


Technical University of Munich
Department of Informatics

Bachelor's Thesis in Informatics

Measuring and Modeling the Performance of OpenStack

Adrian Weis

Technical University of Munich

Department of Informatics

Bachelor's Thesis in Informatics

Measuring and Modeling the Performance of OpenStack

Vermessung und Modellierung der Performanz von OpenStack

Author	Adrian Weis
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Daniel Raumer, Sebastian Gallenmüller
Submission date	October 15, 2016

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Garching b. München,

Signature

Abstract

A typical use case of cloud computing is the deployment of scalable web servers. OpenStack is a widely used open source cloud computing service. This thesis shows a way how to implement an OpenStack environment capable of providing the resources for a web server. Two different network solutions are presented. One implements OpenStack's legacy networking service Nova-networking, while the other one implements Neutron, a networking service, that is about to replace Nova-networking in the future. The network performance of a single node setup as well as a multi node setup is measured. It is found, that for a single node setup Nova-networking performs better than Neutron. The results suggest that Neutron needs further improvements to efficiently replace Nova-networking.

Zusammenfassung

Eine typische Anwendung für Cloud Computing ist die Bereitstellung skalierbarer Webserver. OpenStack ist eine weit verbreitete open source Lösung für Cloud Computing. In dieser Arbeit soll der Aufbau einer OpenStack Umgebung beschrieben werden, die die Ressourcen für einen Webserver zur Verfügung stellt. Dabei werden zwei verschiedene Netzwerklösungen vorgestellt. Die eine implementiert den ursrünglichen OpenStack Netzwerkservice Nova-networking, die andere Neutron, einen Netzwerkservice, der in naher Zukunft Nova-networking ersetzen soll. Die Netzwerkperformance von einem Setup mit lediglich einem Computing Knoten wird ebenso vermessen wie die Performance eines Setups mit mehreren Computing Knoten. Es wird gezeigt, dass in einem Setup mit nur einem Knoten die Performanz von Nova-networking deutlich besser ist als die von Neutron. Die Ergebnisse sprechen dafür, dass Neutron weitere Verbesserungen oder Erweiterungen benötigt, um Nova-networking effizient ersetzen zu können.

Contents

1 Introduction 1
1.1 Goal of the Thesis . . . 2
1.2 Outline . . . 2

2 Background 3
2.1 Related work . . . 3
2.2 Cloud Computing . . . 4
2.3 Development of OpenStack . . . 4
2.4 OpenStack vs. alternative products . . . 5
2.5 How does OpenStack work? . . . 6

3 Setup 11
3.1 Physical Setup . . . 11
3.2 Identity Service . . . 12
3.3 Image Service . . . 13
3.4 Compute and network service . . . 15
3.5 Compute Node . . . 16
3.6 Instances . . . 18

4 Measurements 23
4.1 Tool selection . . . 23
4.2 Results . . . 24

5 Conclusion 30
5.1 Future work . . . 31

Bibliography 32

List of Figures

2.1 OpenStack Conceptual architecture [1] 7

3.1 Setup main topology. 12
3.2 Keystone Identity service. 14
3.3 Setup on the controller node extended by the Glance Image Service. . . 15
3.4 Complete setup on the controller node. 17
3.5 Compute node and its services. 18
3.6 VMs in the Nova measurement setup. 20
3.7 VMs in the Neutron measurement setup. 22

4.1 Latency measured with 20 connections open from same node. Diagram in logarithmic scale. 24
4.2 Data throughput with 20 connections open from same node. Diagram in logarithmic scale. 25
4.3 Measurement with 20 (darkblue) vs 100 (orange) open connections on same host. Diagram in linear scale. 26
4.4 Measurement with 20 connections on same (darkblue) vs remote (light-blue) host. 27
4.5 Neutron (green) vs Nova-networking (darkblue) with 20 open connections. 28
4.6 Measurement Neutron remote (purple) and same host (green). 29

List of Tables

2.1 OpenStack Releases [2]. 5

Chapter 1

Introduction

The very basis of our investigation shall be the following scenario: imagine an enterprise which offers its services via web servers. Over time the traffic on their webpage changes. Be it because of day and night, or because the content is event based like a livestream.

Now in times of high traffic, when there are many connections to be served, it is in this enterprises interest to provide enough resources to fulfill the great majority of incoming requests, so that the user experience is generally positive. In the "low time", when the number of requests is significantly lower, the resources would not be needed.

So how can this uneven allocation of resources be solved without paying to much for unused resources or losing users due to bad experience from many unfulfilled requests, caused by insufficient resources? In this case dynamic allocation and horizontally scalability of resources is the key to handle these issues.

This is the point where cloud computing comes into play. Cloud computing allows another enterprise to provide resources for rent. And depending on the development of traffic on the web server one can either stock up in resources to serve the requests or reduce the rented computing power to save money, when there are only few open connections.

All that is a nice idea, but how is this implemented? The cloud computing enterprise uses a cloud computing service to provide resources to its clients. There are several different of these services. Commercial ones, like e.g. Amazon own cloud computing provider EC2 and there are open source solutions, such as OpenStack and OpenNebula.

So from this basic scenario we want to establish a realistic setup and measure its performance under different conditions. Our base Setup is a web server, which is operated by the resources provided by an OpenStack cloud computing service setup.

1.1 Goal of the Thesis

The goal of this thesis is to get an overview over the capabilities of the performance of OpenStack when providing resources for a webserver. In this thesis we will not focus on a single component of OpenStack, but on measurements that cover a sample setup of an OpenStack environment.

1.2 Outline

First of all we will define Cloud computing in Section 2.2. Then we have a deeper look into what OpenStack specifically is, how it works, and how its different sub services work together to form the providing cloud service in Section 2.5. After this we will introduce our specific measurement setup in Section 3.1. Finally we will present the different results of our measurements in Section 4.2.

Chapter 2

Background

2.1 Related work

Most of the work on cloud computing are comparisons between different services. Wen et al. [3] compared Open Stack to OpenNebula in terms of architecture, security compatibility to other services and ease of use. It is found, that OpenStack offers a wider range of Interfaces to other services such as to the EC2 cloud computing solution by Amazon, and also offers more flexible network solutions, while OpenNebula is more focused on usability of the VM environment it provides. Li et al. [4] compared OpenStack to OpenNebula and Nimbus (which are all Cloud Computing services operating with KVM virtualization) in terms of computing performance. As a performance test the High Performance Computing Challenge benchmark suite was used. It could be shown, that OpenStack reached better results than both its competitors. Callegati et al. [5] compared the performance of especially the virtual network capabilities of OpenStack to virtual networks created solely with Open vSwitch and the Linux bridge. In their work they could make out the Linux bridge as the performance limiting component, while the Open vSwitch was not an influential part to the performance. This is an important finding, as the Open vSwitch contributes many essential features to multi tenant cloud environments, as Goldberg et al. [6] analyzed. Callegati et al. [7] also showed, that the virtual network environment created by OpenStack has a limiting effect on the networking performance that is independent from the hardware on which the environment is hosted. Their results suggest that the performance loss correlates to an increasing number of tenants in the virtual network environment.

2.2 Cloud Computing

There are different options how cloud computing systems provide resources to their clients. One way would be to provide the client with a software interface. Very few functionalities are provided by this interface, and the client has no insights into the implementation, nor can he add functionalities or manage the resources underneath. This concept is called "Software as a Service" (SaaS) [8].

A hybrid version is to provide the client a development platform, on which he can customize e.g. an application from a back end. This concept is called "Platform as a Service" (PaaS) [8].

A basic service cloud computing can provide is to give the client access to the infrastructure. This can e.g. be networking, CPU or memory. Often this is done by providing a VM with an installed Operating system. The client can operate on this machine as if he would have access to a PC and manage the provided resources without limitations. Due to the VM nature of the provided resources, the client has no knowledge about the underlying cloud computing infrastructure. This concept is called "Infrastructure as a Service" (IaaS) [8]

This thesis will only cover systems that fall under the IaaS category.

2.3 Development of OpenStack

OpenStack is a service located in the IaaS realm of cloud computing. It has been founded in the year 2010 by the joint efforts of NASA and Rackspace [9]. Both organizations have been working on solutions for an alternative to Amazon's commercial cloud computing service Amazon Elastic Compute Cloud (EC2). Both started from different points and where working on different parts of the cloud computing service, so when they found out of each other's efforts they decided to bundle it [8]. The outcome was OpenStack, which was from then on published under the Apache 2.0 license. Effectively this guaranteed a free use of the product and its source code [10].

OpenStack has since then undergone many version updates. As seen in Table 2.1 the most recent to this point is Mitaka [2].

The fact that the OpenStack source code is published under this rather permissive license contributed a lot to its great success and spread. Today it is the software of choice for many cloud computing solutions. An article from networkworld.com states e.g. Dell, HP, Cisco and IBM amongst the big users of this cloud computing service [11].

Release	Release Date
Queens	TBD
Pike	TBD
Ocata	2017-02-23 (planned)
Newton	2016-10-06 (planned)
Mitaka	2016-04-07
Liberty	2015-10-15
Kilo	2015-04-30
Juno	2014-10-16
Icehouse	2014-04-17
Havana	2013-10-17
Grizzly	2013-04-04
Folsom	2012-09-27
Essex	2012-04-05
Diablo	2011-09-22
Cactus	2011-04-15
Bexar	2011-02-03
Austin	2010-10-21

Table 2.1: OpenStack Releases [2].

