed storage system for OpenStack. It offers options for storing up to a million of objects in a single container. Objects however have a 5 GiB size limit. Objects are replicated and distributed among a predefined number of storage nodes. Replication occurs according to a weak consistency model and the consistency between replicated objects can therefore deteriorate under increasing load. The weak consistency is caused by the fact that objects are initially stored on node in the pool and subsequently copied over to other nodes from this node. However, when updating the object on one node, other copies of the object are not locked for reading. Other instances requesting the same object from a different storage node might therefore acquire different data.
Swift can also be used for storing virtual machines and cloud apps.

4.2.3 Glance (provisioning)

Glance is the OpenStack component that provides virtual machine images for the compute component (Nova). It can be setup to use multiple back ends, namely the OpenStack object storage system Swift, Amazon's "Simple Storage Service" (Amazon S3) or a hybrid in which interfacing occurs through Swift.

4.2.4 Managing OpenStack

In OpenStack management of cloud resources is achieved by communicating with the system through the nova-API. Running components can be manipulated locally through the "nova-manage" command. This enables the local administrator to start, update, stop and reboot all components like nova-compute, nova-network and the hypervisor.

The framework can also be controlled through the cloud administrator account, which can manage for example instances, storage volumes, projects and subnetworks.

An OpenStack project is a collection of cloud resources, which in itself can be considered a cloud. When creating a project, a cloud admin assigns a user and a fixed private subnet to this project. The user can then upload virtual machines to the cloud and spawn instances of this virtual machine in the assigned project. These instances will get a private IP-address from the assigned subnet, public IP-addresses can be appointed by the users or the cloud admin.

4.3 Amazon virtual private cloud types and choices

In order to build a hybrid cloud the private cloud needs to be connected to a public cloud. For this research we will be making using of the Amazon public cloud. This public cloud provider has been selected because it is the biggest provider of public cloud services [15].

However, to determine whether or not full cloud transparency can be achieved, it was also necessary to attach our private cloud to the Amazon cloud. The most obvious way to connect these clouds was to make use of an Amazon "Virtual Private Cloud" VPC [12], an extra service of the cloud service provider Amazon to connect a private cloud to the Amazon EC2 public cloud.

Option 1

Set up virtual layer 2 connectivity between the public and private clouds. This could be achieved by creating a tunnel. An advantage of this kind of setup is that it requires the least amount of changes in the OS3 infrastructure. It can be expected however that having connectivity at layer 2 directly between the clouds would create too much traffic between the clouds, since all broadcast and multicast messages from higher layer protocols will be sent across this tunnel. Besides, Amazon does not offer this kind of connectivity in its cloud.

Option 2

Set up a tunnel to every node running in this cloud. This would create lots of overhead traffic, since all communication to the Amazon cloud would be happening multiple times to make sure each compute node gets the necessary information. This might also cause added latency.

Option 3

The other solution for connecting the local networking infrastructure to the Amazon cloud is to use a routing protocol. The goal of a routing protocol is to distribute routes across a networking infrastructure. This is exactly what is needed to connect the two sites. Amazon offers support for the "Border Gateway Protocol" (BGP) routing protocol with its Amazon Virtual Private Cloud (VPC). It is a collection of services which provide options used to build a private cloud inside the Amazon cloud, the options offered are:

- VPC with a Single Public Subnet Only
- VPC with Public and Private Subnets
- VPC with Public and Private Subnets and Hardware VPN Access
- VPC with a Private Subnet Only and Hardware VPN Access

From these options the last two can be considered to be suitable candidates. They provide the option to connect to the local infrastructure through a VPN. Within this tunnel Amazon provides a virtual BGP-router which will be used to make sure that VM's with their delegated IP-space are reachable from either side of the clouds and from the internet. Therefore, an Amazon VPC was created.

