An Investigation Into The Readiness Of Open Source Software To Build A Telco Cloud For Virtualising Network Functions

Submitted in fulfilment
of the requirements for the degree of

Master of Science

at Rhodes University

Tapiwa C. Chindeka

Supervisors: Mosiuoa Tsietsi
Alfredo Terzoli

Grahamstown, South Africa
January 2019

Abstract

Cloud computing offers new mechanisms that change the way networks can be created and managed. The increased demand for multimedia and Internet of Things (IoT) services using the Internet Protocol is also fueling the need to look more into a networking approach that is less reliant on physical hardware components and allows new networks and network components to be created on-demand. Network Function Virtualisation (NFV) is a networking paradigm that decouples network functions from the hardware on which they run on. This offers new approaches to telecommunication providers who are looking to new ways of improving Quality of Service (QoS) in cost effective ways. Cloud technologies have given way to more specialised cloud environments such as the telco cloud. The telco cloud is a cloud environment where telecommunication services are hosted utilising NFV techniques. As the use of telecommunication standards moves towards 5G, network services will be provided in a virtualised manner in order to keep up with the demand.

Open source software is a driver for innovation as it is has a collaborative culture to support it. This research investigates the readiness of open source tools to build a telco cloud that supports functions such as autoscaling and fault tolerance. Currently available open source software was explored for the different aspects involved in building a cloud from the ground up. The ETSI NFV MANO framework is also discussed as it is a widely used guiding standard for implementing NFV. Guided by the ETSI NFV MANO framework, open source software was used in an experiment to build a resilient cloud environment in which a virtualised IP Multimedia Subsystem (vIMS) network was deployed. Through this experimentation, it is evident that open source tools are mature enough to build the cloud environment and its ETSI NFV MANO compliant orchestration. However, features such as autoscaling and fault tolerance are still fairly immature and experimental.

Acknowledgements

Firstly, I would like to thank my supervisors Dr. Mosiuoa Tsietsi and Prof. Alfredo Terzoli for their guidance throughout this journey. Their support has been vital to the successful completion of this research and is truly appreciated.

Secondly, I would like to thank my parents for the love and support they have shown me. I would not be here without them and their continued support and encouragement. The time and effort spent on this work is dedicated to them.

Thirdly, I would like to extend my gratitude to my friends and colleagues who have helped me with encouraging words, brainstorming and venting sessions. Special thanks go to Vongai Chindeka, Michael Zvidzayi, Tinashe Magwenzi, Tichaona Zvidzayi, Yonela Pipile, Nicole Oyetunji, Katherine James, Tracy Jeffery, Valerie Motsumi-Moyo, Ezam Majokweni, Fr. Gerry O'Reilly and many others who provided much needed support. I am grateful for the invaluable inspiration and help.

I would also like to thank the Computer Science staff members for their kindness and support.

Lastly, I would to thank the Lord who saw me through every high and low and gave me the strength to persevere.

This work was undertaken in the Distributed Multimedia Centre of Excellence at Rhodes University, with financial support from Telkom SA and CORIANT. The author acknowledge that opinions, findings and conclusions or recommendations expressed here are those of the author and that none of the above mentioned sponsors accept liability whatsoever in this regard.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Service provision in the telecommunication industry has over the years been based on using dedicated hardware devices that perform specific functions within the network. With the nature of telecommunication evolving, there is an increasing demand for diverse new services with high data demands. This in turn increases CAPEX/OPEX for the network service provided. The increase in operating costs can not simply be balanced by increasing subscription fees because competition between TSPs will result in customer churn [1]. Telecommunication networks need innovative, cost effective way to handle the dynamic nature of the load while maintaining a good level of quality. One such way of improving network agility is through the use of the cloud in delivering services. Network Function Virtualisation (NFV) is a mechanism for improving network flexibility and speed that is gaining popularity because it shifts the need for dedicated hardware appliances to using software solutions on commodity hardware. Section 1.1 will look at some of the motivations for adopting NFV as a long-term networking solution.

1.1 Benefits of NFV

NFV decouples network functions that were traditionally appliance-based so that they can run on any server through the use of virtualisation. The power of NFV is mostly in the flexibility it brings, making it easier for service providers to build scalable and automated network solutions. NFV brings flexibility to network functions, as services can be created, scaled or removed by bringing up or tearing down network functions without the need to configure new hardware into the network.

1.2 The Evolution of Telecommunication Systems

The mobile and fixed telecommunication industries have also been through several paradigm shifts as evidenced by the move from first (1G), second generation (2G), third generation (3G) to fourth (4G) and now fifth generation (5G) networks. The unifying theme behind this evolution has been a decisive shift towards packet-switched technologies with a strong focus on IP (Internet Protocol) [2]. The Generation (G) in wireless mobile technology refers to the changes in data capacity, speed, latency, frequency and the general nature of the system. Each generation is characterised by its own standards, techniques and new features that distinguish it from previous generations [3]. Figure 1.1 shows the evolution of mobile telecommunication. 1G technologies came about in 1980 and was characterised by the analog transmission of voice signals. 2G came around 1991 and was based on the global system for mobile communication with a speed of about 64 kbps. It introduced text messaging in mobile telecommunication. 3G technologies debuted in 2000 and enhanced mobile data communication by improving transmission speeds (minimum of 200 Kbps) through the use of packet switching techniques. Other ways it enhanced mobile communication is by offering global roaming Web browsing and multimedia communication [4]. 4G technology is based on packet switching and completely moves away from circuit switched standards. Data transfer rates for 4G should be between 100 Mbps and 1Gbps [5]. It increases data transfer rates and improves handovers across different networks by making them smoother and drastically reducing data loss. 5G is the upcoming generation of mobile technologies that promises even faster speeds and improved connectivity [3, 6, 7]. With the rise of the Internet of Things (IoT) more devices will be linked to the internet and most of these will require fast and reliable connections. Voice over IP (VoIP) and multimedia communication continue to increase in popularity as the favoured means of communicating in both social and business contexts. Users expect improved user experience regardless of the increase in traffic load in the network. In order to cope with this load and maintain Quality of Service (QoS), NFV provides the building blocks of how future networks will be deployed to support the goals of 5G.

1.3 Research Context

While virtualisation in itself is hardly a new concept, until recently telecommunication operators have lacked a comprehensive guiding framework that outlines the design and execution of virtualised functions in a consistent way. What was needed was a standard

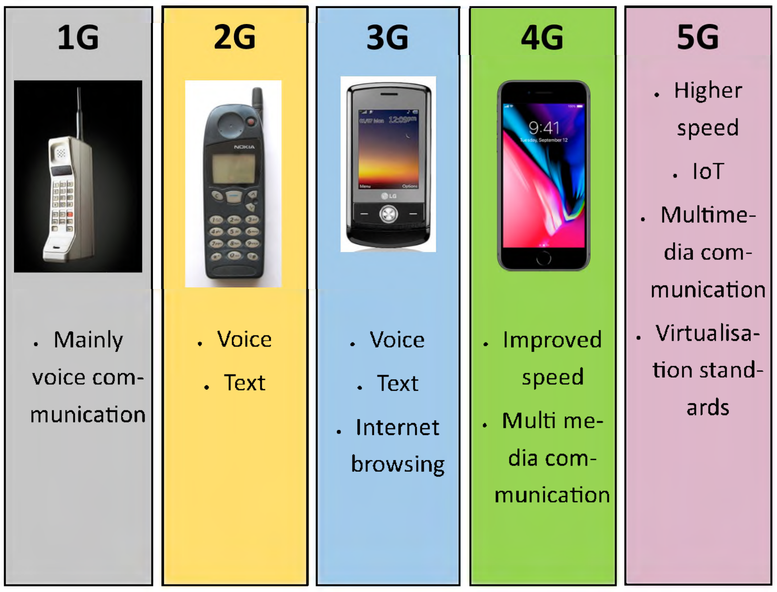


Figure 1.1: The Evolution of Mobile Telecommunication Standards.

(or set of standards) that all operators could follow in the interests of standards compliance and interoperability. This role has been fulfilled by the European Telecommunication Standards Institute (ETSI) which has a long history in developing standards for interoperable services in fixed networks [2]. In particular, ETSI has developed a comprehensive Management and Orchestration (MANO) framework which details the provision of virtual network functions in a way that is loosely coupled to storage, network and compute resources in public or private cloud installations. The resulting framework potentially gives operators a solid base to build on when implementing functions such as auto-scaling, policy management, regulatory compliance, topology management and others [8].

Compute, networking and storage resources are the backbone of any cloud environment into which virtualised network functions are deployed. While ETSI standards can apply to operators leasing virtual functions from public clouds, many of them, for security and operational reasons, will seek to implement their own private clouds. Of all the software platforms that are being used for this purpose, OpenStack is the most prevalent [9]. OpenStack is an open source cloud platform consisting mainly of Nova, Neutron and Cinder components which provide facilities for compute, networking and storage functions respectively [9]. OpenStack can manage large pools of such resources spread throughout an operators datacentre using a single management interface or lightweight RESTful web services. The flexibility and power of the platform has led to the formation of an open consortium of practitioners collectively known as the Telco Working Group. The group seeks to promote the requirements of operators among the general OpenStack community through various initiatives, including the contribution of open source extensions and modifications [10].

While OpenStack provides a suitable integration platform for prototyping a cloud-based telecommunication system, there are a number of sophisticated platforms that are more closely aligned with ETSI MANO specifications. Among these, the most mature is OpenBaton which is a complete, open source ETSI MANO implementation [11]. OpenBaton implements the full underlying architecture and also supports the creation and deployment of new application use cases within OpenStack environments. In addition, to communicate the programming model, it provides a small number of example virtual functions that can be orchestrated, all of which are multimedia services that form part of an IP Multimedia Subsystem (IMS) service control layer. The suitability of the OpenBaton platform for a wide range of example use cases is yet to be determined, while the open source contributions of groups such as the Telco Working Group offers a bare bones alternative.

1.4 Research Question

The main question is the following: "Is it currently possible to build a fully functional telco cloud, in which to virtualise network functions, via free and open source components, deployed and utilized solely based on their documentation or commonly accepted conventions of installation and use?" This question can be decomposed in a number of sub-questions:

- What are the essential components of a cloud and particularly a telco cloud?
- What open source components are available which map to the essential components of a telco cloud?
- What is the level of readiness of these components? Readiness here should be interpreted as the possibility for a component to be deployed and work robustly without the need of exploring the actual source code behind the component.

1.5 Research Objectives

The sub-questions in the previous section map naturally to a series of research objectives, which are the following:

- To determine the essential components of a Telco Cloud,
- To determine the available free and open source components which fulfill the roles identified in the previous objective,
- To investigate the level of maturity of each component, in the sense specified above.

1.6 Methodology

The research question will be answered through gathering and organizing information, on one side, and direct experimentation, on the other. The two activities will run mostly in parallel, re-enforcing each other. Information will be gathered from academic papers, white papers, standards specifications from bodies such as the European Telecommunications Standards Institute (ETSI), and documentation on specific components and projects to be found on the web. The experimentation will structured as the actual construction of a telco cloud prototype, moving through the various phases that transform a bare metal infrastructure into a virtualised one. This will be followed by the deployment of an orchestrator, the addition of specialized monitoring capabilities and finally the deployment of virtualised network services, specifically the IP Multimedia Subsystem (IMS), which remains the foundation of a modern telco network.

1.7 Document Outline

The rest of the thesis is organised as follows:

- **Chapter 2** provides the reader with introductory information on the principles and background of the cloud and Network Function Virtualisation. Some of the existing cloud solutions are discussed. The concept of a telco cloud is unpacked to the reader and the ETSI NFV MANO framework is introduced as a guiding standard.