2.4 OpenStack vs. alternative products

In the race on the market there are other products as well. Earlier we mentioned Amazons EC2, which is a commercial service. It has been published as a beta version in 2006 [12] and gone live in 2008 [13]. Amazon itself is using this service for its well known commercial selling platform [14]. As a product from a major player in the online business it is highly trusted by users, and there is no doubt many users do pay for this service in exchange for easy and externally maintained cloud computing experience. However, a company with a huge name as Amazon is all over the news if one of its products hits controversial ground like when in 2010 Amazon decided to no longer host the Wikileaks platform on its hosting service [15].

Another service in the cloud computing race is OpenNebula. It is one of the earliest modern cloud computing platforms and persists until this day. It was developed from 2005 to 2006 and was finally released in 2008 [16]. Like the OpenStack software it is released under the Apache license [17]. It came more into focus in 2014 when Microsoft announced a collaboration with OpenNebula [18]. It has a extended portfolio of companies using its services, many of them, such as Blackberry, Unity or the ESA are widely known [19], but its impact remains smaller then the one of OpenStack, due to the little number of big players, such as the ones OpenStack counts to their users. Interesting is, that NASA despite having co-founded OpenStack also uses OpenNebula [19].

There are many more cloud computing platforms out there, but mentioning to many of them is futile, as the markets are flexible, and though the demand in cloud computing will probably increase over the years to come, these enterprises will come and go.

So what are the arguments for the use of OpenStack? First of all, in comparison to all commercial platforms it is free, so the customer does not need to pay for the platform software. Especially for small firms, non profit organizations or public institutions this is a great advantage, as it leaves expenses at a minimum. Another advantage is, that it is open source, which means in terms of the Apache 2.0 license, that modifications can be made to the source code, to better fit the customer. The performance of OpenStack has proven superior to its competitors: Chunyan Li et al. [4] used the High Performance Computing Challenge benchmarking suit to compare the performance of open source cloud computing services. The benchmark was performed on Nimbus, OpenNebula and OpenStack. OpenStack showed the highest performance in this comparison.

2.5 How does OpenStack work?

OpenStack is a multiple host setup which provides users ressources in the form of VMs. One of the hosts is the designated controller host. The service that controls the distribution of ressources to the users runs on this node. Apart from this authentication issues are handled, and VM images are stored here. Also a database is located here, which stores user data, as well as data used by the several services that run within the OpenStack environment. Depending on the networking setup, the networking components may also be located here. There are setups where there is a separate networking node. The actual ressources provided reside on special compute nodes

OpenStack is constructed out of different underlying services. These act together to provide the cloud computing service. An Overview of the services can be seen in Figure 2.1. In the following we will give a description for all of the services:

Keystone is the so called identity service. It handles all the authentication issues and manages permissions. For this job it divides the users into different groups. Every client has a User, Tenant and Role association which gives him certain privileges in the use of resources. When a client wants to access a service of OpenStack, he first has to authenticate to Keystone. This can be committed by either user name and password authentication or by signed certificates. After the authentication Keystone grants the user access to all functionalities, that lie within its permissions. This is implemented via a token system. By authenticating to Keystone, the client is given out a timestamped token, which expires after a while. The validity of this token is checked back with Keystone by all other services the client wishes to utilize. After expiring, a new token is given out to the client if he wishes to proceed to work in the OpenStack environment [8].

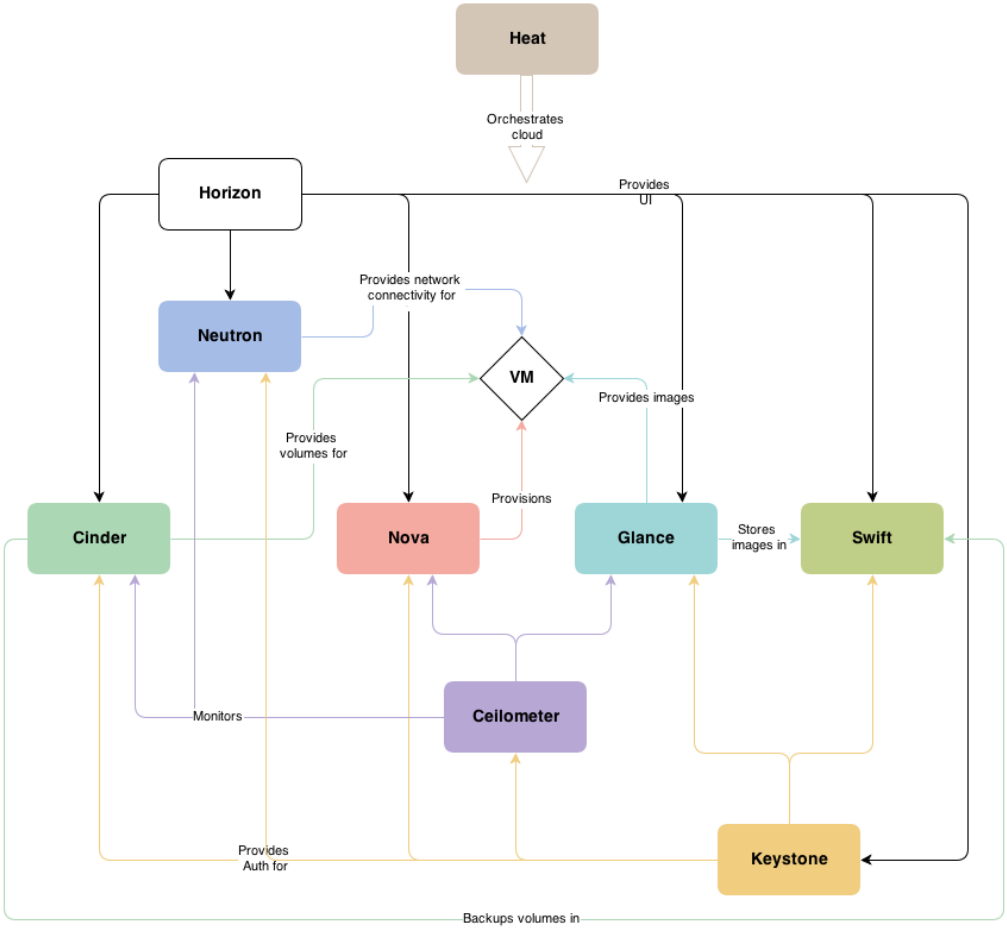

Figure 2.1: OpenStack Conceptual architecture [1]

Another task of Keystone is the management of the back end APIs of the services. To communicate with an OpenStack service every user and also other services must go through Keystone to achieve the endpoint URL over which the communication is established. All OpenStack services must therefore be registered with Keystone. As it is also a service, even Keystone needs to be registered with itself. This registration contains the creation of the endpoint in Keystone and the creation of a respective database entry [20].

Glance is the Image Service. Its main purpose is to store images for the VMs that are provided to the client. Glance can load local copies of images or directly load and store new image files from their web source. The managing system allows to give names and IDs to the images for a better overview. Also Glance works together with the authentication service Keystone to define security rules for the usage of images. It is possible to disallow or allow the use of certain images to certain roles in a tenant [21].

All these meta information is stored in the mysql database. In the Glance service it can also be determined where the images are stored. Usually, this is just a dedicated directory on the controller node, where this service is running. However, it could also be combined with the object storage service Swift, which makes it possible to distribute the storage of the image away from the controller node. This is specially useful when dealing with a great amount of images, or when images need to be redundantly stored. Glance can store and manage images of different file formats (qcow2, raw, vhd, vmdk, vdi, iso, aki, ari, and ami) [22] and container formats (bare, ovf, aki, ari and ami) [22]. Container formats determine how the meta data of an image file is passed on.

Glance also handles the storage of snapshot images from VMs. In this concept it is a good idea to combine it with the storage service, as the number and size of snapshot files can grow very fast in production environments [8].

Another helpful functionality of Glance is called file injection. This allows Glance to attach certain files to a VM, when booting. This way you can e.g. establish a randomly created password for the root user to increase the security of freshly booted machines, and ensure, that it can not be highjacked until the user it rightfully belongs to, has gotten access and the chance to change the password. This is also useful if your clients of a certain tenant are employees of the same company. This way you can equip their freshly started VMs with e.g. company Software or important data. Another important aspect of file injection is the possibility to inject configuration files, like e.g. network configuration, so that the configurations take place in the first boot, and the VM is instantly ready to use [8].

Swift is the object storage service of OpenStack. The non hierarchical organization gives way to automatize the access to the saved data. In the case of Swift this data consists of the images and snapshots of the VMs. The object storage is horizontally scalable and therefore suitable for setups with a huge number of VMs to deploy and clients to serve. A disadvantage is that to fulfill its redundancy as proposed, the Swift setup contains out of seven different hosts of which two serve as proxys and the rest is actual storage [8]. This makes the use of swift only economically applicable in large scale setups.

Cinder is also a storage service. In contrast to Swift it is a block storage, which means the storage is provided as Volumes. Another difference is its use. It is reserved for the dynamic disk allocation for the VMs [8].

Nova is the computing service. It handles the actual deployment of the VMs and the resource allocation among VMs on the hosts. It is composed of several sub services. Of these the *nova-api* is the most important. Over this interface all requests are processed and access to resources is given. The *nova-scheduler* is also important, as it processes the deployment of new VMs on the compute nodes. The scheduler hereby performs several balancing mechanisms between the available hosts so that the deployment

brings optimal resource usage, while fulfilling requirements of the launched images. Nova also handles the communication necessary for multi node systems, so that the OpenStack users only interferes with the diversity of compute hosts if he wishes to. (For example when certain VMs shall be deployed together on a single node to boost performance. Which is possible as shown in the results). The fact that Nova can handle several compute nodes by itself contributes to the horizontal scalability, as the addition of new hosts is as "easy" as configuring them and let them register to Nova via the messaging API so the scheduler can access them. It has a built in VNC proxy that allows remote access over VNC consoles. There is a certain compatibility given to other systems, as Nova has implemented its API to fit systems like e.g. Amazons EC2. Nova is known to be very fault tolerant due to the nature of its architecture. It is built around processes which run isolated from each other as much as possible to stop exponential growth of faults [8].