5 OpenStack configuration and clouds setup

The different cloud frameworks have been discussed. Also a selection and overview of these frameworks have been given. Therefore we can now focus on the setup of these private and public cloud frameworks.
First we define ways in which OpenStack can be applied within the System and Network Engineering education. From these ways we select one to base our setup on. This will be described in section 5.2 “Private cloud setup”. Finally the way in which the connection between the OpenStack and Amazon cloud will be built is described.

5.1 OpenStack configuration

In this setup a VM was installed to test the features necessary for successful deployment within the education. These are the features described in the section requirements of chapter 2. Besides these basic requirements, another concern is how students can access this system and how they should use it. Although this can be considered a requirement as well as those described in chapter 2, we discuss them now, since the features vary depending on the framework used.

The OpenStack management model supports multiple ways of allowing students access to cloud resources.

Classical approach

One could be a more classical approach in which, instead of giving each student one physical server, all students get one instance. On top of this one instance, they would then have to build all experiments (virtualization, delegation of public IP-space, run internet services, secure environments). Although this would make the management overhead relatively low compared to other options, it is not very practical. When considering the requirements stated in chapter 2, we see that students need to be able to run multiple virtual machines. The only way this can be achieved when allowing students one instance is by forcing them to run nested virtual machines. This however causes increased overhead is still very experimental and therefore not suitable for this research.

Project per student

The other approach could be setting up an OpenStack project for each student, as is described 3.6 “Managing OpenStack”. This kind of approach would allow students to administer their own cloud, set up their own virtual machines, run both Linux and Windows and manage their own IP-space. Since this setup makes it possible to spread the experiments across multiple VM's, more different types of experiments become possible in this way. A setup like this fulfils most of the requirements stated in chapter 2, therefore we will use this type of setup.

5.2 Private cloud setup

Initially the private cloud was built according to the OpenStack reference design [14]. This setup consists of two physical servers, which together host four components. The components nova-volume, -scheduler and -network are all installed on one physical server, the central node. The nova-compute component is installed on the second server, called the compute node (figure 3). Although it is also possible to run all the necessary components on one machine, this does not reflect a realistic scenario. Making use of two machines ensures that the setup is scalable from the start, since extra compute nodes can be added at a later time with relative ease.

The most recent version of Ubuntu Linux (11.10 Oneiric Ocelot [12]) was used as a base system. The Ubuntu operating system was chosen because it is the main development platform for OpenStack. This means the latest version of the OpenStack framework (Diablo [13]) can installed with minimal adjustments to the operating system. The advantage of running the latest version of the framework

is that it we can avoid bugs already solved in older versions. The installation was done according to the reference design [8].