- **Chapter 3** discusses the IP Multimedia Subsystem (IMS) network which is the network service intended for virtualisation for this work.
- **Chapter 4** introduces the OpenStack cloud operating system which is the the software chosen to run the infrastructure of the telco cloud. The OpenStack concepts and services were also discussed to give the reader an overview of the architecture.
- **Chapter 5** discusses the first phase of deploying the telco cloud which involved building the Infrastructure as a Service. The process of how the OpenStack cloud is built from the physical server up is explained.
- **Chapter 6** deals with the incorporation of the management and orchestration system. One of the objectives of this research is to build a standard compliant telco cloud, therefore the ETSI NFV MANO compliant implementation chosen is discussed.
- **Chapter 7** deals with adding monitoring capabilities for monitoring the Virtualised Network Functions (VNFs) that make up the network service. The choice of monitor is presented and experiments done to test the interoperabilty of the monitor with the rest of the infrastructure are detailed.
- **Chapter 8** discusses the last phase of experimenting with building the telco cloud. Different ways of deploying an IMS network on the OpenStack cloud were experimented with and discussed. Details of the attempts to perform autoscaling and fault management on the VNFs are provided.
- **Chapter 9** gives an overall discussion of the implementation process and insights gathered from the different phases of deployment.
- **Chapter 10** concludes the thesis by outlining the conclusions drawn from the work in relation to the objectives of the research. Possible ideas for future work are also presented.

Chapter 2

Literature Review: Cloud Computing and Network Function Virtualisation

The cloud paradigm has revolutionised computing by offering advantages through reduction in operating costs, offering flexible system configuration and on-demand service provisioning. The shift to the cloud, which has been successful for general computing, is still growing and is also revolutionising how networks are set up and managed through cloud virtualisation. Network Function Virtualisation (NFV) is one of the ways in which cloud virtualisation is changing how networks are built. Network operators have taken interest in the advantages offered by the cloud and NFV and are slowly migrating to the Telco Cloud. The Telco Cloud is a cloud environment tailored for telecom applications and IT services, where network functions are ran as applications on virtual machines [12]. This chapter starts by looking at a few ICT principles that are used for the provision of high quality services. These principles are also important in cloud and virtualised contexts. This is followed by a discussion of the cloud as the basis for the realisation of NFV and the existing cloud platforms, focusing on the Telco Cloud in particular. Next NFV concepts are presented and the advantages it offers to network operators discussed. NFV introduces new relationships between the virtualisation components and as such, network operators needed a set of guiding standards for managing and orchestrating NFV components. The existing standards for managing and orchestrating NFV systems are also discussed in this section. The final section will provide a summary of the chapter.

2.1 ICT Principles for Improved Quality of Service

ICT principles such as *high availability* and *fault tolerance* are key elements of providing high quality services and improved user experience. These concepts are of great interest as they can also be applied to cloud and virtualised environments.

2.1.1 High Availability

It is important for network operators to be able to connect calls and transfer data in a timely manner in order to maintain QoS. Network failure, even for a few seconds, can mean loss in revenue for the provider and a negative reputation among consumers. A big concern for network operators is what happens when there is a fault in the network: will the network continue to deliver services or will the services will be unavailable while the fault gets fixed [13]?

High Availability (HA) systems are configured to ensure operational continuity so that services are always on. The period in which a service or application is unavailable is referred to as downtime. High availability aims to prevent downtime by configuring the system to have access to more resources that work in unison [14]. The minimum benchmark for availability is five nines (99.999%). This means that the service downtime in a period of a year may not exceed 5 minutes [13].

HA alone is not enough because even with an HA of five nines, the network may still not be resilient. Resilience is a combination of HA and the ability to maintain QoS Service Level Agreement (SLA). This includes service continuity and the ability to return to its original state after recovering from a fault, without compromising the QoS [13]. For example, a system that only has an HA of (99.999%) but no resilience configured will only have downtime of 5 minutes per year, however, every time it recovers from a fault it does not return to its original state and therefore degrades the QoS.

2.1.2 Fault Tolerance

Fault tolerance is the ability for a system to continue functioning in the event of a failure. Failure is when a system cannot perform its intended function due to it being in an invalid

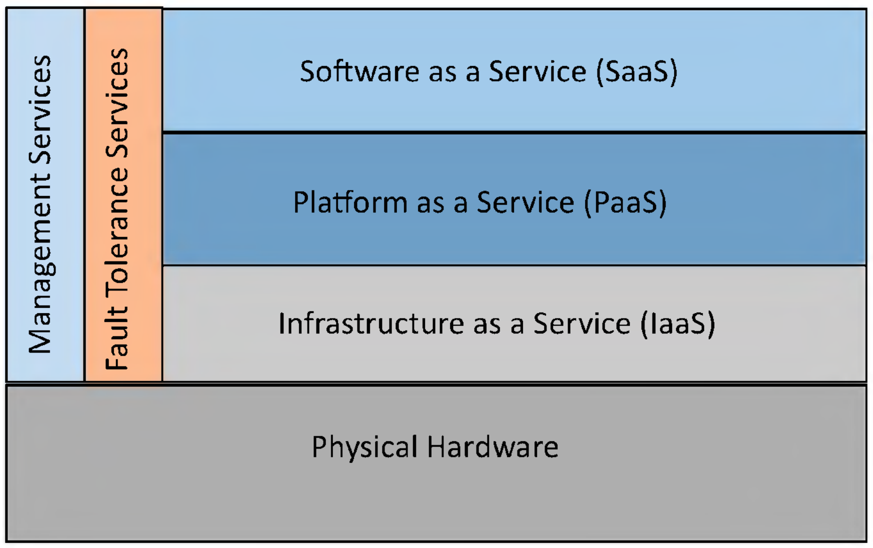

Management Services
Fault Tolerance Services
Software as a Service (SaaS)
Platform as a Service (PaaS)
Infrastructure as a Service (IaaS)
Physical Hardware

Figure 2.1: Cloud Computing Layered Architecture. Adapted from [15]

state. In order to develop a fault tolerant system, it is important to clearly define what the correct system behaviour is. This will enable developers and operators to specify what the characteristics of a failure are and use that knowledge to build a fault tolerant system [15].

Figure 2.1 shows the layered architecture of cloud computing systems. The layers: Software as a Service (SaaS), Platform as a Service PaaS) and Infrastructure as a Service (IaaS) are explained in Section 2.2. Due to the architecture in Figure 2.1, when failure occurs in any one of the layers, it generally affects the layers above it [15]. For example, a failure in the IaaS layer is likely to cause failures in the PaaS and SaaS layers as well.

The architecture of a fault tolerant system can include several components such as node managers and supervisors distributed across multiple nodes in order to maintain desirable system performance and reliability when some components of the system fail. Fault handling is part of creating a fault tolerant system and includes detecting failure of system components. The node manager can then attempt to restart the component. The supervisor can also migrate the functionality of that component to another node [16].

2.2 The Cloud

Over the years, there has been an increase in demand for computing resources and this has led to the growth in popularity of "the cloud". The advantages offered by the cloud make it easier to provide better performance through: flexibility in system configuration, reduction in operating costs, automated infrastructure optimisation and on-demand service provisioning [12, 17]. The cloud is an abstraction for software, services and infrastructure that is provided over the Internet on a pay-per-use basis [18]. These are made available through modalities such as:

- **SaaS (Software as a Service)**: Popular examples include web mail and streaming applications such as Google Apps and Netflix. SaaS takes the responsibility of managing and deploying software from the user and makes it the responsibility of the service provider. These applications are often offered on a subscription basis.
- **PaaS (Platform as a Service)**: This provides the platform where software is deployed. It takes care of the underlying servers, networking and operating system and provides the users with a ready to use environment.
- **IaaS (Infrastructure as a Service)**: This provides storage, networking and computing power which are the backbone of cloud services. IaaS solutions are typically managed via a dashboard and/or API, though terminal access is also usually available via SSH connection.

The cloud is also very relevant for networking in telecommunication because network capacity requirements are becoming more dynamic and the cloud can be used to provide better performance [12]. It also allows for automated infrastructure optimisation when dealing with dynamically changing workload [17].

2.2.1 Existing Cloud Solutions

Cloud solutions come in three flavours: public, private and hybrid. A public cloud is a cloud environment where the tenant does not manage and maintain the hosting of the cloud solution. Unlike a public cloud, a private cloud is an internal cloud hosted locally where the maintenance and management is the responsibility of the user. The hybrid cloud uses a combination of the on-premises, private cloud and the off-premises, third party, public cloud where traffic is able to flow between the clouds [19, 20].

With the increase in popularity of the cloud, users have several cloud solutions to choose from when wanting to implement a public, private or hybrid cloud. Below is a discussion of some of the popular cloud solutions.

Amazon Web Services (AWS)

Amazon Web Services (aws) [21] is a very mature closed source cloud services provider that offers a range of products including compute, storage, database, Internet of Things, machine learning, media services and networking and content delivery solutions. Most of these are provided on a subscription basis with some services available free for a limited quota or period.

A product that is of great importance to developers, administrators and operators is the management tools. The tools inside the toolkit include:

- **Amazon CloudWatch**: Which provides monitoring and management services. The data makes it easy to get insight on the application behaviour, ways to optimise resource utilisation and maintain good operational health.
- **AWS Auto Scaling**: Which monitors applications and automatically scales resources to maintain good performance.
- **AWS CloudFormation**: Which makes use of simple text files that work as templates that model how resources should be provisioned in an automated manner.
- **AWS OpsWorks**: Which is a service for configuration management of Chef and Puppet instances. Chef and Puppet are platforms for automating server configurations using code.
- **AWS Systems Manager**: Which gives a view and control of infrastructure. It has a user interface that allows the administrators to view operational data from multiple AWS services, as well as manage resources and applications in a simplified manner.

OpenStack

OpenStack is an open source cloud operating system that provides IaaS for creating public and private clouds. It provides compute, networking and storage resources which can be manage using the OpenStack API or a GUI dashboard [9]. Chapter 4 will provide more details on OpenStack and the services it offers.

Google Cloud

The Google Cloud Platform (GCP) [22] is made up of physical resources including servers and hard disk drives, as well as virtual resources such as VMs, all located at the Google data centres. There are several data centres located in different global regions. Each region consists of multiple isolated zones. The distribution of resources brings benefits such as redundancy, which is useful in the event of failure and makes the platform fault tolerant. Another benefit is reduced latency as resources can be allocated from the closest zone to the client.

GCP provides a variety of services that can be used to access different underlying resources on the platform. Services include: a Compute Engine that supports scalability, Cloud Storage which provides object storage, Persistent Disk for block storage, an App Engine which is a PaaS for applications, a Kubernetes Engine for containerised applications and Virtual Private Cloud (VPC) for networking resources. It also has services that help manage and orchestrate resources and application on the GCP. The services include: a Cloud Deployment Manager which manages cloud resources using templates and a Stackdriver that manages and monitors services, applications and infrastructure.

Oracle Cloud

Oracle Cloud [23] offers technologies such as platform automation, Artificial Intelligence (AI), machine learning and blockchain on their platform and infrastructure. The Oracle Cloud provides solutions in the form of SaaS applications, PaaS on which applications can be built, deployed and managed and IaaS that works on a subscription basis.

An Oracle Management Cloud suite is available to make monitoring and managing applications and infrastructure. The Oracle Management Cloud includes services such as:

- **Infrastructure Monitoring**: Monitors the health of the IT infrastructure. Visibility of resources status across all data centres using a single platform means that administrators are alerted on issues quickly.
- **Application Performance Monitoring**: Provides developers with information needed to fix application issues fast. Developers and operation teams are also able to identify any bottlenecks before they become they compromise the user experience and are able to use the on demand scale function.
- **Log Analytics** : Provides real time monitoring and analysis of log data from applications and infrastructure. This helps to reduce the average time it takes to resolve problems.
- **Orchestration**: Automates tasks through the use of scripts, REST endpoints or third part automation frameworks. The tasks performed can be at application level or cloud infrastructure level.