Nova also has a networking service. It is a lightweight networking service offering a flat network bridge, a DHCP server and also VLAN functionalities. [23]

Neutron is an OpenStack standalone networking service. With the Folsom release in 2012 the Nova-networking service was strived to be superseded by the Quantum-networking service. Just one year later it was renamed due to a copyright issue [24]. This now called Neutron networking offers a lot of differentiated network functionalities and is often run on a dedicated network-node. In light-weight solutions Nova-networking is often still preferred.

Horizon is a graphical interface designed to manage the cloud service as a whole. Many administrative tasks can be done here. This contains managing images and networks, or handling user related issues like creating new users or managing security configurations among the user base. But also the users can use the dashboard to e.g. boot VMs or access statistics and overview over used resources. Another important feature Horizon has to offer is is a Virtual Network Computing (VNC) client, to which the input/output of the VM can be redirected and even the kernel logs can be monitored e.g. for debugging [8].

Ceilometer is also called "Measurement Service" [8]. It measures the resource usage and maps it to the different users. The data is only collected and not processed. This lies in the hand of the cloud administration. Ceilometer is often used for monitoring commercial OpenStack environments and plays an important role in the billing of the client [8].

Heat is the orchestration service. It organizes the processes of all services to provide the possibility to launch a cloud environment consisting of multiple VMs by making use of templates. These templates act as configuration files and contain all the information necessary to launch the right VMs equiped with initial data files and configurations (provided by file injection). The templates can be used over and over again, and can be adapted to new circumstances with a few alternations of their configuration [8].

Not all of these service will be used in the measurement setup. The important ones are **Keystone**, **Glance**, **Neutron** and **Nova**.

Apart from the OpenStack services there are several auxiliary services:

mysql is used as a database to store important data. It is used to store information about user privileges and security groups. Also it contains the metadata about VM images and their storage place on the controller node.

Rabbitmq is used as a messaging server between the different services. It is a lightweight but very flexible messaging broker. It supports a large number of messaging protocols and provides several features such as mirroring of queues for high availability applications, tracing for easier debugging or the possibility to write and embed own plugins [25].

Chapter 3

Setup

3.1 Physical Setup

We set up the nodes on a system of servers connected via a HP-3800-48G4SFPP VLAN switch over 1 Gbit links. The following are the properties of the hosting servers:

Riga:

- CPU: Intel(R) Xeon(R) CPU E31230 @ 3.20 GHz
- Number of CPUs: 1
- Memory: 16 GB
- Mainboard: X9SCL/X9SCM
- NICs: 4x 82576

Tartu:

- CPU: Intel(R) Xeon(R) CPU E3-1230 V2 @ 3.30 GHz
- Number of CPUs: 1
- Memory: 16 GB
- Mainboard: X9SCL/X9SCM
- NICs: 2x Intel 82599ES (1x X520-SR2)

Vilnius:

- CPU: Intel(R) Xeon(R) CPU E31230 @ 3.20 GHz
- Number of CPUs: 1
- Memory: 16 GB
- Mainboard: X9SCL/X9SCM
- NICs: 4x 82576

Our controller node (and in case of the Neutron setup also the networking node) is located on the Riga server. Tartu and Vilnius act as compute nodes. Figure 3.1 shows which host will take on which part in the setup.

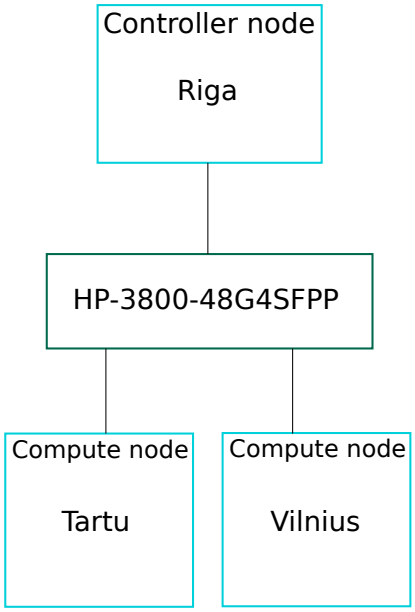

Figure 3.1: Setup main topology.

3.2 Identity Service

As mentioned before, the starting point in every OpenStack cloud is the identity service Keystone. To get this running, the system service we need to install is called *keystone*.

Keystone is the authentication service and handles roles, tenants and users. First lets have a look at what all these entities are:

User: A user is one specific person that logs into the OpenStack system. A user can be associated to one or more tenants and fills one specific role within this tenant [8].

Tenant: A tenant or project is the union of a specific group of users which belong together. This can be for example the entirety of VMs rented by a specific company. Depending on the security role allocations these users can access selected VMs associated to this tenant. A tenant needs at least one user fulfilling the admin role. The admin has access to all VMs and can create and manage diverse roles and the security rules associated with them. Direct access to VMs of other tenants is not possible from a certain tenant, while the connection between VMs of different tenants is possible like e.g. the connection between hosts in a network. This makes indirect access e.g. via ssh possible [8].

Role: Every user within a tenant or project fulfills a certain role. With these roles come different levels of security. These can be created and managed by the Admin role and contain features like the right to see or access different VMs within the tenant. There has to be at least one user in the admin role per tenant [8].

In our specific setup an admin user within an admin tenant is created and associated the admin role. This account is used for the deployment of VMs, but also to perform administrative tasks on OpenStack.

Also a service tenant is created, in which all the OpenStack services are registered. This is necessary, as all the OpenStack services need to be identifiable via Keystone as a role in a certain tenant to be able to communicate with each other. This time the Keystone service itself does not have to be added to this tenant, as it is the registration entity itself. So as for now this tenant is empty, it gains more and more members, as the services are added one by one.

Nonetheless we need to create and register a Keystone API endpoint over which Keystone will be able to communicate with other services. The APIs are the communication interfaces of the OpenStack services. As Keystone needs to communicate with the other services we create an endpoint for it here. As with the users of the service tenant also the number of endpoints increases, as we add new services to our OpenStack environment.

The information about the created tenants and users, and the roles we associated to them, as well as the metadata about the created endpoints is stored in the mysql database. Figure 3.2 summarizes the components of Keystone which have been added.

3.3 Image Service

The next service that comes into play is Glance. Glance is the image service of the OpenStack installation. In our case Glance will save the images in a dedicated directory directly on the controller node. This is especially possible, because we only have two images stored. One is a Cirros minimal setup used mainly for test purposes, and the

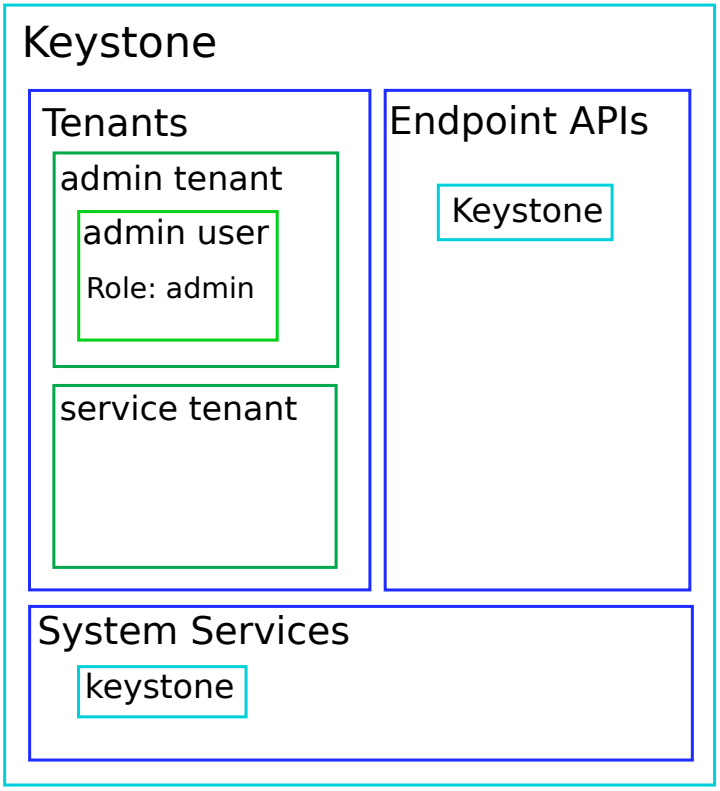

Figure 3.2: Keystone Identity service.

production Operating System (Ubuntu 14.04 amdx64) for the VMs of the measurement setup.

While the imagefiles themselves are stored in a directory on the controller node, the path to this directory is stored in the mysql database. (This way the path could also be altered to point to a Swift storage instead of a local directory.)

The respective system services that have to be installed are *glance* and *python-glanceclient*. Unlike Keystone, Glance consists of several system services.

Now the service has to be registered to Keystone. To achieve this we create a database entry for Glance. Then we create a Glance user within the service tenant of Keystone to settle privileges and security issues. And finally we create an API endpoint, so that Glance can have an ear on what is going on in the OpenStack environment, and that it can react when its services of providing a VM image are needed.

After the successfull installation of Glance we need to achieve and store some images. For this we can simply call the *create* command of the Glance service with respective parameters. This command creates a database entry for the imagefile and stores it in a dedicated directory. Its parameters contain the following information: the original location from where to aquire the imagefile (can be a weblink or a locale save), the

name under which the image can be referred to, the disk format as well as the container format, information about the security privileges (in general whether the image can be used by non-admin users) and other optional parameters [26].