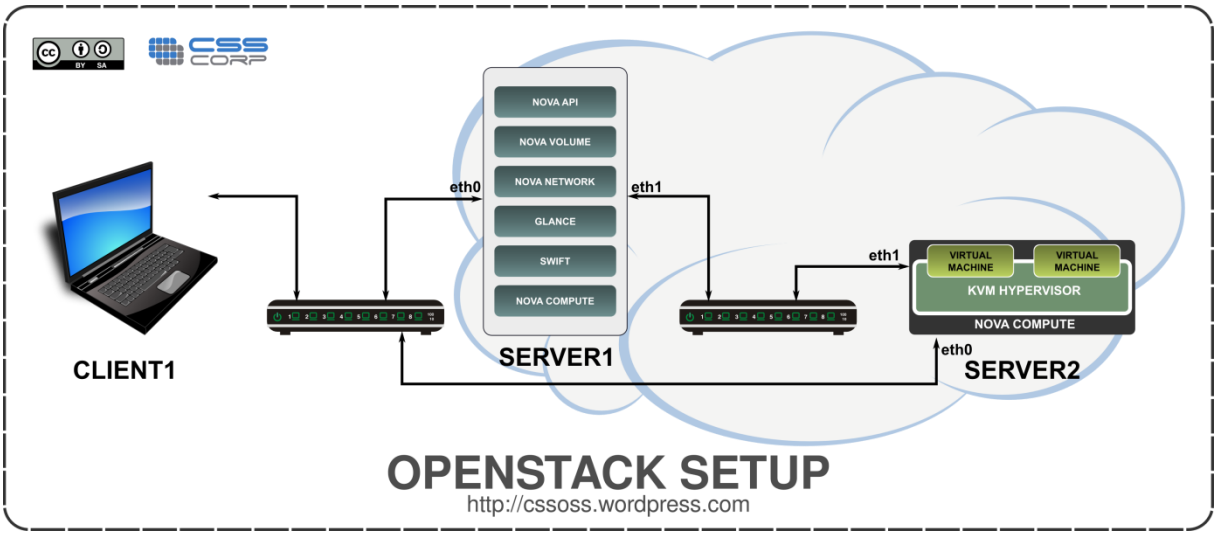

Figure 4: Reference design, dual node deployment OpenStack [15]

5.2.1 Private cloud setup according to the reference design

The reference design splits the network in two parts. A public facing interface on the controller node and a management network between the controller and compute nodes. Since the OS3 networking environment uses public IP-addresses, the reference design could not be followed to full detail and IP-addresses were configured differently where needed. An overview of the initial configuration can be found in appendix I. Another point in which the setup was changed was the connection between the central node and the compute node. Although they were already connected via a switch, they received an extra ethernet cable directly to each other. This was mainly to offload the switch that also had other purposes not related to this research.

During the configuration of the system several errors were encountered in the setup of the storage service (Swift) and the imaging service (Glance). Since they are not relevant to this research they are documented in appendix III.

While configuring the network other issues were found. Initially instances would spawn, however it was observed that the private IP-addresses of instances were not assigned by the nova-network component. Although the OpenStack network should make sure that the appropriate routing rules are configured on all nodes, we did not manage to achieve this with the settings from the OpenStack reference guide. Routing rules would not be applied on the controller- and compute-nodes and launched instances would not acquire an address and thus they were not reachable from the outside.

This might be due to the fact that unlike the reference design (figure 4), the used setup did not contain a physical switch between the central node (server1) and the compute node (server2). Therefore configuring the network with the OpenStack VLAN networking model [19] might not be appropriate.

5.2.2 DHCP- & Flat-networking models

The other OpenStack networking models, namely DHCP- and Flat-networking, were tested to see if these would lead to connectivity. However the observed behaviour was consistent (instances would start on the compute nodes, but without connectivity. Since this seemed very much like a configuration error and time was running out, the decision was made to switch to another Linux

distribution, which ran with OpenStack preconfigured. This distribution was StackOps [19]. Using this distribution it was possible to create a working setup.

5.2.3 Private cloud setup using StackOps

StackOps is based on Ubuntu 10.04 and comes with the previous release of OpenStack, named Cactus. With StackOps it is only possible to configure the network using DHCP and only predefined setups can be configured. In this case the "dual node" deployment was chosen. This is exactly the same setup as before. It uses two servers, where the first is running the management component (central node) and the second is hosting all the instances (compute node). While using this distribution, initially there was still no connectivity. Although the cloud framework would launch the instances and show their assigned IP's (see figure 5), they would not respond to pings.

```
$ euca-describe-instances
RESERVATION     r-aws7z4p8      sne-cloud       default
INSTANCE        i-00000012      ami-46ccba64    10.0.0.2 10.0.0.2
running None (sne-cloud, waldorf)       0               m1.small
2011-12-14T15:41:31Z    nova
```

Figure 5: Output of command to describe running instances, shown with corresponding public and private IP, cloud-project and compute node

To find out if the outside was also not reachable from within the instance, a direct connection was made to the KVM instance running on the compute node, by connecting with a virtual machine manager from a desktop. When connected directly to the console of the instance, it was seen that the default route and IP-address were not being assigned to the instance. When configuring the default route and IP-address by hand, this would lead to connectivity to and from the instance.