Microsoft Azure

Microsoft Azure [19] is a closed source cloud computing service that was developed by Microsoft and officially released in 2010. It allows users to build, deploy and manage applications and services that are hosted on data centers managed by Microsoft. Solutions include hybrid cloud applications, IoT, application monitoring and SaaS applications. These services are available on a subscription basis.

It supports hosting Linux and Windows VMs. A product called Batch allows administrators to schedule jobs and manage compute resources, while the Virtual Machine Scale Sets make it easy to scale up the number of VMs in the cloud. Cloud Services is a product that developers and operators can use to create, host and run highly scalable and highly available cloud applications. These Cloud Services can be used using .NET or Python. Management of the cloud can be simplified through automation. An Azure Monitor is available that gives real-time monitoring data to give administrators a view of how their applications and resources are behaving. To maintain high availability and performance, a Traffic Manager is available that can route incoming traffic appropriately.

2.2.2 Telco Cloud

The telco cloud leverages cloud computing capabilities such as supporting QoS, high availability of resources on demand and improved network manageability [12]. It presents the opportunity to have networks that are less dependent on dedicated hardware, and in turn, optimise resource utilisation since resources can be provided on demand rather than by over-provisioning. Virtualisation technologies makes it possible to provide reliable and carrier-grade performance by having network functions running on standard hardware [12]. This is a paradigm shift from a static resource deployment to a more dynamic and on-demand approach which is more flexible and offers improved resource utilisation [12, 24, 25].

Most network functions have carrier grade requirements such as high availability, QoS, fault tolerance and fault recovery. Typical telecom networks have geographically separate primary and secondary sites. Primary sites typically host the main network functions and the secondary sites are for concentration and distribution [25]. Hence, telcos have the advantage of having distributed points of presence (PoPs) which can be used to host small cloud environments that are suited for redundancy support[12].

Elastic Networks, High Availability and Fault Tolerance in the Telco Cloud

Having network functions hosted as VMs in the cloud makes it easier to scale network resources up and down and to reconfigure the network automatically. These networks that are configured to adjust accordingly in response to dynamic load are often referred to as elastic or flexible networks due to their dynamic nature.

Multimedia content has grown as the largest form of traffic in mobile networks over the years and such services require low latency and high bandwidth to maintain the expected quality of experience (QoE) [25]. This growth in multimedia traffic is the main contributing factor to load peaks experienced by telco networks. To minimise the effects of these changes, telco clouds can make use of high resource availability and fault tolerance. High availability minimises downtime by quickly restoring services in the event of a failure by making use of shared resources. Fault tolerance often relies on redundancy to make it easier to switch from one component to another when there is a failure or more components are required to handle the load.

Fifth Generation Technology - 5G

The emerging fifth generation (5G) technology will be "an agile resilient converged converged fixed/mobile core network based on NFV and SDN technologies and capable of supporting network functions and applications from different domains". Key NFV features for 5G include [26]:

- **Network Slicing**: which provides network-on-demand to applications and can be seen as an implementation of Network as a Service (NaaS).
- **Cloud-native Network Functions**: These are network functions implemented using re-usable components in the cloud rather than monolithic implementations.
- **Scalability**: Virtual Infrastructure Managers, which will be discussed in greater detail in Section 2.3, need to be flexible enough to support the level of scalability and distribution needed by 5G networks.
- **Reliability**: The requirements for reliability cover fault management, service availability and the ability to prevent, detect and recover from failure.

Six challenges that 5G is expected to address that are not yet effectively addressed by previous technologies are: massive device connectivity, higher data rates, low latency, consistent Quality of Experience, higher capacity and lastly, for all this to be achieved at a lower cost [27]. A subset of the services that will be provided by 5G includes voice over IP (VoIP) and generally, multimedia communication over IP, typically hosted in the IP Multimedia Subsystem (IMS). For simplicity, this thesis focuses specifically on IMS and not on all the services that 5G aims to cover.

Software Defined Networking - SDN

Current networks are typically vertically integrated, meaning that the control and data plane are together. Software Defined Networking (SDN) is a paradigm that changes vertically integrated nature of networks by separating the control plane from the data plane. The control logic is separated from the underlying switches and routers, creating logically centralised network control [28]. SDN aims to efficiently support the dynamic nature of future network functions while lowering operational costs [29]. The key features of SDN are [28, 29]:

- Separation of the control plane and data plane,
- Network programmability,
- A centralised controller that has a view of the network,
- Open interfaces between the control plane and data plane,
- Forwarding decisions are based on flows and not destination-based.

2.3 NFV

As previously mentioned, one of the main advantages of moving to the cloud is being less reliant on dedicated hardware components and take advantage of virtualisation strategies. Network Function Virtualisation is the implementation of network functions as software that run on standard hardware servers. This means that the network can be reconfigured and network functions can easily be moved to other locations in the network without the need to install new hardware equipment into the network [30]. In essence, NFV entails moving from monolithic proprietary hardware solutions to having software solutions running on virtual machines hosted on commodity hardware. Network functions are decoupled from compute, storage and network resources through the addition of a virtualisation layer [31]. The areas that NFV can be applied to include mobile network nodes, NGN signalling (for example IMS), switching, security functions (such as firewalls and intrusion detectors) and application optimisation (for example CDNs and load balancers) [30]. Telecommunication providers will essentially be able to implement the aforementioned network functions as instances of plain software [1].

NFV is a complementary approach to SDN. It offers a different approach to the design, deployment and management of networking services.Unlike SDN which separates the control plane from the forwarding plane to central control of the network, NFV optimises the network services themselves. It is important to note that SDN and NFV are not dependent on each other but are highly complementary as shown in Figure 2.2 [30, 32, 33].

The benefits of virtualising network functions include [30, 31]:

- The ability to scale network services up/down as required

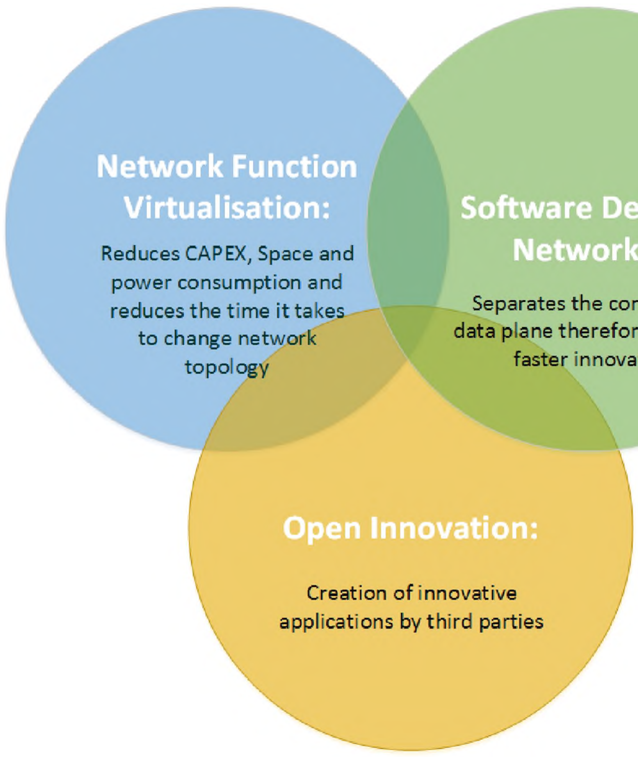

Network Function Virtualisation:
Reduces CAPEX, Space and power consumption and reduces the time it takes to change network topology
Software Defined Networks:
Separates the control and data plane therefore enabling faster innovation
Open Innovation:
Creation of innovative applications by third parties

Figure 2.2: The Relationship between SDN and NFV. Source: [30].

- Reducing energy costs by using power management features in standard servers
- Reducing Capital Expenditure (CAPEX) and Operation Expenses (OPEX).
- Infrastructure provides elasticity for capacity expansion
- Optimising network topology and configuration in near real time in response to traffic and service demand
- Decreasing time to market by simplifying the network service deployment process
- Ability to use a single platform for different applications and users
- Encourages innovation

In order to realise these benefits, some challenges that need to be addressed include [30]:

- VNFs only scale if all the functions can be automated
- Ensuring resilience to hardware and software failures

- Achieving highly portable virtualised network services that are compatible with different hardware vendors and hypervisors
- Managing and orchestrating many virtual network applications

It is important to note that network functions can be decoupled from proprietary hardware without the use of virtualisation technology. Network functions can be developed or purchased and run on standard physical servers. However, this would be at the cost of losing functionalities offered by NFV which include energy efficiency, the automation of processes such as deployment, configuration, service provisioning and lifecycle management of network functions [1, 12]. In complex networks it may be challenging to migrate to a fully virtualised environment. For such cases, it may be useful to have a transitional phase before the fully virtualised implementation as network providers can have hybrid setups where VNFs co-exist with network functions running on hardware. The efficient management of physical resources is important as physical servers have finite compute, memory and storage resources. This is especially important when wanting to have features that support dynamism such as automation and scalability [1].

2.3.1 The NFV MANO Framework

In November 2012, seven of the world's leading telecommunication providers (AT&T, BT, Deutsche Telekom, Orange, Telecom Italia, Telefonica and Verizon) chose the European Telecommunications Standards Institute (ETSI) to be the parent group of the Industry Specification Group (ISG) for NFV [1, 34]. The telecommunication industry needs standardisation organisations to guarantee interoperability between vendors and to create a common ground for making decisions on the direction of the industry [35]. The NFV paradigm brings with it new relationships within the network and it is important to have a set of guiding standards for the operation, maintenance and provisioning in the virtualised networking approach. In order to promote the growth of multi-vendor implementations, achieving interoperability for NFV solutions is a vital goal. The 3rd Generation Partnership Project (3GPP) standards body is in liaison with ETSI for the development of globally accepted standards for the management of virtualised network functions. Figure 2.3 shows the architecture of the ETSI NFV Management and Orchestration (MANO) framework. The three key elements in the architecture are: VNFs, Network Function Virtualisation Infrastructure (NFVI) and NFV MANO. The core components of the NFV MANO aspect of the framework as shown in Figure 2.3 are the NFV

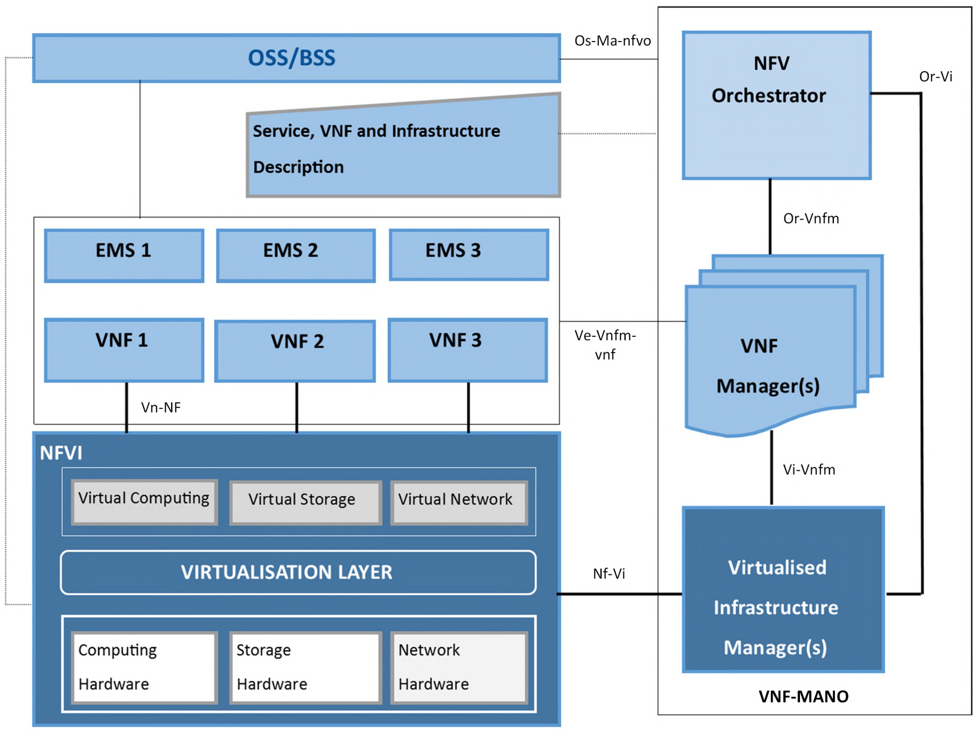


Figure 2.3: ETSI NFV MANO Architecture. Adapted from [8].