So by now we have expanded our service environment by yet another component: Glance the Image Service. As can be seen in Figure 3.3 also Keystone has gained more entries.

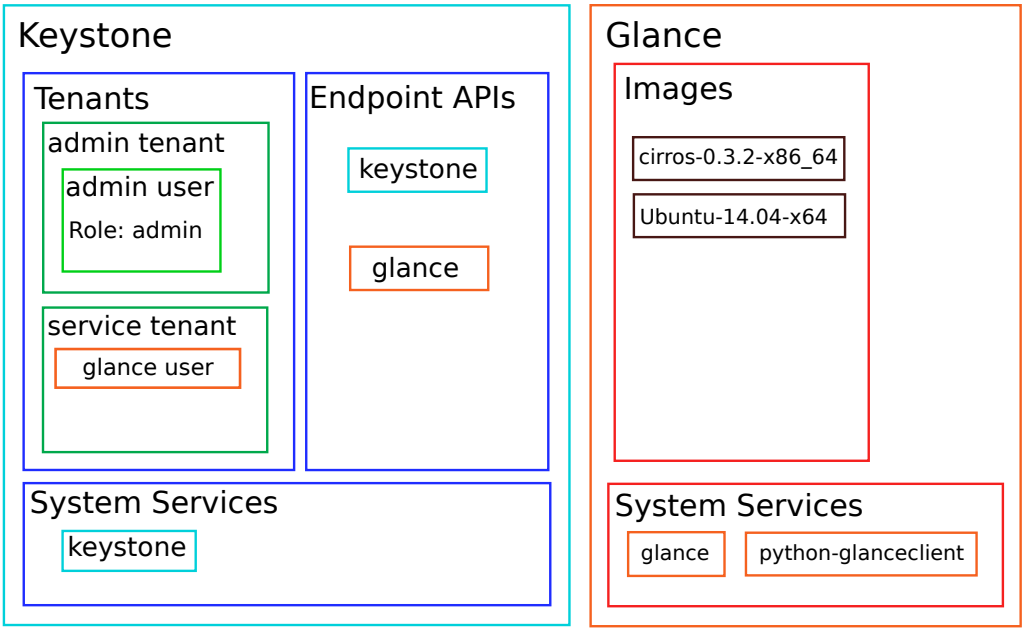

Figure 3.3: Setup on the controller node extended by the Glance Image Service.

3.4 Compute and network service

The last thing left to be configured on the controller node for now is Nova. There are two tasks that Nova has to fulfill. One of them is to run the computing service, which is responsible for the balancing and distribution of the resources provided by the compute hosts. The other is to provide basic network functionalities.

For the resources part we have to install the services *nova-api*, *nova-cert*, *nova-conductor*, *nova-consoleauth*, *nova-novncproxy*, *nova-scheduler*, and *python-novaclient*. Before we can start to use the services we need to establish a connection between the different services, so we start to configure the rabbitmq Messaging service, the database connection and the VNC functionalities in the Nova configuration files.

Nova is an OpenStack service as well and so a database entry and a Keystone API endpoint have to be created and registered. To finish the installation we need to configure

the use of the credentials when identifying to Keystone. This is also done in the Nova configuration files. And also we create a Nova user in the service tenant. The only thing that is left to do is restarting all the Nova services, and our environment got extended by our computing service.

The second part of Nova is the networking service. The networking is mainly done on the compute nodes, when Nova-networking is in use, so there is relatively little configuration work on the Nova config files. After restarting the Nova services again the configuration of the controller node is completed. Finally, on our controller we can find the following environment shown in Figure 3.4.

3.5 Compute Node

On the compute node we have to install the *python-mysqldb* service to be able to work with the mysql database, e.g. when requesting an image via Glance. The next thing to install is *nova-compute-kvm*. This service is handling the hypervisor, which ultimately manages the resources, that the VM has access to, and also manages the separation of VM and underlying hardware. In our case this will be KVM. In some other cases QEMU is used as the only hypervisor. This is e.g. the case, when the kernel version does not allow hardware acceleration done by the KVM module [27]. After this installation, the use of *rabbitmq* and *mysql* has to be configured correctly and also the remote access via VNC has to be set up. The specification of the Glance host (which in our case will be the controller node) and the authorization instance (which is Keystone) finish the Nova-compute setup on the compute node.

What is left to do is to configure the compute node for network access. The easier part is the installation of the two packages we need for this to work: *nova-network* and *nova-api-metadata*. Then the networking parameters have to be configured. After having defined the api responsible for the communication with other services, we define the used security-group and the firewall driver. Nova-network comes with its own built in firewall, which works via IPtables. We then select the flat DHCP manager as the network manager to delegate the task to give new instances IP addresses to an DHCP server. In our specific setup this DHCP server has to be configured to only apply to the instances addresses within the realm of the OpenStack environment. Otherwise this DHCP server would conflict with the one installed in the testbed. Nova now has to be configured for a multiple host environment. This lets the *nova-network* service listen to the same service on other compute nodes. And finally we define a public interface to gain internet access for the VMs. After having restarted the freshly installed services, we have finished the configuration of the compute node. You can see an overview of the services in Figure 3.5.

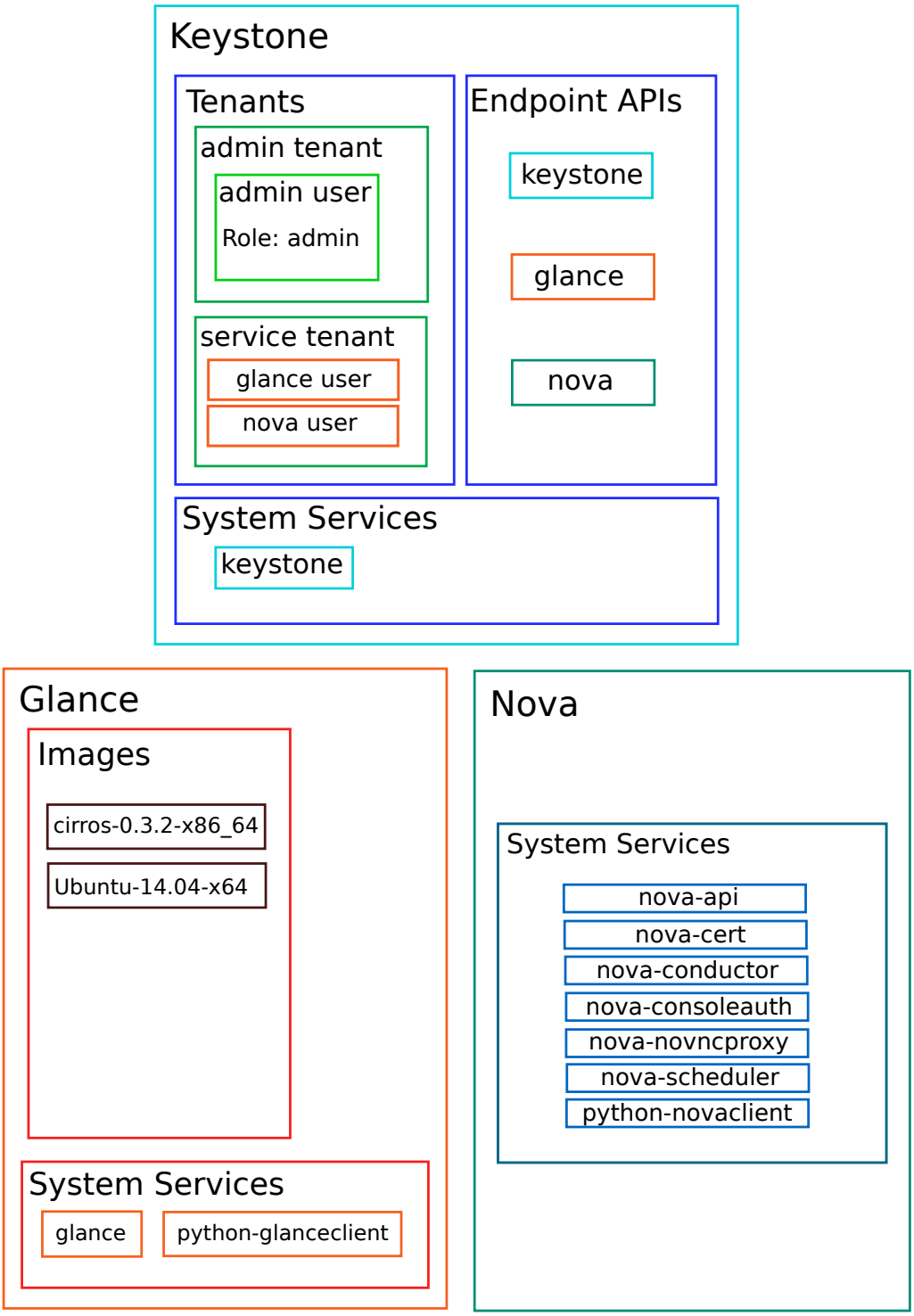

Figure 3.4: Complete setup on the controller node.