Since the networking type is always DHCP in StackOps, we checked whether the dnsmasq service was actually running on the central node. This was not the case, so a custom dhcp server (udhcpd) was installed on the central node, to find out whether this would lead to the instances getting an address from the server. This lead to connectivity. Since we were unable to find out what prevented the dnsmasq service from starting we archived the installation and reinstalled both physical nodes with different settings (appendix II).

This time the instances did acquire an IP-address from the cloud framework. It was however still not possible to attach public IP-addresses to the instances, using the default cloud management tools. Due to errors (appendix III) in the communication between the tools and the database which stored the distribution of the IP-addresses, assignments of addresses where not stored. To be able to assign addresses the (Mysql) database was manipulated directly. This was successful. Now that the private cloud was running, we continued by creating the connection from this cloud to the Amazon cloud.

5.3 Public cloud (Amazon)

To achieve full cloud transparency, it should be possible to route IP-addresses between the private cloud and the Amazon public cloud. In chapter 4.3 we discussed the possible ways to achieve this. From these one option was selected which we tried to implement.

VPC tunnel

To setup the Amazon virtual private cloud (VPC), two IPsec tunnels needed to be built. Within these tunnels BGP peering was configured. To connect this infrastructure to the local OpenStack installation, the controller node would have to be configured to spawn instances directly to Amazon or an OpenStack node should be placed in the Amazon Cloud. This has been done before and should probably be more successful. Unfortunately, after setting up the tunnel we did not manage to route custom addresses across the BGP session. Only Amazon's private addresses could be acquired through the tunnel.

No tunnel

In order to find out whether a setup would be successful we decided to setup an OpenStack node within the Amazon cloud without a tunnel. Although we were able to install a compute node in the cloud, it was not possible to run instances on this remote node. The necessary networking rules were not passed on to the remote node and no instance data was transmitted to the remote node. Again the same behaviour was observed as with the initial setup according to the reference guide.
The compute node running in the Amazon cloud would be visible from the central node, but instances instructed to run from this node would fail to launch. Although there seemed to be traffic flowing to and from the Amazon compute node, this wasn't nearly enough to facilitate the transmission of an image of 5GB in file size. Unfortunately we were unable to find out what exactly limited the transmission of the images to the Amazon cloud.

In this case we suspect the failure to be caused by the absence of a tunnel. Registration of the node located in the Amazon cloud can be successful since the registration of this node against the central node occurs at layer 3. When an instance needs to be launched however, the nova-network component needs to distribute routes on layer 2. These routes do not propagate across WAN without tunnels.

6 Results

Now that the design and practical setup have been discussed, we will review our findings.

When following the reference design, there are a few processes which should be started through the framework, but we did not manage do get them started initially:

- Bridges need to be generated on all physical nodes
- Routing rules need to be generated, when launching instances
- Instances need to acquire a private IP-address
- Public addresses should be mapped to the private addresses by a network component

After switching to the StackOps distribution, we managed to relieve these issues. However we were still unable to build a hybrid cloud.

7 Conclusion

We started this research by looking into the origins and development of clouds. Then we discussed its possible advantages to the System and Network Engineering education. Based on the requirements of this education we defined our research questions. Following we discussed the different types of clouds used and cloud frameworks available. From these frameworks a selection was made to use in this research. For the selected frameworks a deployment method was devised and executed.

Unfortunately we were unable to answer all our research questions. Therefore we cannot draw a final conclusion on whether or not OpenStack is suitable as a cloud framework for the SNE-education. From the results we have and the research we have done, we can deduce the following answers for our research questions:

Research Questions

- Is it possible to execute SNE-education experiments in the cloud?
  - Since we were unable to complete the cloud setup, we were unable to run any experiments.
- Can connectivity be achieved on a level which allows the creation of custom layer 3 (IP) networks by users of this environment?
  - Yes, private networks can be created and are connected through virtual bridges.
- Can full cloud transparency be achieved?
  - Unfortunately we were unable to test this.