Orchestrator (NFVO), VNF Manager (VNFM) and Virtualised Infrastructure Manager (VIM). These functional blocks of the framework are discussed in a bit more detail below.

Operations Support System/Business Support System (OSS/BSS)

The OSS/BSS are operations and business support functions that need to exchange information with the NFV MANO framework but are not explicitly part of the framework. These operations are important for the provision of specific network services and operators have to ensure that they are implemented [8].

Element Management System (EMS)

The EMS handles the FCAPS [1] management functions for a VNF. These include [8]:

- Configuring the network functions provided by the VNFs,

[1] A network management framework that uses five levels namely, fault-management (F), configuration (C), accounting (A), performance (P) and security (S).

- Handling fault management for network functions,
- Managing VNF security,
- Doing the accounting for VNF usage.

The EMS often works together with the VNFM to fulfill some of the functions.

Virtual Network Functions (VNFs)

NF are network components with well-defined functions and interfaces, for example, DHCP servers, routers, firewalls, etc. A VNF is simply a NF implemented on virtual infrastructure. Each VNF may be made up of more than one internal component and for this reason, a VNF can be implemented across multiple virtual machines with each virtual machine implementing a single component. Naturally, the service experience offered to users should be the same regardless of using VNFs or NFs running on dedicated hardware [1].

NFV Infrastructure (NFVI)

NFVI is the union of hardware and software components used to create the environment in which the VNFs are deployed [31]. These resources are [8]:

- Compute resources such as machines (physical or virtual) with CPU and memory
- Network resources including networks, forwarding rules, subnets, ports, addresses and connections between and within VNFs.
- Storage resources which includes file-system and block storage

The hardware is made up of the commercial-off-the-shelf (COTS) computing, network and storage hardware. The software components are the virtual abstractions of the compute, network and storage resources. These virtual resources are achieved through the use of a virtualisation layer which makes use of hypervisors [1].

NFV Management and Orchestration (NFV MANO)

The ETSI NFV management and orchestration framework (ETSI NFV MANO) manages the NFVI and orchestrates resource allocation to VNFs and network services (NSs) [8]. The framework aims to encompass the functions needed to execute VNFs and NSs effectively and is made up of a Virtualised Infrastructure Manager (VIM), VNF Manager (VNFM) and NFV Orchestrator (NFVO) [31].

VIM: The VIM orchestrates NFVI compute, networking and storage resources within a single domain [31]. A VIM may be specialised to handle a specific NFVI resource or it can simply handle multiple types of NFVI resources. It is also the one that manages the capacity of the virtual resources available and reports the usage [8].

VNFM: The VNFM is responsible for managing VNFs and their lifecycles. It communicates with the VNF for provisioning, configuration and fault management [31]. Each VNF may be associated with its own VNFM, however, it is possible to have a single VNFM managing multiple VNF instances. Some functionalities of the manager include [8]:

- Instantiating the VNFs,
- Updating/upgrading VNF software,
- VNF instance scaling,
- Termination of a VNF instance.

NFVO: The NFVO has two main responsibilities, the first being to orchestrate NFVI resources across multiple VIMs and the second being the management of the Network Services being provided. Together with the VNFM, it manages the instantiation of VNFs. It also validates and authorises requests for NFVI resources from VNFMs. The NFVO is also responsible for NS instance automation management [8, 31]. Some of the specific functions that the orchestrator is responsible for are [11]:

- Managing the lifecycle of network services,
- Ensuring that Key Performance Indicators (KPIs) are met,

- Ensuring end-to-end reliability,
- Ensuring end-to-end fault management,
- Maintaining system consistency during scaling.

2.3.2 Virtual Machines

The components that make up the network functions in a network service are implemented on virtual machines. A virtual machine (VM) is the software implementation of a computing environment or machine architecture [36]. It is typically a computer image file that behaves like an actual computer when it is running [37]. Virtual machines have the advantage of being able to run their operating systems without worrying about any specific details of the underlying hardware platform [38]. Virtual machines are increasingly popular as they can fulfill a variety of tasks [36, 37]:

- Each virtual machine is sandboxed from all the other virtual machines on the platform, ensuring that the software inside one virtual machine can not affect the underlying environment. This increases fault tolerance, protects the system from intrusion and isolates it from untrusted code.
- Operating costs are reduced by lessening the number of servers needed through consolidating different computing environments into virtual ones. The use of virtual machines is one of the ways in which NFV is able to reduce operating costs.
- Virtual machines are able to fully emulate platforms that are different from the one they are running on, which expands the variety in the software that a user can run using the same infrastructure. As a result, NFV is able to run different network functions on the same infrastructure.
- They are useful as test bed environments due to their isolated nature and ease in deployment.

All these characteristics of virtual machines is the reason why they are at the core of NFV.

2.3.3 NFV MANO Implementations

There are several NFV MANO aligned implementations available, a few of which will be discussed in this section. This thesis focuses on free and open source implementations. As telecommunication technologies tend towards 5G, standardisation and open source are becoming more complementary as tools to promote faster innovation [39]. Most implementations offer a partial implementation of the NFV MANO framework by focusing on specific aspects of the NFV MANO framework.

Open Baton

Open Baton [11] is an open source project developed by Fraunhofer FOKUS and Technische Universität Berlin. It is a customisable NFV MANO-compliant implementation that is able to orchestrate network services across heterogeneous NFV environments. It manages a wide variety of VNFs through a generic VNFM and a generic EMS. It is also possible to integrate with other existing VNFMs using a plug-and-play model which exposes Advanced Message Queueing Protocol (AMQP) and RESTful API. It also provides SDKs in Java, Python and Go. Some of the main features of Open Baton include:

- It provides autoscaling and fault management by using information provided by the monitoring system which sits at the NFVI level,
- Support for multi-site NFVI,
- Support for network slicing at infrastructure level by making use of SDN technologies for the isolation of network services that share the same physical resources.

Figure 2.4 shows the Open Baton architecture which includes components defined in the ETSI NFV MANO framework. The main components in the Open Baton framework are [11]:

- **NFVO:** implemented according to the ETSI MANO specification.
- **Generic VNFM and Generic EMS:** together they manage the lifecycles of VNFs. The Generic VNFM can manage one or more VNFs and they can be of the same type or different types. It is an intermediate component the sits between

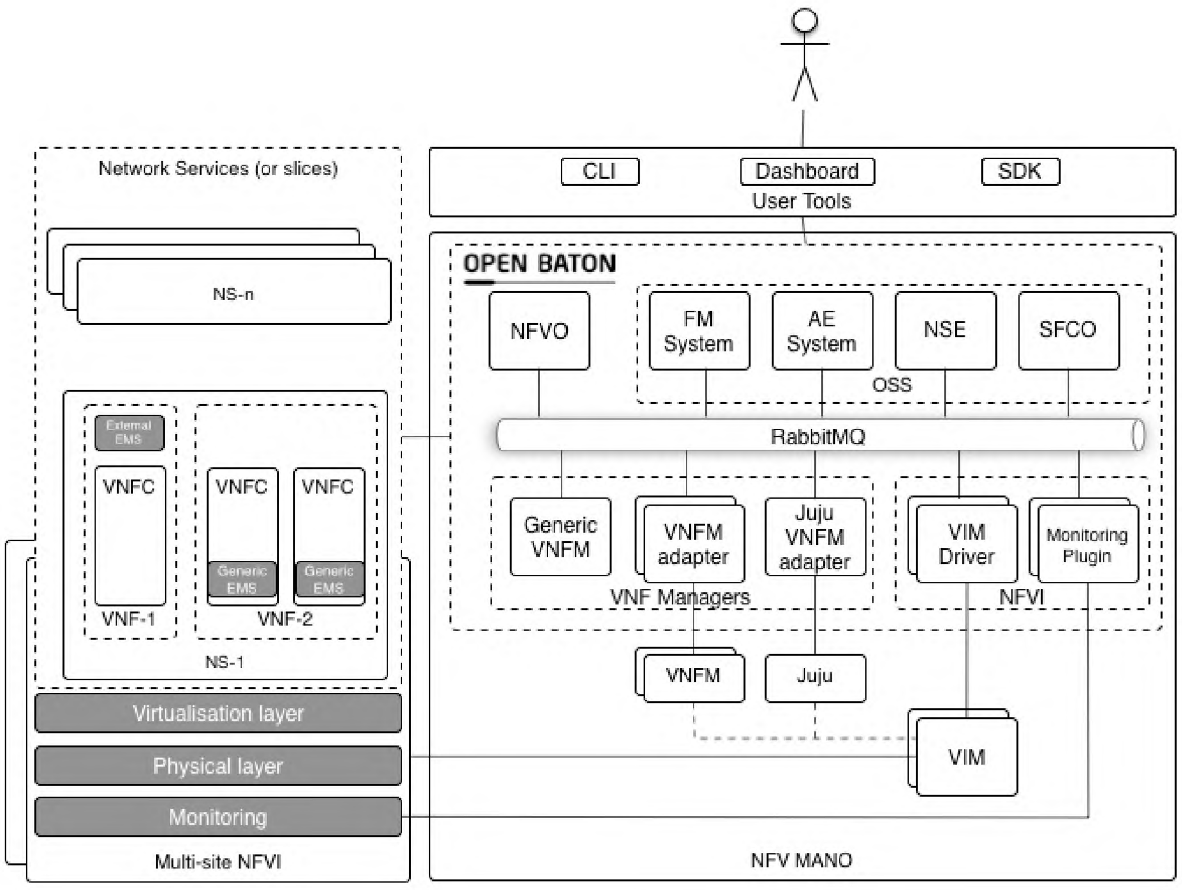


Figure 2.4: Open Baton Architecture. Source: [11].

the NFVO and the VMs on which the VNF software is installed. It integrates with the Open Baton Element Management System (EMS) which works as an agent inside the VMs and executes scripts from a VNF package or VNF descriptor (VNFD). The VNFM also sits between the NFVO and the EMS. The VNFM sends commands to the EMS running in the VM and the EMS executes the commands in the VNF component (VNFC).

- **Juju VNFM Adapter:** used for deploying Juju charms or Open Baton VNF packages. The Juju VNFM is not currently fully interoperable with the Generic VNFM.

- **VIM driver:** supports different types of VIMs without having to change the orchestration logic. OpenStack is the mainly supported VIM driver but others include Amazon and Docker. VIM drivers are implemented using the Remote Procedure Call (RPC) mechanism.

- **Docker VNFM and VIM driver:** used to instantiate containers on the Docker Engine.

- **Monitoring Plugin:** to integrate with Zabbix as a monitoring service. The Zabbix plugin is an open source project that provides an implementation of two interfaces of the VIM. The interfaces are *VirtualisedResourceFaultManagement* and *VirtualisedResourcePerformanceManagement*, both are based on the ETSI NFV MANO specification. The consumers (VNFMs, NFVO, AutoScaling Engine, Fault Management System) communicate with the Zabbix Plugin using JSON, allowing consumers to be written in any language. The consumers are independent to the monitoring system as shown in Figure 2.5. To avoid the need to contact the Zabbix Server every time a metric is required, values are cached and updated periodically.

- **Libraries:** these come in Java, Python and Go. They can be used to build your own VNFM. The openbaton-libs project contains modules that are shared folders used by the different projects in the Open Baton framework.