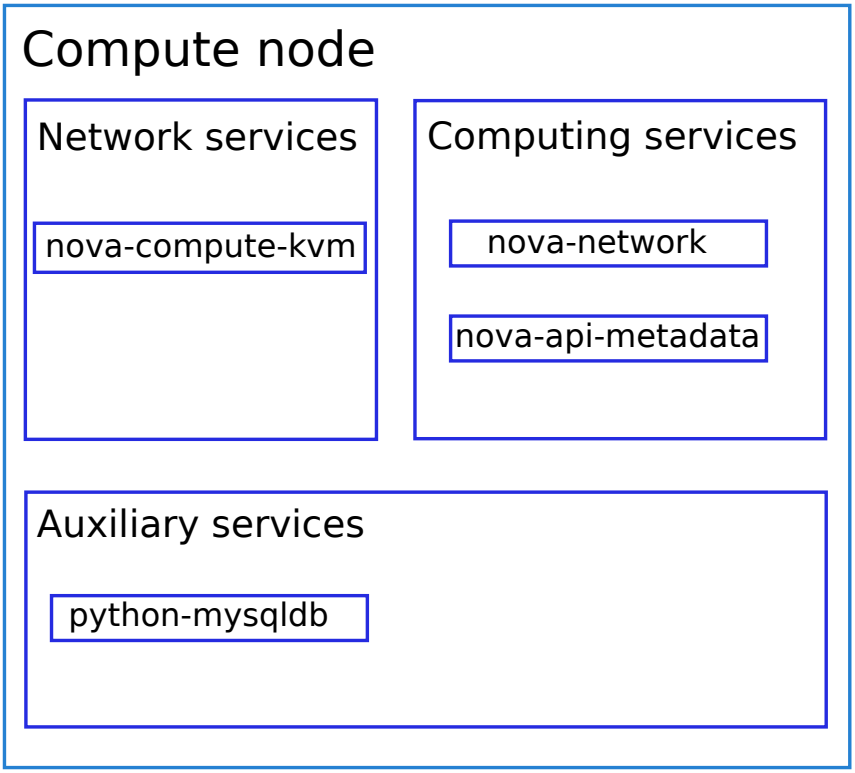

Figure 3.5: Compute node and its services.

So while the number of services running on a compute node are much smaller than the ones running on the controller node, there is a lot of configuration resting on the compute nodes, which contributes to the behavior of the overall system. The steps that are carried out in order to configure the compute node also have to be carried out on every other compute node you want to add to the environment. After the configuration is successful the new host automatically communicates via *nova-api-metadata* service to acknowledge the controller that it is available for hosting VMs. In our case we have configured both our compute nodes **Tartu** and **Vilnius** this way.

3.6 Instances

The next thing to do is starting the instances. We have three Instances we want to bring up, two on **Tartu** and one on **Vilnius**. They all reside on the same network and share a securitygroup that allows ICMP traffic on all ports, TCP traffic on port 22 and HTTP traffic on port 80. The following are their properties:

Instance01

- Host: Tartu
- Number of virtual CPUs: 1
- Memory: 2048 MB
- Disk size: 20 GB
- OS: Ubuntu 14.04 amdx64 cloud
- Yields: nginx webserver listening on port 80

Instance02

- Host: Tartu
- Number of virtual CPUs: 1
- Memory: 2048 MB
- Disk size: 20 GB
- OS: Ubuntu 14.04 amdx64 cloud
- Yields: wrk measurement tool

Instance03

- Host: Vilnius
- Number of virtual CPUs: 1
- Memory: 2048 MB
- Disk size: 20 GB
- OS: Ubuntu 14.04 amdx64 cloud
- Yields: wrk measurement tool

The following Figure 3.6 depicts the deployment situation with the connections that are to be measured in our environment. The connection does not leave the host, when both VMs are deployed on the same compute node, while it is tunneled through the Nova-networking service on the controller node in case of the VMs residing on different hosts.

Neutron is the other possibility to set up a networking infrastructure in OpenStack. It is much more voluminous than Nova-networking and more extensive in the setup. First the network infrastructure of the controller node has to be altered, as the Neutron networking services run here. In large scale OpenStack installations it is possible to deploy the Neutron Network service on a separate host. This would be called the

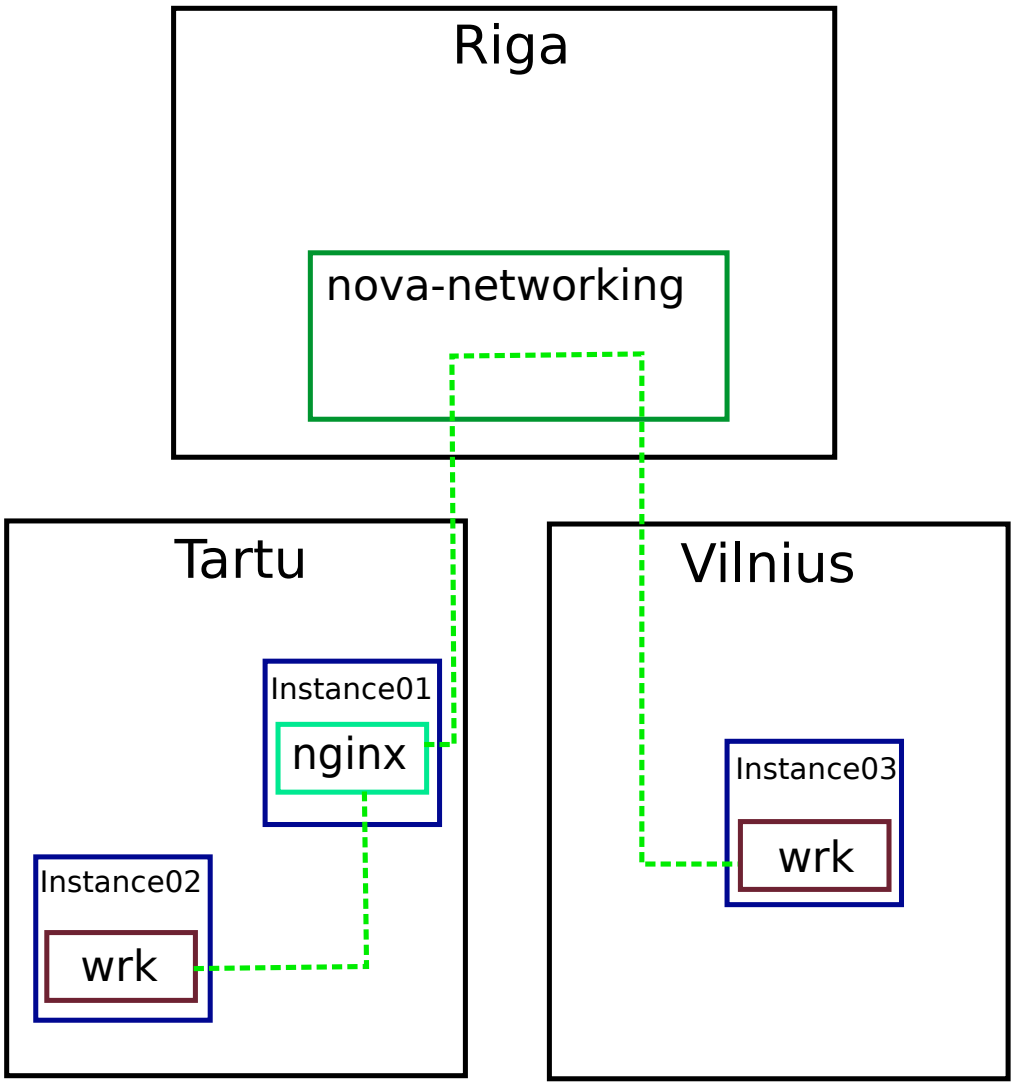

Figure 3.6: VMs in the Nova measurement setup.

Network node. In this case Neutron's main services run on the controller node. The first step is to create the Neutron user in the service tenant, and create the Neutron API. Then the Neutron service must be registered to Keystone. Then Nova has to be configured to use Neutron as the networking service. The next step is the installation of the Neutron services. They are *neutron-server neutron-plugin-ml2 neutron-plugin-openvswitch-agent neutron-l3-agent neutron-dhcp-agent*. The *neutron-plugin-ml2* is a service that controls the use of resources like the Open vSwitch, and the Linux bridges, that build up the networking backbone of Neutron [28]. Open vSwitch is a virtual switch over which the VMs are connected. The tunneling mechanisms to separate tenants from each other are implemented here. Neutron also operates its own DHCP service for its internal network. The modular Layer 2 (ML2) plug-in is then configured to use Open vSwitch

and to use the hybrid IP tables firewall Open vSwitch offers. The integration bridge and the external bridge used for separating tenant communication and the internet access of the VMs have to be set up and added to the Open vSwitch. Adding the external interface of the controller node to the external bridge is a critical part in this setup, as we access the host via a ssh connection that is dependent on this interface. The port binding and reconfiguration of the network addresses (one has to give the external address to the external bridge, while removing it from the external interface) is done by executing a script. The configuration on the controller node is finished

Regardless of whether you deploy the main services of Neutron on a separate node or on the controller node, there are services to be run on the compute nodes. These are *neutron-common neutron-plugin-ml2* and the *neutron-plugin-openvswitch-agent*. On this node the ML2 plugin has to be configured to work together with the counterpart on the controller node. Then the integration bridge is added to the Open vSwitch. After this the configuration of Neutron is finished and instances can be launched. The setup of VMs and connections is similar to the situation in Nova, but flows through the integration bridge, as can be seen on Figure 3.7.