Requirements

- Is it possible to run (virtual) machines?
  - Yes, users can upload and manage virtual machines through client programs which communicate with the nova-API.
- Is it possible to setup (virtual) networks?
  - Yes, VM's can be assigned to private subnets by the use of OpenStack projects.
- Can IP-space be delegated?
  - Yes it is possible, however this requires additional configuration inside VM's. Besides, it is not possible to assign public ip-space directly to VM's, this has to be done through NAT.
- Run Internet services like DNS and e-mail.
  - We were able to run and access an SSH-service, therefore we expect it to be possible to run other services as well. We deduce this expectation from the workings of the nova-network component. Commands controlling this component internally modify routing rules. These routing rules allow or deny services to run. DNS and e-mail depend on the same networking layer as SSH. Therefore they should work also.
- Secure and administer these environments
  - Yes, all virtual machines can be configured with their own firewalls. Besides these local firewalls the nova-network component can control access to groups of VM's through its security groups. OpenStack users can create and modify the security groups of their own VM pools.

8 Discussion

When looking into cloud frameworks the following features are, in my opinion, necessary for successful deployment within the SNE-education:

- The ability to use a custom addressing scheme and thus use the addresses in the public range which are used by the education.
- An interfacing component which enables users to upload their own images

Only after these requirements would be met, this setup could be extended to the Amazon cloud. An extension to the previous setup would then be to:

- Setup an interfacing component towards the Amazon Cloud.
- Create a custom scheduler for load distribution between the public and private cloud.

Although OpenStack supports all requirements to a certain extent, the main issue is the inability to map public addresses directly to VM's. Therefore a solution should be chosen which does support this kind of addressing. It might be the case for virtual infrastructure managers (VIM). Therefore it might be interesting to look into this kind of solution in future research.

9 References

[1] “Activiteiten hogeronderwijsinstellingen,” [Online]. Available: http://www.surfsites.nl/cloud/surf-encloud/. [Accessed 11 2011].

[2] B.Sotomayor, R.S.Montero, I.M.Llorente and I.Foster, “Virtual Infrastructure Management in Private and Hybrid Clouds,” *Internet Computing,* vol. 13, no. 5, pp. 14-22, 2009.

[3] R. Montero, "OpenNebula user mailinglist," 03 07 2009. [Online]. Available: http://lists.opennebula.org/pipermail/users-opennebula.org/2009-July/000551.html. [Accessed 15 12 2011].

[4] "The virtualization API," [Online]. Available: http://libvirt.org/. [Accessed 15 12 2011].

[5] "OpenNebula," [Online]. Available: http://www.opennebula.org/. [Accessed 15 12 2011].

[6] "VMware vSphere," [Online]. Available: http://www.vmware.com/products/vsphere/. [Accessed 14 11 2011].

[7] “VMware vCloud: Architecting a vCloud,” [Online]. Available: http://www.vmware.com/files/pdf/VMware-Architecting-vCloud-WP.pdf. [Accessed 01 12 2011].

[8] D.Nurmi, R.Wolski, C.Grzegorczyk, G.Obertelli, S.Soman, L.Youseff and D.Zagorodnov, "The eucalyptus open-source cloud-computing system," *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid,* pp. 124-131, 2009.

[9] OpenStack, [Online]. Available: http://www.openstack.org. [Accessed 13 10 2011].

[10] "OS3 home," [Online]. Available: https://www.os3.nl/start. [Accessed 31 12 2011].

[11] S. Vaughan-Nichols, "Canonical switches to OpenStack for Ubuntu Linux cloud," 10 05 2011. [Online]. Available: http://www.zdnet.com/blog/open-source/canonical-switches-to-openstack-for-ubuntu-linux-cloud/8875. [Accessed 15 12 2011].