- **Network Slicing Engine (NSE):** to ensure QoS for the NS based on configurations specified in the Descriptors provided by the NFVO. It is implemented in java and uses the spring.io framework. The NSE uses a plugin mechanism to allow the necessary drivers to configure QoS. The only supported driver so far is the neutron driver for configuring QoS in OpenStack.

- **Event Engine:** to handle the execution of lifecycle events. Open Baton can be extended by writing external modules that react to certain events happening in the system. The event mechanism allows the user to register the endpoint they want to receive the event on. The currently supported kinds of endpoints are REST and AMQP.

- **Autoscaling Engine:** for automatic runtime management of VNF scaling operations. It provides an NFV-compliant AutoScaling Engine (ASE) implemented in java and like the NSE, also uses the spring.io framework. The ASE uses a plugin mechanism to allow usage of the preferred Monitoring System. By default, Open Baton uses Zabbix as the monitoring system.

- **Fault Management System:** for automatic runtime management of faults at different levels. This is the Open Baton Fault Management System (openbaton-fms) which is an external component to the NFVO. It communicates with the NFVO via the Open Baton SDK and RabbitMQ. It handles alarms from the VIM and uses them to perform actions through the NFVO.

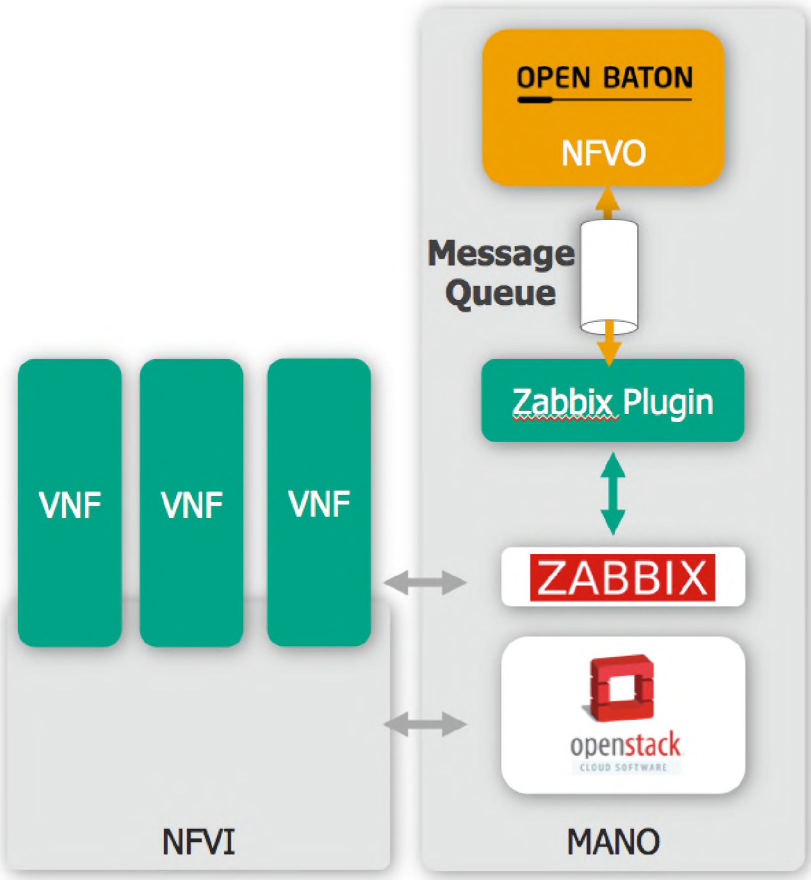


Figure 2.5: Zabbix Monitoring Architecture. Adapted from: [40].

Cloudify

Cloudify [41] is an open source cloud orchestrator designed to manage cloud environments at an infrastructure and application deployment level. It handles operations such as automating the deployment, configuration, healing and scaling of applications and NSs in cloud and stack environments. It can manage different types of cloud environments including the hybrid cloud. Cloudify has advanced management features such as monitoring, auto-scaling, self-healing, multi-tenancy and policy management.

Cloudify makes use of a compute host called the Cloudify Manager and runs the Cloudify management service. **Blueprints** which are human-readable YAML configuration files that describe the cloud, network tools and applications to be deployed. The blueprints are uploaded on to the Cloudify manager before they can be used to deploy entities. Cloudify also makes use of **workflows** which are written in Python and define the policies

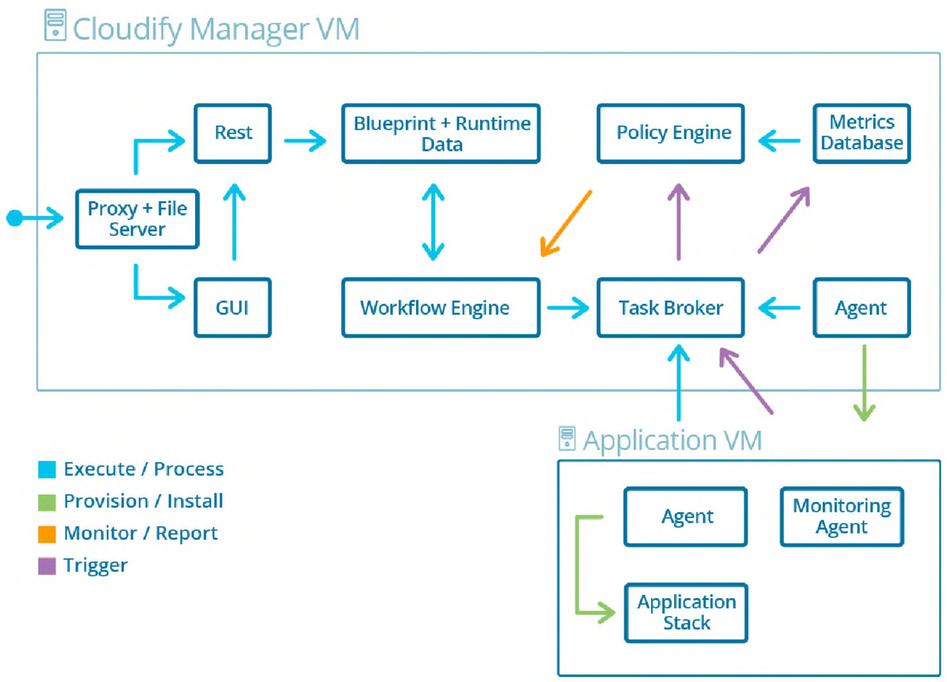


Figure 2.6: Cloudify Manager Flows. Source: [41].

for triggering tasks such as automated healing and scaling. It is also possible to define custom policies that are environment specific. Workflows determine which tasks are to be executed and when. Where tasks are operations implemented by a plugin or running arbitrary code. Workflows are deployment-specific and workflows for each deployment are defined in the blueprint. Communication between Cloudify and external services is enabled by the use of plugins. For example, if a blueprint defines an OpenStack deployment, an OpenStack plugin will be required.

The blueprint is processed by a **core engine** that automates actions to create, connect, update or scale elements in the cloud environment. It allows multiple clouds, configuration managers, serverless functions and applications to be connected and managed through a customisable UI. A **generic automation engine** provides a variety of tools used to provide services such as applications, databases and network services. Figure 2.6 shows the interaction between the Cloudify manager and the VMs it manages. In the context of the ETSI NFV MANO framework, Cloudify fulfills the roles of a VNF manager and NFVO.

2.4 Summary

The cloud computing paradigm introduced a new approach to how computing and IT related services could be provided. This led to the emerging telco cloud which is a cloud environment tailored for telecommunication services that require higher levels of QoS and QoE as compared to standard cloud services. NFV is the process by which network functions that were implemented directly on dedicated hardware components are decoupled from the hardware so that they can be implemented as virtual functions on standard hardware. NFV introduces a new approach to networking and in order to provide network operators with a set of standards for implementing NFV, the ETSI NFV Management and Orchestration (MANO) framework was developed. The ETSI NFV MANO framework has three main components namely: NFV orchestrator (NFVO), VNF manager (VNFM) and the virtualised infrastructure manager (VIM). There are various implementations aligned with the ETSI MANO framework, this chapter looks at Open Baton and Cloudify. Both Open Baton and Cloudify are open source resources satisfying the NFVO and VNFM roles in the ETSI MANO framework.

Chapter 3

Literature Review: IP Multimedia Subsystem - IMS

Network operators offer a variety of services but over the years, there has been a great increase in demand for multimedia services. This research focuses on the provisioning of IP Multimedia Subsystem (IMS) services in a virtualised manner in a Telco Cloud environment via open source components. Therefore, the IMS network and the different IMS network implementations are discussed in this chapter.

This research aims to investigate the readiness of open source software to build a Telco cloud. As such, the IMS core network was chosen as the key technology for investigation because many operators have implemented IMS in a virtualised manner, making it easier to focus on implementing the telco cloud platform and not on virtualising the network services.

IMS is a complex environment designed to enrich communication by providing multimedia services such as audio, video, image, text and data across different types of networks [42, 43, 44].

The requirements for IMS as a framework according to [2] are:

- The ability to establish IP multimedia services.
- The ability to negotiate good Quality of Service (QoS).
- The ability to provide support for roaming

- The ability to provide support rapid service creation
- The ability to inter-work with the Internet and circuit-switched networks
- The ability to support for operators to impose control over the services delivered to users

For good usability, operators need to be able to provide users with the best possible QoS.

3.1 IMS Core network

In this work we are interested in, the IMS "core network" and not the full IMS network. The core IMS network is shown in Figure 3.1, which depicts the minimum set entities needed to run a core IMS network. The set comprises Call Session Control Functions (CSCFs) and at least one Home Subscriber Server (HSS). There can be more than one of each entity for redundancy and scalability [45]. Each entity provides specific functionality within the network as discussed in more detail below.

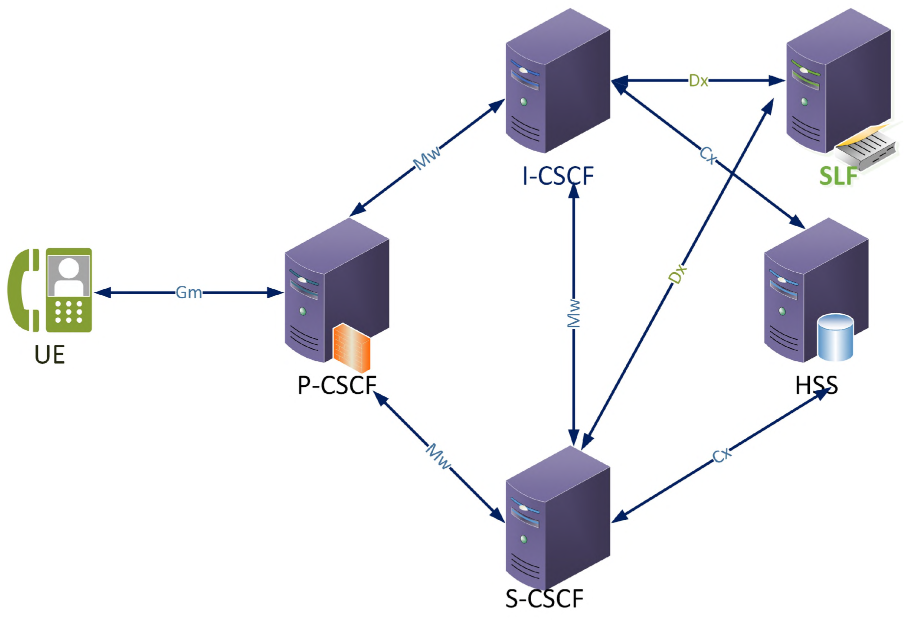


Figure 3.1: IMS Core Network Architecture. Sources [46].