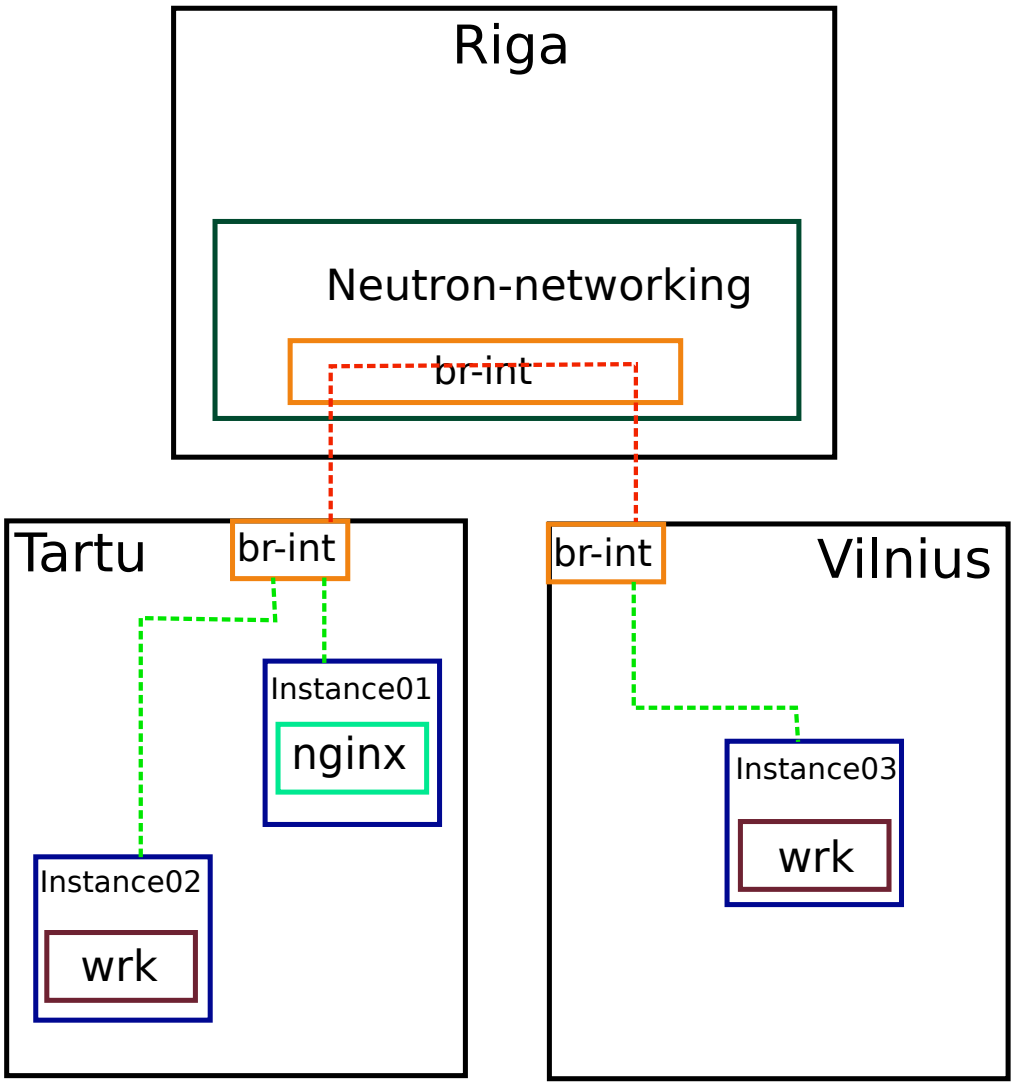

Figure 3.7: VMs in the Neutron measurement setup.

Chapter 4

Measurements

On Instance01 a nginx webserver is deployed. It hosts files of different sizes and serves them on demand via a HTTP request. This request can be carried out from the wrk tool installed on both Instance02 and Instance03. In the case of Instance02 the traffic never leaves the host, while as explained above the traffic caused by the request from Instance03 will pass through the network service on the controller node. This should result in a significant difference of latency times between the two connections.

4.1 Tool selection

Despite the development of the releases shown above, we used the Icehouse version of OpenStack. This is due to the fact that it is the most documented on the openstack.org site. This is the source which many of the expertise necessary to build a setup came from. We decided to install the Ubuntu 14.04 Operating system as the very basis of our setup, and Icehouse is the default release for the cloud computing on Ubuntu 14.04 [29].

Ubuntu 16.04 was only released during our work [30], and it would have been too time consuming to switch both the Linux distribution and the OpenStack release (which would have been almost necessary, as the default release for Ubuntu 16.04 is Newton [31]).

For our webserver setup we will use **nginx**. This choice was made, as it is smaller and easier to install and operate than e.g. Apache, while still providing all needed functionalities and beeing free to use. The statistics speak for itself: It is the second most used web server among the top 10 million websites [32].

As a measurement tool we use a lightweight solution called wrk. It is an open source software which works great with a wide range of webservers and not only with specific ones like e.g. ApacheBench (ab) [33].

4.2 Results

All measurements are carried out from the wrk tool sending HTTP requests to the nginx web server located on Instance01. This instance is deployed on **Tartu**. The measurements are all from the same network. Each measurement cycle will last 60s and will be carried out 2 threaded. 7 different file sizes will be provided by the webserver. For each file size there is a full measurement cycle carried out.

The files are chosen to cover a wide range of diferent applications. To cover very small files like short .txt documents a file of size 100 byte is provided. The next bigger file is 100 kilobyte. It represents e.g. small images or sound examples. Both the next two sizes shall represent webpages. Today's webpages are usually between 1500 and 2000 kilobyte in size. To test the boundaries of our setup there are three files left. These are 50 MB, 100 MB and up to even 1 GB in size.

As a base case a measurement with 20 simultaneously open connections is conducted. As can be seen in Figure 4.1 the latency increases with file size, as expected.

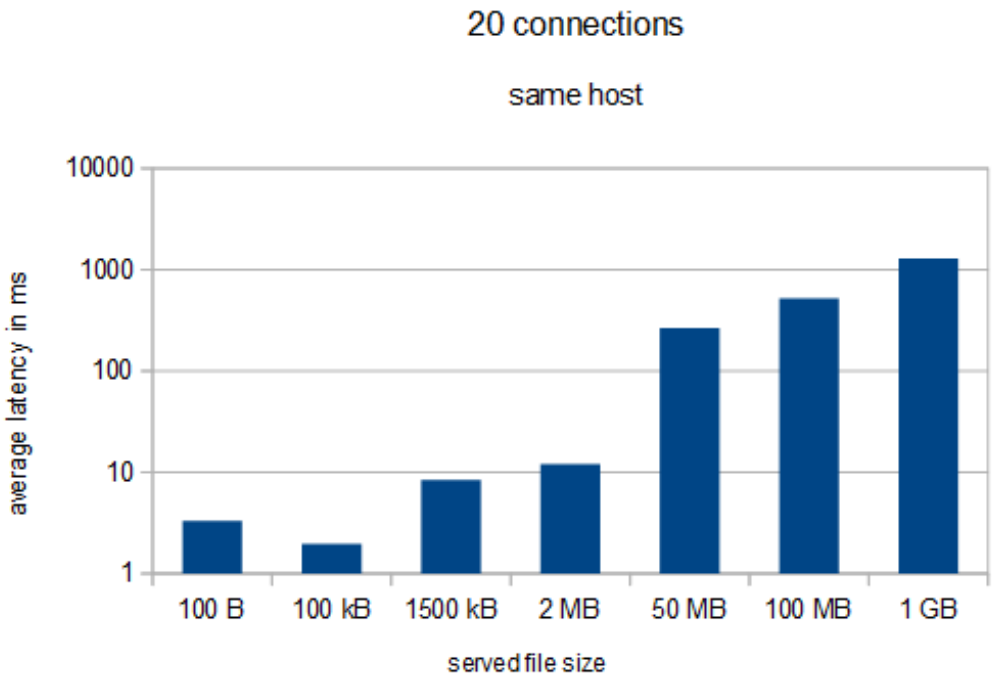

Figure 4.1: Latency measured with 20 connections open from same node. Diagram in logarithmic scale.

From the diagram we can see one anomaly: The latency is higher when the file of 100 B is transferred. This is likely to be explained by the packet size: The packets sent for this file size are much smaller then for the other file sizes. Therefore a much larger number of them can be given out by the webserver in a shorter period of time. The setup can not deal with such a high packet rate, leading to higher packet loss and therefore to higher latency.

Measurements of the throughput reaffirm this suggestion. From Figure 4.2 one can see that the throughput is significantly less in the case of the 100 B file.

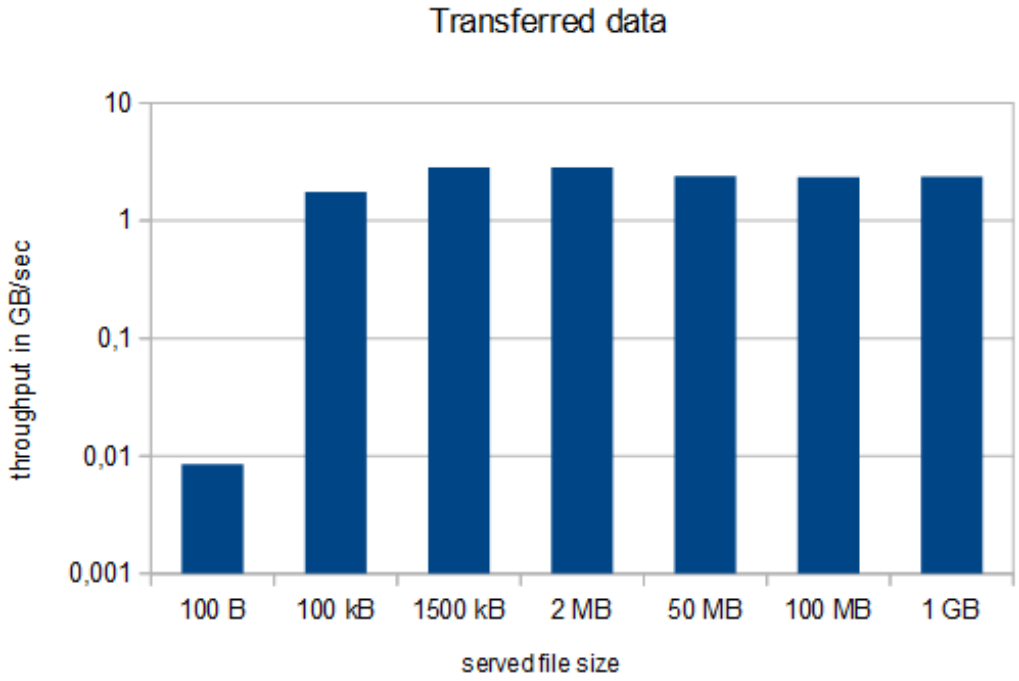

Figure 4.2: Data throughput with 20 connections open from same node. Diagram in logarithmic scale.