[12] “OpenStack Compute Starter Guide (diablo),” 11 11 2011. [Online]. Available: http://docs.openstack.org/diablo/openstack-compute/starter/openstack-starter-guide-diablo.pdf.

[13] [Online]. Available: https://www.os3.nl/_media/2010-2011/students/sudesh_jethoe/network1.zip. [Accessed 23 12 2011].

[14] J. Robinson, "Kernel Korner: Linux as an Ethernet Bridge," *Linux Journal,* vol. 2005, no. 135, p. 11, 2005.

[15] A. Dayaratna, "Quantifying Cloud Computing Market Share," 06 2011. [Online]. Available: http://cloud-computing-today.com/2011/06/30/quantifying-cloud-computing-market-share-2/. [Accessed 30 12 2011].

[16] “Amazon Virtual Private Cloud,” [Online]. Available: http://aws.amazon.com/vpc/. [Accessed 11 11 2011].

[17] "OpenStack Starter Guide," [Online]. Available: http://docs.openstack.org/diablo/openstack-compute/starter/. [Accessed 17 12 2011].

[18] K. Stewart, “Ubuntu Oneiric Ocelot Release Announcement,” Canonical, 13 10 2011. [Online]. Available: https://wiki.ubuntu.com/OneiricOcelot/ReleaseAnnouncement. [Accessed 01 12 2011].

[19] T. Carrez, "OpenStack Diablo Release Announcement," OpenStack, 22 09 2011. [Online]. Available: https://lists.launchpad.net/openstack/msg04228.html. [Accessed 01 12 2011].

[20] "Vlan mode switch setting," [Online]. Available: https://answers.launchpad.net/nova/+question/157715. [Accessed 21 12 2011].

[21] “StackOps,” [Online]. Available: http://www.stackops.com/. [Accessed 01 12 2011].

[22] H.E.Schaffer, S.F.Averitt, M.I.Hoit, A. Peeler, E. Sills and M.A.Vouk, “NCSU’s VirtualComputing Lab: A Cloud Computing Solution,” *Computer,* vol. 42, no. 7, pp. 94-97, 2009.

[23] “Amazon Web Services: Developer Tools,” [Online]. Available: http://aws.amazon.com/developertools. [Accessed 09 11 2011].

[24] M.A.Vouk, “Cloud Computing - Issues, Research and Implementations,” *Journal of Computing and Information Technology,* vol. 16, no. 4, pp. 235-246, 2008.

[25] “Amazon Web Services,” [Online]. Available: http://aws.amazon.com. [Accessed 01 12 2011].

[26] “Rackspace API,” [Online]. Available: http://www.rackspace.com/cloud/cloud_hosting_products/servers/api/. [Accessed 09 11 2011].

[27] D. Parrilla, "Dual node deployment," StackOps, 17 11 2011. [Online]. Available: http://docs.stackops.org/display/doc03/Dual+node+deployment. [Accessed 01 12 2011].

[28] "cloud_image," [Online]. Available: http://www.cloudxperience.nl/cloudxperience/blog/wp-content/uploads/2011/02/cloudcomputing.png. [Accessed 31 12 2011].

Appendix

I. Initial configuration according to reference design

OpenStack configuration files

Initial configuration of the central node, with VLAN-networking:

```
# enable VLAN networking mode:
--network_manager=nova.network.manager.VlanManager
--dhcpbridge_flagfile=/etc/nova/nova.conf
--dhcpbridge=/usr/bin/nova-dhcpbridge
--public_interface=eth0
# interface for cloud network:
--vlan_interface=br100
# public IP of central node, used for NAT of public IP's of instances:
--routing_source_ip=145.100.104.36
# global scope cloud network:
--fixed_range=192.168.0.0/16
# size of user private subnets in addresses including network and host
address:
--network_size=8
--iscsi_ip_prefix=192.168.
```