3.1.1 Proxy-CSCF

The Proxy-CSCF (P-CSCF) is the first point of contact between the User Equipment (UE) and IMS network. It acts as the SIP inbound/outbound proxy server and routes requests and responses appropriately. The P-CSCF verifies the correctness of the SIP messages it receives. SIP messages are text-based, which makes them fairly large. To reduce the transmission time, the P-CSCF comes with a compressor/decompressor. SIP messages are compressed before being transmitted to the rest of the network and are decompressed at the destination [2, 47].

3.1.2 Interrogating-CSCF

The Interrogating-CSCF (I-CSCF) is at the edge of the administrative domain and finds the hop that a SIP message needs to make. It has interfaces to the HSS and SLF so that it can retrieve user information and route the message correctly, often to a Serving-CSCF, which is described below [47].

3.1.3 HSS

The HSS is a repository for user information. It stores user-related subscription information that is required to support multimedia sessions. This information includes location information, profile information (services that the user is subscribed to) and security information (for authentication and authorisation) [2, 47].

3.1.4 Subscriber Locator Function

If an IMS network has more than one HSS, an SLF is needed to resolve the HSS that a user's data is stored in. Each particular user's information is found in one HSS. An SLF is a simple database used to map user addresses to HSSs. When the SLF is queried with the user address as input, it returns the HSS that the user's information is stored [47].

3.1.5 Serving-CSCF

The Serving-CSCF (S-CSCF) is part of the signalling plane. It acts as a SIP registrar and performs session control. It maintains the link between a user's location and their SIP address of record. It communicates with the HSS for the following reasons [2]:

- Downloading user profile from the HSS
- To inform the HSS that it is the S-CSCF assigned to the user for the duration of that session.
- Downloading authentication information of a user trying to access the IMS network.

3.1.6 Communication Protocols

The public Internet is prone to delays, packets arriving out of order and packet loss and the IMS aims to fix that as a carrier grade network should. IMS makes use of SIP as the session control and signaling protocol between the network entities [2, 42]. The IMS architecture in Figure 3.1 shows SIP-based reference points and Diameter-based reference points used for communication between the network entities. The SIP reference points are Gm and Mw, implemented between the UE and CSCFs. The Diameter reference points are Cx and Dx and are implemented for communication with the HSS and SLF [47].

3.2 IMS Core Implementations

There are a number of implementations of the IMS core network readily available for deployment. This section will look at some of the main options.

3.2.1 Open IMS Core

The Open IMS Core project is an open source IMS implementation providing IMS CSCFs and a lightweight HSS. The Open IMS Core project was started by Fraunhofer FOKUS in 2004 and later taken over by Core Network Dynamics in 2015. The components provided by the project are shown in Figure 3.2 and are based on open source software such as the

SIP Express Router (SER) and MySQL. Open IMS Core is not intended for commercial use but provides an IMS core implementation for IMS application prototyping and IMS research test beds [46]. The components that come with the Open IMS Core project are P-CSCF, I-CSCF, S-CSCF and HSS. These are a fairly straight forward match with the components explained in Section 3.1.

NFV introduces flexibility in the way in which IMS can be deployed. The following section discusses an implementation of IMS that was developed for the Cloud and virtualised environments.

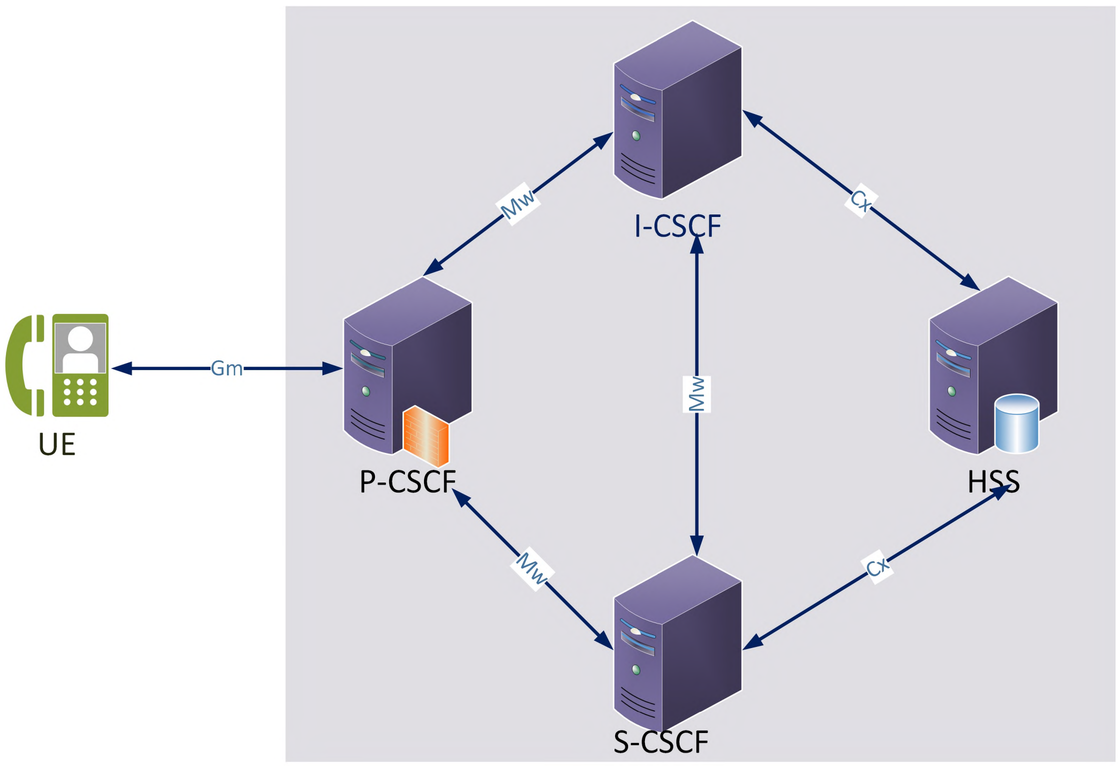


Figure 3.2: Open IMS Core Architecture. Source [46].

3.2.2 Kamailio IMS

Kamailio is an open source SIP server that can be used as a SIP load balancer, SIP proxy server, SIP registrar, SIP application server and SIP redirect server. It is a server that routes different kinds of SIP packets. Most of its features came from SIP Express Router (SER) server. It provides Java, Python, Perl, C# and JavaScript programming interfaces [48].

The core features of Kamailio include:

- SIP message parser which is implemented as an incremental parser.
- Stateless forwarding
- Configuration file parser and interpreter. The configuration file contains init and runtime instructions.
- DNS and transport layer management. It has an internal DNS cache for DNS-based load balancing and failover.
- Memory management since Kamailio is a multi-process application that needs shared memory. The memory manager creates private and shared memory chunks during start-up.
- Timer API
- RPC control interface API. Control interfaces allow communication with th Kamailio server for administrative functions.
- A locking system that uses machine specific code to improve speed.

Kamailio provides different functionality that can be accessed by loading different modules. These include SIP transaction management in the form of stateful processing, configuration file debugger that is interactive, load balancing and stateless replying [48].

Kamailio can be also used to build an IMS network. It has support for the Diameter protocol which is used in the IMS network. Most importantly, it supports the implementation of the IMS core servers namely, P-CSCF, I-CSCF and S-CSCF. Configuration files for a stripped-down version of IMS are available for building an IMS platform using Kamailio. The implementation setup for Kamailio IMS is very similar to OpenIMSCore. It is made up of the IMS core servers and an HSS server from the Fraunhofer OpenIMSCore [48].

3.3 Project Clearwater

Project Clearwater is also an open source implementation of the IMS core network developed by Metaswitch Networks [1]. It is designed for deployment in scalable Cloud environments providing SIP-based video, voice and messaging services. It was designed to provide

[1]https://www.metaswitch.com/

a telco-grade communication network solution and since it was built for the Cloud, it is suited for deployment in an NFV environment [49]. Figure 3.3 shows the architecture of Clearwater. It is structurally different from that of the standard IMS core network but it fulfills the same purpose. This is evident with a more detailed understanding of what each node does. Below is an explanation of the functions of each entity [49].

- **Bono** - acts as a SIP edge proxy and provides a SIP IMS Gm interface to clients. It provides the entry point into the Clearwater system much like the Proxy-CSCF. It is horizontally scalable and when there is more than one Bono node, client connections are load balanced across them. During the registration process, the client is bound to a specific Bono node unless the connection fails or there is client error.
- **Sprout** - is a combined SIP registrar and authoritative routing proxy. It fulfills the roles of both the Interrogating-CSCF and Serving-CSCF. There is a load balancing mechanism across all the nodes in the Sprout cluster so that there are no long-lived associations between a client and a specific Sprout node. It uses web services interfaces to communicate with Homer and Homestead (see below) in order to get user profiles and authentication data. It also maintains API communication with the Vellum entity to store registration data and run timers.
- **Dime** - runs a service called Homestead which is an HSS cache and has a web service interface to Sprout for access to user profile information and authentication credentials. If there is an external IMS HSS, it communicates with it over the Cx interface. In the case where there is no external HSS included, the subscriber data is stored in the Vellum node.
- **Vellum** - stores all long-lived state data. It uses a cloud-optimised Cassandra database to store authentication credentials needed by Homestead. No long lived state data is stored on the other production nodes, making dynamic scaling easier and quicker, while also lessening the impact from loss of a node.
- **Homer** - is an XML Data Management Server (XDMS) that stores MMTel (Multimedia telephony) service settings.
- **Ellis** - is a sample provisioning portal that provides password management and self sign-up. It is not part of the core Clearwater deployment but it is meant to make the system easy to use immediately and also a useful way to test the deployment once installed.

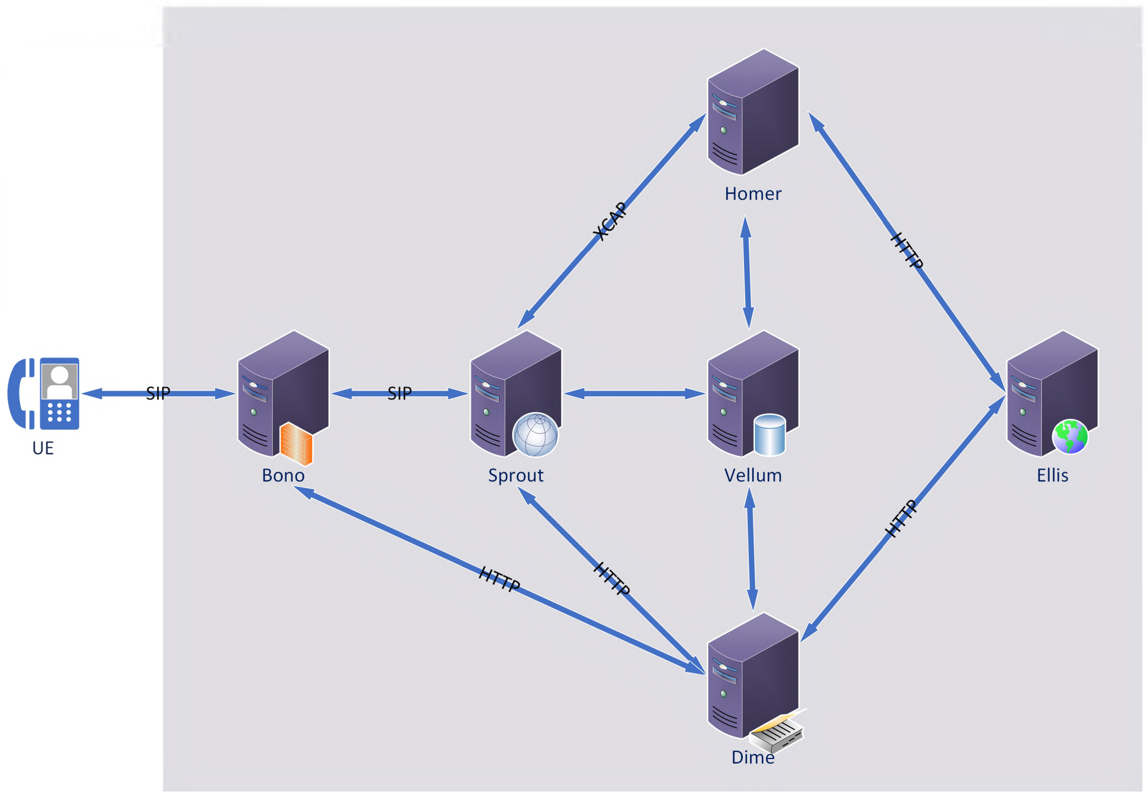


Figure 3.3: Project Clearwater Architecture. Adapted from [49].