To push the capabilities of the setup, the next measurement was made with 100 open connections. What can clearly be seen in Figure 4.3 is the latency going up. In contrast to the former Figure 4.1, this diagram is in linear and not logarithmic scale. The darkblue bars depict the 20 connections base case, while the orange bars indicate the case for 100 connections. One can see that the latency increase fluctuates around a constant, and is not dependent on the size of the file. This can be explained by the more open connections which create more packets in the communication link. A higher number of packets leads to an increase in latency.

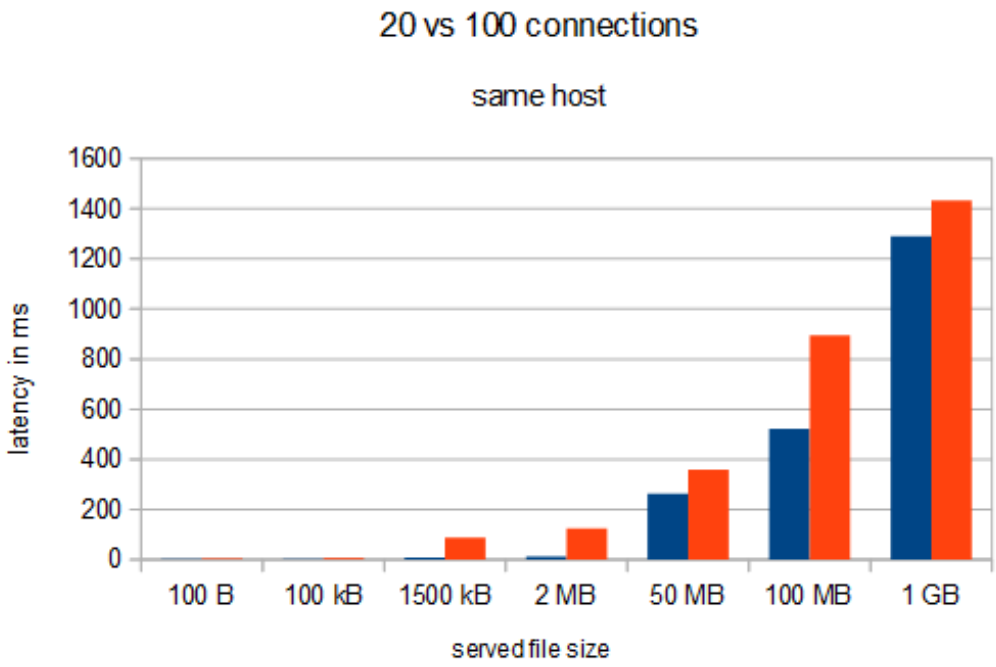

Figure 4.3: Measurement with 20 (darkblue) vs 100 (orange) open connections on same host. Diagram in linear scale.

In the next step the measurement is taken from the remote host. This traffic is directed through the controller node. This should lead to a substantial increase of the latency, as the traffic has to leave the remote host, enter the controller node, is redirected there and then lead to the host where the webserver resides. This prediction is proven valid, as visualized in Figure 4.4. The dark blue bars again stand for the base case measurement carried out from a VM residing on the same node (**Tartu**) as the VM that runs the webserver. The light blue bars depict the latency in ms for the same file requested from the VM deployed on the remote host (**Vilnius**). There is a change in the number of displayed bars, which is due to the fact that the connection for bigger file sizes crossed the 30 seconds mark, which causes the nginx webserver to time out the connection. All of the connections in the measurements from the remote host requesting the files from 50 MB upwards timed out that way.

One can see that the anomaly of larger latency in the smallest file size decreases when the measurement is taken from the remote host. The suggested explaination is that the multi node setup can handle higher packet rates.

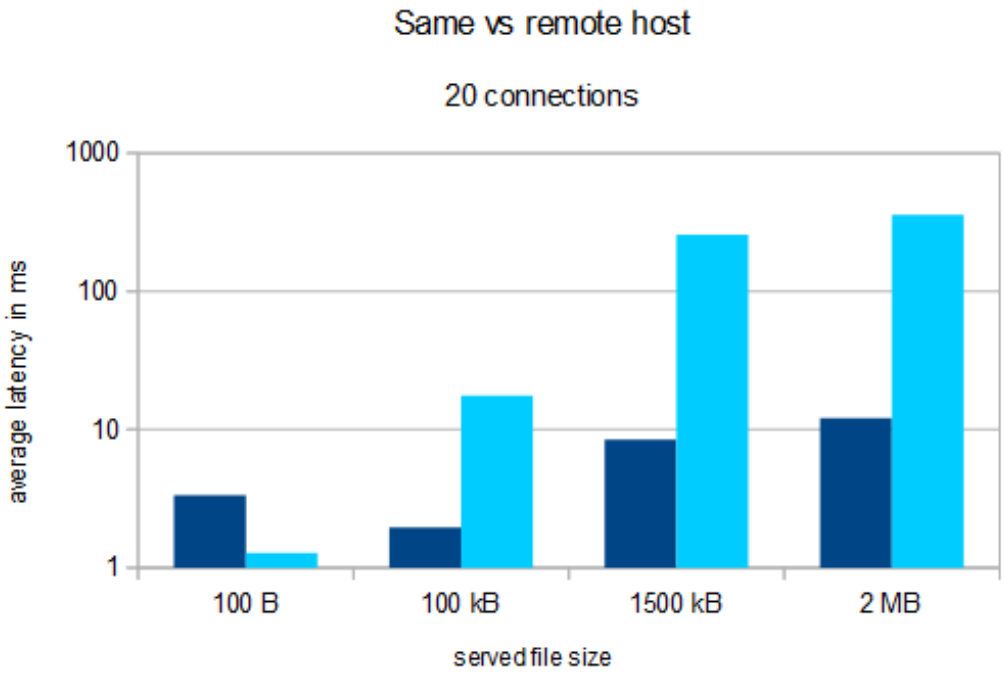

Figure 4.4: Measurement with 20 connections on same (darkblue) vs remote (lightblue) host.

Neutron is advocated to replace Nova-networking as the OpenStack network service in the future instead of existing as an alternative. So an interesting question is: how does it perform compared to Nova-networking. Our base case was made with the Nova-networking setup and again it is depicted as the dark blue bars in Figure 4.5. The green bars depict the same measurement (20 connections, same host) carried out on the setup with Neutron. The diagram shows, that the latency with Neutron is higher than the one with Nova-networking. This can be explained by the nature of Neutron's packet flow. As mentioned before all traffic from Neutron is always routed through the network module, no matter if it is located on the same node or not. This of course makes it possible to e.g. implement further security mechanisms, but this test case shows the price for that, which is payed in networking latency. In our measurements the latency when using Neutron is around 13 - 14 times the latency when using Nova-networking.

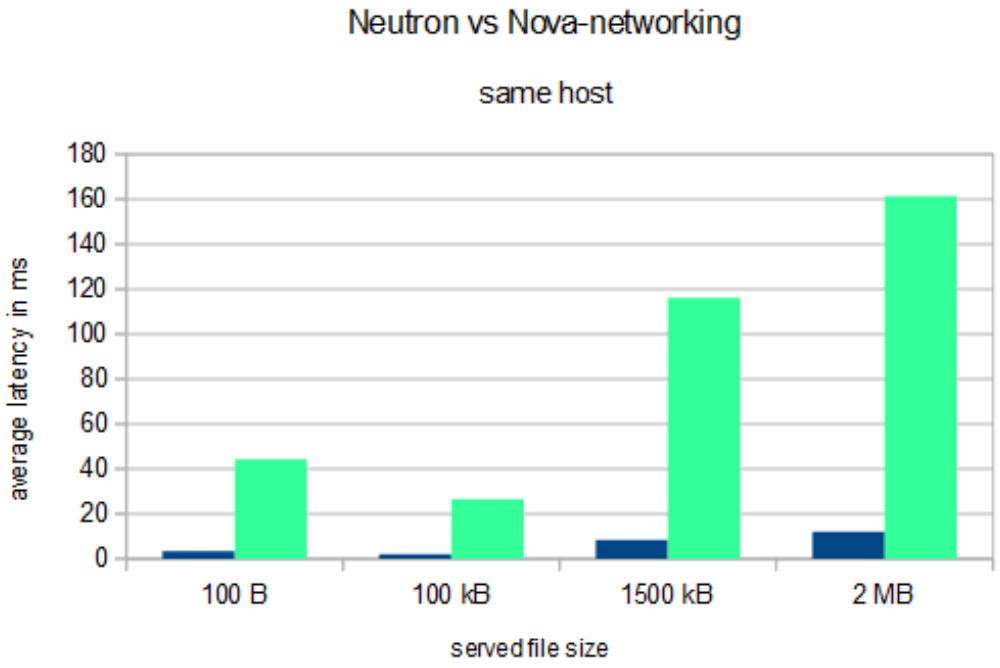

Figure 4.5: Neutron (green) vs Nova-networking (darkblue) with 20 open connections.