Initial configuration of the compute node, with VLAN-networking:

```
# network flat:
--network_manager=nova.network.manager.FlatManager
# Block of IP addresses that are fixed IPs (private range)
--fixed_range=192.168.0.0/16
# number of addresses in each private subnet
--network_size=24
# public ip of server running nova-network (used for sNAT when no public
addr)
--routing_source_ip=145.100.104.36
--flat_network_bridge=br100
# interface to bind private network to:
--flat_interface=eth1
--flat_network_dhcp_start=192.168.1.2

# network dhcp:
#--network_manager=nova.network.manager.FlatDHCPManager
#--flat_injected=False
#--flat_network_dhcp_start=145.100.106.161
#--public_interface=eth0
#--flat_network_bridge=br100
#--flat_interface=eth2
```

Addressing

Central Node:

```
/etc/network/interfaces
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
 auto eth0
 iface eth0 inet dhcp

# OpenStack reference design:
#auto eth1
#iface eth1 inet static
#       address 192.168.3.1
#       netmask 255.255.255.0
#       network 192.168.3.0
#       broadcast 192.168.3.255

# Manual adjustment:
auto br100
iface br100 inet dhcp
    bridge_ports        eth1
    bridge_stp           off
    bridge_maxwait       0
    bridge_fd            0
```

Compute Node:

```
\etc\network\interfaces
# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet dhcp

# OpenStack reference design:
#auto eth1
#iface eth1 inet static
#               address 192.168.3.2
#               netmask 255.255.255.0
#               network 192.168.3.0
#               broadcast
192.168.3.255

# Manual adjustment:
auto br100
iface br100 inet dhcp
    bridge_ports        eth1
    bridge_stp           off
    bridge_maxwait   0
    bridge_fd            0
```

According to the OpenStack reference design, the bridges should be created automatically by the networking component of the framework, this however, was not the case and therefore the bridges were needed to be setup manually in the networking configuration.
The observed error in the log files:

```
2011-11-20 12:15:51,496 ERROR nova.exception [-] Uncaught exception
2011-11-20 12:15:51,497 ERROR nova.compute.manager [-] Instance '8' failed
to spawn. Is virtualization enabled in the BIOS? Details: Failed to add tap
interface to bridge
 'br100': No such device
```

II. Secondary setup using StackOps

Interfaces configuration on Central Node, using StackOps:

```
root@statler:/etc/network# cat interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 145.100.104.36
        netmask 255.255.255.224
        network 145.100.104.32
        broadcast 145.100.104.63
        gateway 145.100.104.33
        # dns-* options are implemented by the resolvconf package, if installed
        dns-nameservers 8.8.8.8 8.8.4.4
        dns-search studlab.os3.nl
# storage gateway
auto eth1
iface eth1 inet static
        address 192.168.0.1
        netmask 255.255.255.240
        broadcast 192.168.0.15
```

Interfaces configuration on Compute Node, using StackOps:

```
root@waldorf:/etc/network# cat interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
        address 145.100.104.37
        netmask 255.255.255.224
        network 145.100.104.32
        broadcast 145.100.104.63
        gateway 145.100.104.33
        # dns-* options are implemented by the resolvconf package, if installed
        dns-nameservers 8.8.8.8 8.8.4.4
        dns-search studlab.os3.nl

# storage tunnel
auto eth1
iface eth1 inet static
        address 192.168.0.2
        netmask 255.255.255.240
        broadcast 192.168.0.15
```

Firewall configuration

On Central Node:

```
root@statler:/home/svjethoe# iptables -t nat -L
**snip**
#####################################################
Chain nova-network-POSTROUTING (1 references)
target     prot opt source               destination
ACCEPT     all  --  10.0.0.0/16          10.128.0.0/24
ACCEPT     all  --  10.0.0.0/16          10.0.0.0/16
Chain nova-network-floating-snat (1 references)
target     prot opt source               destination
Chain nova-network-snat (1 references)
target     prot opt source               destination
nova-network-floating-snat  all  --  anywhere             anywhere
SNAT       all  --  10.0.0.0/16          anywhere
to:145.100.104.36
#####################################################
Chain nova-postrouting-bottom (1 references)
target     prot opt source               destination
nova-network-snat  all  --  anywhere             anywhere
```

In the used setup, all addresses in nova-network were part of the preconfigured 10.0.0.0/16 network.