The architecture of Clearwater follows that of scalable web applications. The interfaces between front-end SIP components and backend services use RESTful web interfaces. Project Clearwater is horizontally scalable and uses a stateless load balancer. DNS load balancing ensures that the cluster can adapt to changes in total load. Connection pooling and statistical recycling of connections are used so that there is even distribution of load as nodes are added and removed [49].

Reliability was traditionally achieved in telco platforms by using one-to-one, low-level data replication. This method does not work well for cloud and virtualised environments. Hence, Clearwater uses a different approach and achieves reliability by keeping most components stateless and using reliable and scalable data stores for long-lived state data [49].

3.4 IMS Performance and Benchmarking

Baseline testing refers to the process of running tests to get information about the general performance of the system. **Benchmarking** on the other hand compares the performance of the system with regards to some given industry standard. With regards to an IMS network, the purpose of testing is to measure the performance and behaviour of a system as the load it has to handle increases. This is often as a result of an increase in users that need to be served at the same time or changes in the type of load that the system has to handle. ETSI developed IMS/NGN Performance Benchmark which defines a performance benchmark specification consisting of three parts [50]:

1. The general Benchmark environment.
2. Providing Benchmark use-cases and subsystem configurations.
3. Defining the benchmark tests in terms of traffic sets and traffic profiles.

A benchmark test comprises of a Test System (TS) which simulates several User Equipments (UEs) and the SUT which reacts to the request coming from the TS. The traffic profile defines the use case scenarios which are to be executed. A scenarios is an interaction sequence occurring between two users. Benchmarking uses Scenario Attempts Per Second which is the average number of scenarios being initiated by the TS [50].

Several works have looked at the benchmarking the performance of IMS networks and SIP servers. The performance of SIP servers is can be affected by a number of factors such as using stateless vs. stateful proxying, UDP vs. TCP and using MD5-based authentication [51]. The experimental setup in [51] uses three main SIP scenarios: registration, proxying and redirection. The SIPp load generator was used and the metrics of interest where: throughput, CPU profiles, success rate and latency. The results show that throughput for stateless proxying using UDP and no authentication yielded best results.

When benchmarking an IMS system, a pre-registration phase and stir-phase can be defined which come before the run phase. Where the pre-registration phase is the initial phase before any users have been registered with the SUT and the stir phase is when the SUT is initialised by registering users and randomising subscriber data. Lastly, the run phase is when actual stress tests are done on the SUT using a predefined number of SAPS and increasing this number at time intervals until the number of Inadequately Handled Scenarios (IHS) exceeds a set threshold [50]. A benchmark done in [50] using IMS Bench SIPp to test the performance of an SUT hosting an Open IMS Core network concluded that the Design Objective Capacity (DOC) of the deployed testbed was 120cps. The IHS for each scenario was 0.1%. The authors also monitored CPU load and memory load of the SUT during the benchmark test.

3.5 IMS Bench SIPp

SIPp is an open source traffic generator for the SIP protocol for testing purposes. It uses user agent scenarios, User Agent Client (UAC) and User Agent Server (UAS), where the UAC is a peer-to-peer entity that sends SIP requests and the UAS is the entity that responds to the SIP requests sent by the UAC. The establishment and release of calls is done using the INVITE and BYE methods. Scenario files describe the call flows. Statistics such as the call rate and round trip delay are displayed dynamically while the tests are running [52]. Both UDP and TCP traffic are supported as shown in Figure 3.4. SIPp is a useful benchmarking tool that is able to emulate thousands of user agents making calls in a SIP system.

Figure 3.4 shows a high level overview of the IMS Bench testing system which consists of a manager and one or more SIPp load generators that execute various benchmark scenarios. There can also be one or more optional system monitoring agents that monitor and report CPU and memory utilisation. The System Under Test (SUT) is the system whose performance is being tested or benchmarked and is assumed to be working correctly when the benchmark operation is executed [52]. The manager has the following responsibilities:

- Configuring the SIPp test systems by uploading scenarios and sending configuration data.
- Executing steps of the benchmark run including, setting the ratios of each scenario in the scenario mix and setting the rate of scenario attempts as defined by the benchmark configuration.
- Monitoring the rate of failure so that the benchmark is stopped when the threshold is exceeded.

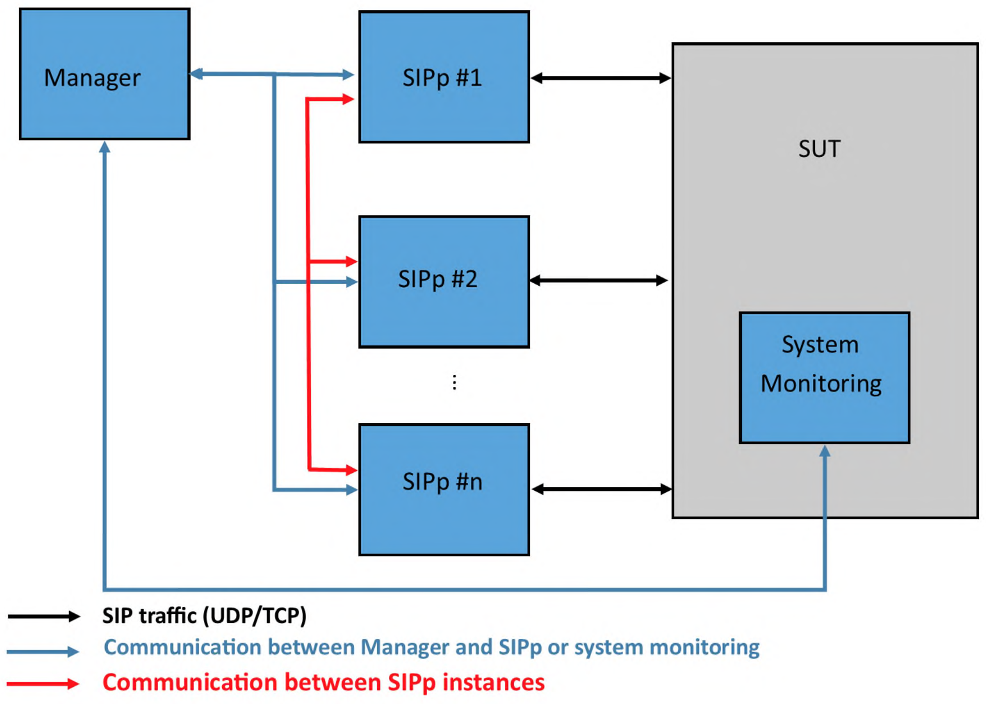

Manager
SIPp #1
SIPp #2
SIPp #n
SUT
System Monitoring
SIP traffic (UDP/TCP)
Communication between Manager and SIPp or system monitoring
Communication between SIPp instances

Figure 3.4: High Level Overview of IMS Bench testing. Adapted from [52].

- Logging system resource utilisation (CPU and memory).

The results of the benchmark are saved for future analysis in a csv file. The results from this csv file can then be used to create plots for analysis and visualising the data.

3.6 Summary

This chapter introduces the IMS network, specifically the IMS core network. IMS is a complex environment that provides real-time and multimedia services to users. Two open source implementations of the IMS core network are discussed namely, Open IMS Core and Project Clearwater. The architectures of the two IMS implementations are explained and some differences and similarities highlighted. The Open IMS Core implementation is more mature but on the other hand, the Project Clearwater implementation was designed for deployment in the cloud. A popular tool for testing the performance of an IMS network is IMS Bench SIPp. IMS Bench SIPp generates SIP traffic and uses different scenarios to test the performance of the network. The data gathered from the benchmarking and testing is saved in a csv file and can be used to generate plots for further analysis.

Chapter 4

Literature Review: OpenStack

This work aims to build a cloud platform through open source components that requires having access to computing, storage and networking resources. To fulfill this need, one of the most prominent open source solutions is OpenStack. Because of its centrality, a more detailed discussion will be presented in this chapter. OpenStack is a set of open source software tools that can be used to build and manage cloud platforms for both the public and private cloud. It is a cloud operating system that is managed by the OpenStack Foundation and controls compute, storage and networking resources. Since it is open source, one of the benefits is that it has thousands of developers working on it to make it a secure and robust product [9, 53].

4.1 OpenStack- IaaS

OpenStack is an Infrastructure as a Service (IaaS) platform, meaning that it provides infrastructure that allows virtual machines to be deployed "on the fly". This makes horizontal scaling easier, and if more resources are needed, new instances can be added [9, 54]. Voice over IP (VoIP) services such as IMS can take advantage of the robust services offered by OpenStack. OpenStack can therefore be used as the backbone for VoIP/IMS solutions that are able to efficiently manage resources for dynamically changing workload demands without compromising quality of service. The ability to scale resources up and down is a key component for scalable IMS platforms.

4.2 OpenStack Core Services

OpenStack is made up of different projects so that it can meet a variety of needs. The OpenStack community is also able to add components to fit their needs. However, there are nine projects that have been identified as the core of OpenStack and these are distributed as part of the OpenStack system. These projects are discussed further in this section.

All services get authenticated through the same Identity service. A service is made up of several processes and at least one of those processes is an API process which facilitates integration with other services. The processes in each service communicate with each other using the OpenStack messaging bus [9]. Figure 4.1 shows the conceptual architecture of how the projects interact with each other.

4.2.1 Nova

Nova is the primary computing engine for OpenStack. Nova is an important component that hosts and manages OpenStack cloud computing systems and is able to scale horizontally. The compute service interacts with other services in order to be able to launch instances. It interacts with the Dashboard service, Horizon, for access to the user and administrative user interface; the Identity service, Keystone, for authentication and the Image service, Glance, for access to server images [9].

4.2.2 Neutron

Neutron is the OpenStack networking service that provides networking capabilities by supporting the creation and attachment of interfaces devices. It ensures that components are communicating efficiently. It interacts closely with the compute service in order to provide connectivity to instances within OpenStack. The OpenStack architecture is made more flexible by accommodating various networking equipment through the use of plugins. The networking service is made up of the following components [9]:

- **neutron-server** - responsible for routing API requests to the correct plug-in for action.