With the logic from the former result, one can predict, that the difference between a measurement from the same host and one from the remote host should not be as high as in the Nova-networking case, when using Neutron-networking. Figure 4.6 shows the measurement from the remote host (purple bars) compared to the one from the same host (green bars) for the Neutron case. And as predicted there is only a insignificant difference, which can be explained by a small asymmetry in the connection of the hosts.

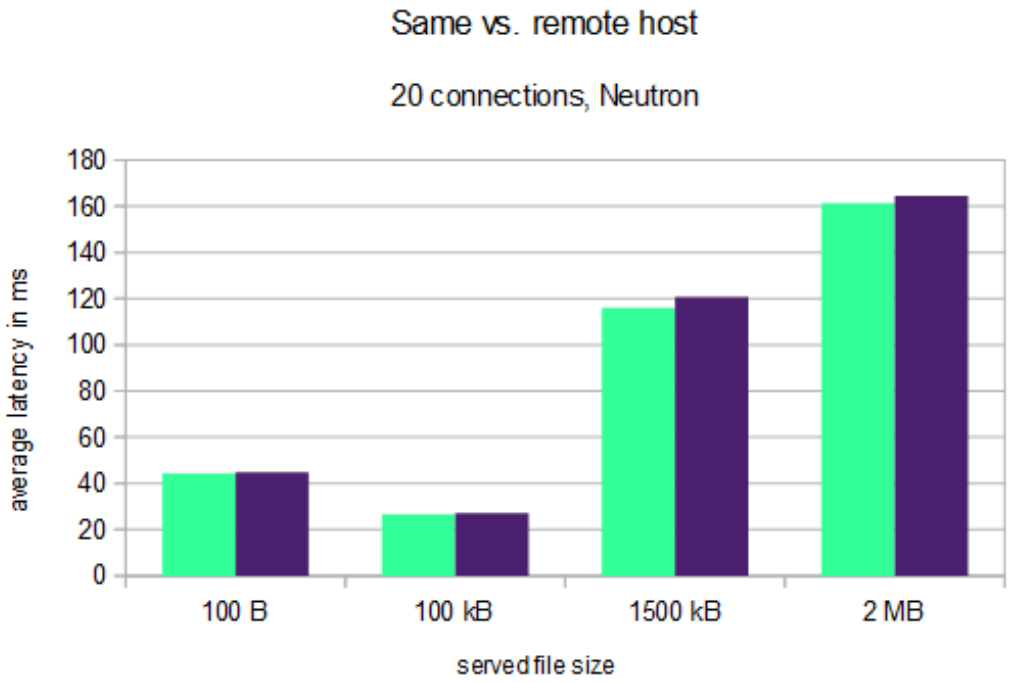

Figure 4.6: Measurement Neutron remote (purple) and same host (green).

Chapter 5

Conclusion

In this thesis a cloud computing setup was proposed, which is capable of hosting a web server. This webserver was run on VMs which were deployed in two different OpenStack environments. The first environment implemented the networking solution of the Nova-networking service. In the second environment the Neutron-networking service was used. Measurements of the network performance were made under different file sizes, and from different places within the multi node environment. The average latency occuring when using Nova-networking and Neutron in a one host setup was compared. In the same-node test cases the latency measured when using Neutron was by a factor of more than 10 larger than the latency when using Nova-networking. The results suggest that nova networking provides a better networking performance than Neutron, when used on the same node.

This shows that Neutron is not ready yet to replace its predecessor Nova-networking. This issue could be approached by adding a light weight Nova-like networking module to Neutron, for cases, when the user wants to reach the highest network performance possible, and does not need the additional benefits Neutron has to offer.

Neutron has advantages with large setups, containing big numbers of VMs. This is due to the scalability Neutron has to offer. When run on a separate host, it does not have to share resources with the controller nodes services. Nova-networking is more limited in performance due to its intertwining with the controller node's Nova services.

For smaller setups, especially when there is only one compute node, Nova-networking is a good choice, as the traffic flow stays in the same node and does not have to move through the networking modules on the controller node. This results in a lower latency. A disadvantage of this solution is, that the Neutron services such as Firewall as a Service or VPN as a Service are not included in Nova-networking.

5.1 Future work

The anomaly of the higher latency when serving a very small file size could be further investigated. Observing the packet behavior could confirm the suggestion of this thesis, that the higher latency is caused by a limited packet rate in the same node setup.

In the future the impact of Neutron's additional services on the network performance could also be analyzed. These services are e.g. Firewall as a Service or VPN as a Service. Then these could be compared to a solution that is based on the Nova-networking service. This would contribute to the development process of Neutron and the ultimate goal to replace Nova-networking.

Bibliography

[1] "Openstack conceptual architecture," http://docs.openstack.org/icehouse/install-guide/install/apt/content/ch_overview.html, accessed: 2016-10-04.

[2] "Openstack releases," https://releases.openstack.org/, accessed: 2016-08-26.

[3] X. Wen, G. Gu, and Q. Li, "Comparison of open-source cloud management platforms: Openstack and opennebula," *Fuzzy Systems and Knowledge Discovery (FSKD)*, May 2012.

[4] C. Li, J. Xie, and X. Zhang, "Performance evaluation based on open source cloud platforms for high performance computing," *Intelligent Networks and Intelligent Systems (ICINIS)*, Nov. 2013.

[5] F. Callegati, W. Cerroni, and C. Contoli, "Performance of network virtualization in cloud computing infrastructures: The openstack case," *Cloud Networking (CloudNet)*, Oct. 2014.

[6] V. Goldberg, F. Wohlfahrt, and D. Raumer, "Datacenter network virtualization in multi-tenant environments," *8. DFN-Forum Kommunikationstechnologien*, Jun. 2015.

[7] F. Callegati, W. Cerroni, and C. Contoli, "Performance of multi-tenant virtual networks in openstack-based cloud infrastructures," *Globecom Workshops (GC Wkshps)*, Dec. 2014.

[8] T. B. et al., *IaaS mit OpenStack- Cloud Computing in der Praxis*, 1st ed. dpunkt Verlag, 2014.

[9] "A bit of openstack history," http://docs.openstack.org/project-team-guide/introduction.html, accessed: 2016-10-08.

[10] "Apache license - version 2.0, january 2004," https://www.apache.org/licenses/LICENSE-2.0, accessed: 2016-09-28.

[11] "15 most powerful openstack companies," http://www.networkworld.com/article/2176960/cloud-computing/15-most-powerful-openstack-companies.html, accessed: 2016-07-28.

[12] “Amazon ec2 beta release,” https://aws.amazon.com/de/blogs/aws/amazon_ec2_beta/, accessed: 2016-08-28.

[13] “Amazon ec2 final release,” https://aws.amazon.com/de/blogs/aws/big-day-for-ec2/, accessed: 2016-09-28.

[14] “Aws cloud tour 2011 | australia: Event highlights,” https://www.youtube.com/watch?v=uf07L1RUOW4, accessed: 2016-09-28.

[15] “Wikileaks website pulled by amazon after us political pressure,” https://www.theguardian.com/media/2010/dec/01/wikileaks-website-cables-servers-amazon, accessed: 2016-09-28.

[16] “About the opennebula project,” http://opennebula.org/about/project/, accessed: 2016-09-28.

[17] “Opennebula key features,” http://opennebula.org/about/key-features/, accessed: 2016-09-28.

[18] “Microsoft forges more open-source partnerships for added cloud interoperability,” http://www.computerweekly.com/news/2240225137/Microsoft-forges-more-opensource-partnerships-for-added-cloud-interoperability, accessed: 2016-09-28.

[19] “Opennebula featured users,” http://opennebula.org/users/featuredusers/, accessed: 2016-09-28.

[20] E. S. Kevin Jackson, Cody Bunch, *OpenStack Cloud Computing Cookbook*, 3rd ed. Packt Publishing Ltd., 2015.

[21] “Openstack - image service installation,” http://docs.openstack.org/icehouse/install-guide/install/apt/content/glance-verify.html, accessed: 2016-09-28.

[22] “Openstack - image service installation,” http://docs.openstack.org/icehouse/install-guide/install/apt/content/glance-verify.html, accessed: 2016-09-28.

[23] “Networking with nova-network,” http://docs.openstack.org/admin-guide/compute-networking-nova.html, accessed: 2016-10-06.

[24] “Openstack networking name change: From quantum to neutron,” http://searchsdn.techtarget.com/news/2240200685/OpenStack-networking-name-change-From-Quantum-to-Neutron, accessed: 2016-10-06.

[25] “what can rabbitmq do for you?” https://www.rabbitmq.com/features.html, accessed: 2016-10-04.

[26] “Verify the image service installation,” http://docs.openstack.org/icehouse/install-guide/install/apt/content/glance-verify.html, accessed: 2016-10-08.

[27] “Configure a compute node,” http://docs.openstack.org/icehouse/install-guide/install/apt/content/nova-compute.html, accessed: 2016-10-08.

[28] “Neutron/ml2,” https://wiki.openstack.org/wiki/Neutron/ML2, accessed: 2016-09-04.

[29] “Openstack packages - ubuntu cloud archive,” http://docs.openstack.org/icehouse/install-guide/install/apt/content/basics-packages.html, accessed: 2016-09-28.

[30] “Ubuntuusers - ubuntu 16.04 lts,” https://wiki.ubuntuusers.de/Xenial_Xerus/, accessed: 2016-10-04.

[31] “Openstack cloud archive,” https://wiki.ubuntu.com/OpenStack/CloudArchive, accessed: 2016-10-04.

[32] “Usage of web servers broken down by ranking,” https://w3techs.com/technologies/cross/web_server/ranking, accessed: 2016-10-08.

[33] “ab - apache http server benchmarking tool,” https://httpd.apache.org/docs/2.4/programs/ab.html, accessed: 2016-10-08.