For the central node the connected interface to this network was **br100**, this is a virtual bridge attached to physical interface **eth1**:

```
root@statler:/etc/nova# ip addr (ifconfig does not show all attached
addresses)
7: eth0:
    inet 145.100.104.36/27 brd 145.100.104.63 scope global eth0

9: br100:
    inet 10.0.0.1/25 brd 10.0.0.127 scope global br100
    inet 192.168.0.1/28 brd 192.168.0.15 scope global br100

root@statler:/etc/nova# brctl show
bridge name     bridge id               STP enabled     interfaces
br100           8000.b8ac6f80574b       no              eth1

root@statler:/etc/nova# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use
Iface
192.168.0.0     *               255.255.255.240 U     0      0        0
br100
localnet        *               255.255.255.224 U     0      0        0 eth0
10.0.0.0        *               255.255.255.128 U     0      0        0
br100
default         router.studlab. 0.0.0.0         UG    100    0        0 eth0
```

On the compute node:

```
root@waldorf:/home/svjethoe# ip addr
6: eth0:
    inet 145.100.104.37/27 brd 145.100.104.63 scope global eth0

internal cloud interface:
7: eth1:
    inet 192.168.0.2/28 brd 192.168.0.15 scope global eth1

root@waldorf:/home/svjethoe# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use
Iface
192.168.0.0     *               255.255.255.240 U     0      0        0 eth1
localnet        *               255.255.255.224 U     0      0        0 eth0
default         router.studlab. 0.0.0.0         UG    100    0        0 eth0
```

Now when an instance is started the following can be observed:

```
root@waldorf:/home/svjethoe# euca-describe-images
IMAGE   ami-46ccba64    db6/debian6-base.img.manifest.xml
available       public          x86_64      machine

root@waldorf:/home/svjethoe# euca-run-instances ami-46ccba64
RESERVATION     r-aws7z4p8      sne-cloud       default
INSTANCE        i-00000012      ami-46ccba64                    scheduling
None (sne-cloud, None)  0  m1.small 2011-12-14T15:41:31Z    unknown zone

root@waldorf:/home/svjethoe# euca-describe-instances
RESERVATION     r-aws7z4p8      sne-cloud       default
INSTANCE        i-00000012      ami-46ccba64    145.100.106.162 10.0.0.2
running None (sne-cloud, waldorf)   0               m1.small        2011-12-
14T15:41:31Z    nova
```

III. Encountered errors during initial setup

During installation of the Swift component, which is used for storage, the following error occurred:

```
Exception: Could not create account AUTH_admin for user admin:admin
```

This error is logged in the bugtracking system of OpenStack under:
https://answers.launchpad.net/swift/+question/163331

After checking that all entries were set correct in the database and all passwords were configured correctly, I found out that the problem actually lied with the rights on certain folders, which were not configured to be accessible by the Swift user account on the central node.

When running the following command to fix these rights, swift was finally set up correctly and I was able to proceed with installing OpenStack.

```
chown -R swift:swift /srv/[1-4]/node/*
```

After setting up the storage component, also the imaging service (Glance) needed to be set up correctly. Here another issue was encountered. I was unable to upload images. When uploading a VM from a desktop pc, with the following command:

```
glance add name="Ubuntu 11" is_public=True < oneiric-server-cloudimg-amd64.tar.gz
```

This error is encountered:

```
ClientConnectionError: Unable to connect to server. Got error: [Errno 111] ECONNREFUSED
```

The error indicates that the desktop is unable to authenticate against the glance server. This is after authentication has been setup and “uploading” images to the object store does work when executed directly on the server running the glance component.