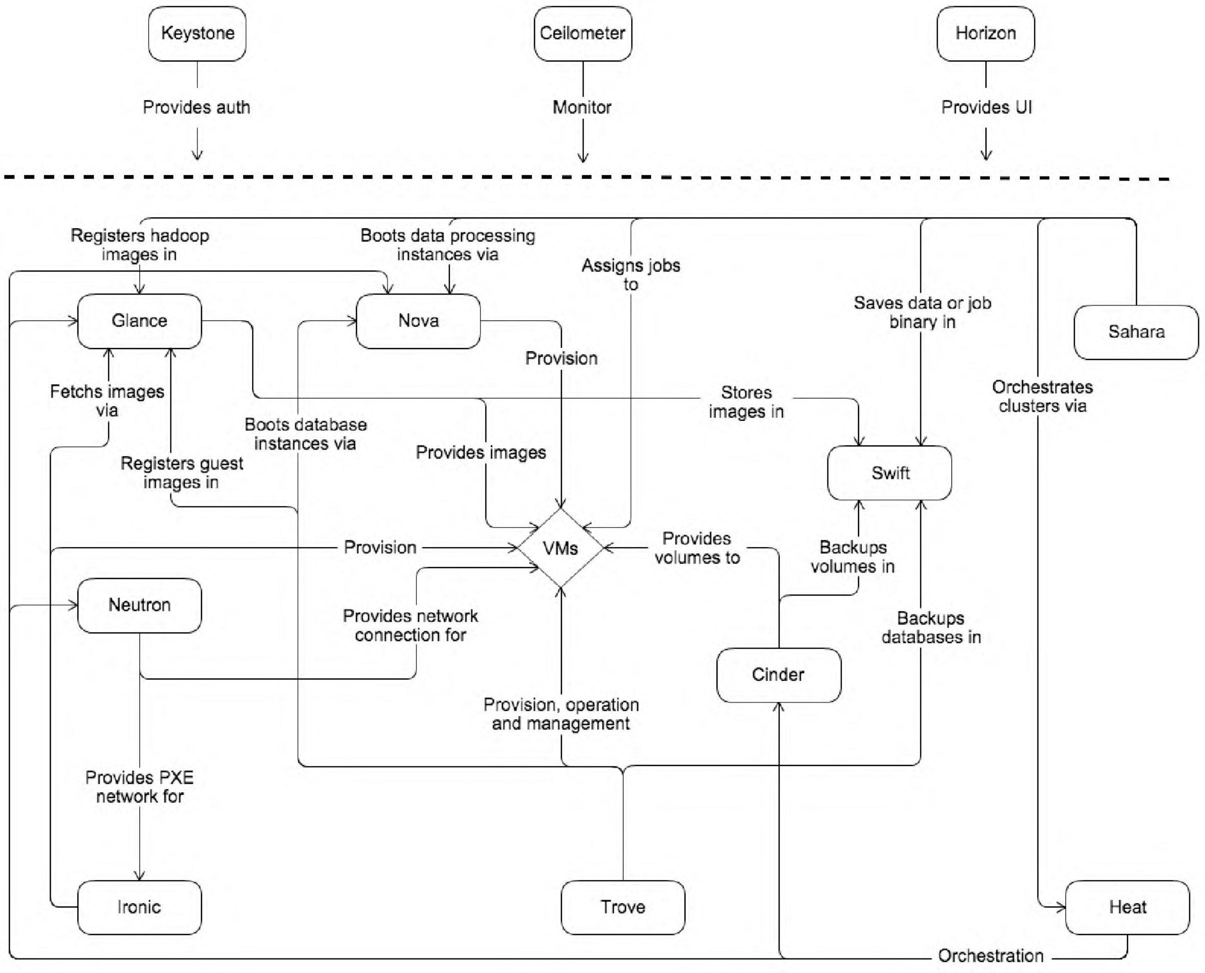


Figure 4.1: Conceptual architecture of OpenStack. Source: [9]

- **OpenStack Networking plug-ins and agents** - these make it possible to create networks and subnets as well as provide IP addresses. The common networking agents are a plug-in agent, DHCP and layer 3 (L3). Plug-ins and agents are vendor specific and are also influenced by the technologies deployed in that cloud.
- **Messaging Queue** - mostly used by networking installations to route messages between agents and the neutron server.

4.2.3 Keystone

Keystone is the identity service providing a central list of the OpenStack cloud users and the services they have permission to use. Creating a single point for managing authentication and authorisation for other services and users. It is generally the first service a user interacts with in order to be authenticated and once authenticated, the user's identity is used to allow access to other OpenStack services. For flexibility, Keystone can integrate with other external user management systems such as LDAP [9].

4.2.4 Glance

Glance is the image service providing copies of operating system images that can be used when deploying new instances. A REST API allows the user to query virtual machine image metadata and get an actual image. Server images can be stored in various types of repositories, an example being the OpenStack Object storage [9].

4.2.5 Horizon

Horizon is the dashboard service for OpenStack, essentially providing the graphical web based user interface to OpenStack. It enables administrators to have a view of resources created in the OpenStack cloud [9].

4.2.6 Cinder

Cinder is the block storage service that provides block storage to instances. It is analogous to how data is traditionally accessed from specific locations on the disk drive. The

Block Storage drivers (NAS/SAN, Ceph, iSCSI, NFS, etc) determine how the storage is provisioned to instances [9].

4.2.7 Swift

Swift is the object storage project that provides a multi-tenant object storage system that is highly scalable. The RESTful HTTP API enables it to manage large amounts of unstructured data at a low cost [9].

4.2.8 Heat

Heat is the OpenStack orchestration service that manages the entire lifecycle of infrastructure and applications in the OpenStack cloud. The orchestration engine that is implemented by Heat allows cloud applications to be launched based on text file templates. The text files are treated as code. It makes use of the AWS CloudFormation template format through the OpenStack-native REST API as well as the CloudFormation-compatible Query API [55]. As mentioned in Chapter 1, the AWS CloudFormation uses templates to model how resources are allocated automatically [21].

How Heat works

Heat provides auto-scaling capabilities using event data. The event data is provided by another service, making Heat work well with Ceilometer as Ceilometer provides the event data needed by Heat. Heat provides an OpenStack native REST API which provides compatibility with templates [9].

Heat Orchestration Templates (HOT) provide the infrastructure for a cloud platform in a text file. Listing 4.1 shows an example of HOT YAML template that can be used to create a CirrOS VM instance. The infrastructure that can be described in the templates includes: servers, security groups, volumes and Floating IPs [9]. Heat makes it possible to have a scaling group as part of the infrastructure resources in a template. Heat facilitates the auto-scaling service by integrating with Telemetry [55].

Listing 4.1: Sample HOT YAML Template

```
heat_template_version: 2015-10-15
description: Launch instance with CirrOS image using the ``m1.small``
             flavor, ``testvm-keypair`` key, and one network.

parameters:
  key_name:
    type: string
    description: key to be used.
  flavor:
    type: string
    description: flavor to be used.
  image:
    type: string
    description: image to be used.

resources:
  stack_instance1:
    type: OS::Nova::Server
    properties:
      key_name: testvm-keypair
      image: cirros
      flavor: m1.small
      networks:
      - network: 5952cae2-9868-407d-b91c-53a373408927
```

Heat Architecture

Heat orchestrates cloud systems using an AWS CloudFormation template that describes cloud applications by executing OpenStack API calls to generate or tear down cloud applications.Heat is made up of a number of components, namely [9]:

- **heat** - a CLI which communicates with the heat REST API or the heat-api to execute AWS CloudFormation APIs.
- **heat-api** - provides an OpenStack-native REST API that is responsible for processing API requests and directing them to the heat-engine over RPC.

- **heat-api-cfn** - provides an AWS Query API that is compatible with AWS Cloud-Formation. It processes AWS Query requests in the same way the heat-api tool processes API requests.
- **heat-api-cloudwatch** - an API service which is CloudWatch-like.
- **heat-engine** - the tool responsible for orchestrating when templates are launched and provides feedback to the API consumer.

4.2.9 Ceilometer

The Ceilometer service is a data collection service that makes it possible for data to be normalised across the OpenStack core components, namely: compute, network and storage. Ceilometer is part of a bigger project called Telemetry whose overall aim is to collect data on resource utilisation (both virtual and physical), store this data for analysis and to trigger actions in the case where defined criteria are met [56]. Ceilometer is an alias for the Telemetry Data Collection service which polls metering data from OpenStack services, collects event and metering data sent from OpenStack services by monitoring notifications and publishes the data to several targets. [57].

Ceilometer's Logical Architecture

Ceilometer offers two main services, namely the polling agent and the notification agent. The polling agent is a daemon that polls OpenStack services and creates meters. The notification agent is a daemon that listens for notifications that are on the message queue, creates Events and Samples using the notifications and applies pipeline actions [9].

The Ceilometer service comprises of the following components [9]:

- **A compute agent (ceilometer-agent-compute)** - which runs on each of the compute nodes and collects statistics on how resources are being utilised.
- **A central agent (ceilometer-agent-central)** - which runs on a central management server and collects resource utilisation statistics for resources that are not related to compute and instance nodes. In order to scale the service horizontally, multiple agents can be deployed.

- **A notification agent (ceilometer-agent-notification)** - which runs on the central management server(s) and gets messages from the message queues and processes them to get event and metering data.
- **A collector (ceilometer-collector)** - which is an optional service that runs on the central management server(s) and sends the collected unmodified data to a database or an external consumer. This service is optional because the notification agent can be configured to provide the same functions.

Gathering the Data

Data is collected using two methods, the notification agent and the polling agent. *Notification agents* is the preferred method for data collection. It takes messages on the notification bus and creates Ceilometer events and samples. This method is supported by the ceilometer-agent-notification which monitors the presence of notifications on the message queue. *Polling agents* is the least preferred method as it overloads API services. Polling agents poll API to collect data at regular intervals [9]. Figure 4.2 shows a representation on how data is collected in Ceilometer.

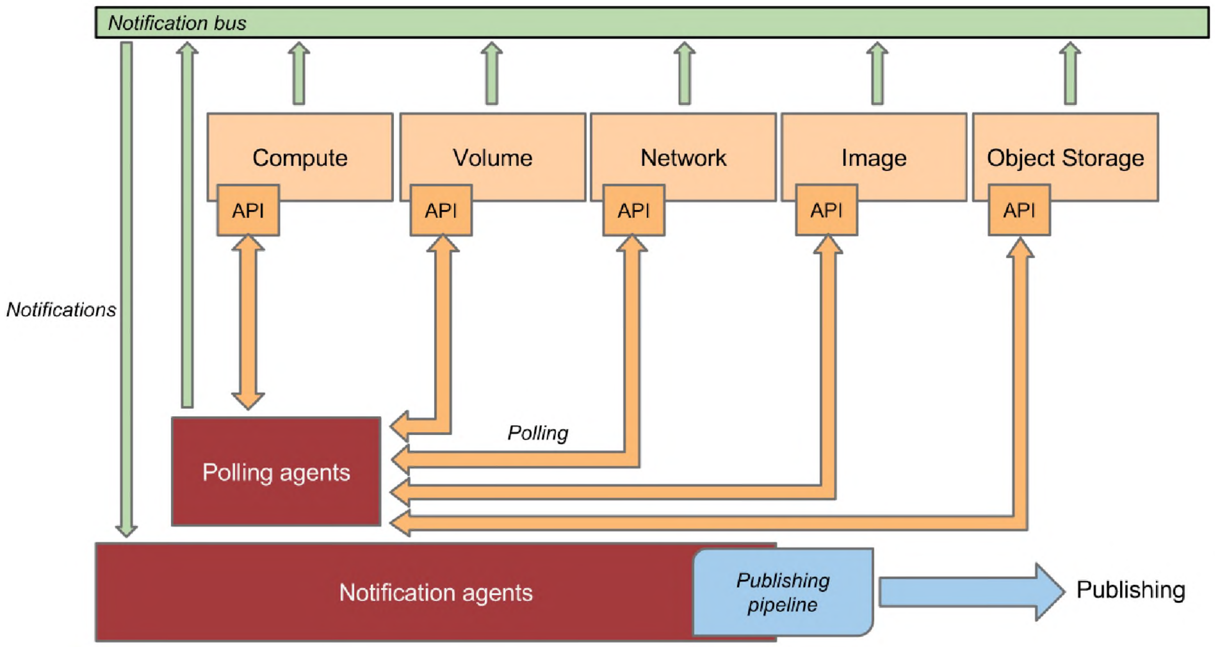


Figure 4.2: Ceilometer Data Collection. Source [9].

Gnocchi - Target for metering data

Ceilometer has performance issues when it made use of standard databases as back-end storage. The Gnocchi project addresses these performance issues.It fulfills the need for a time series database (a database with a list of aggregates[1] ordered by time) for the cloud environment.This is because capturing measurement data in time series format helps to optimise storage and querying [9]

Gnocchi is an open-source metrics and resources database designed to store metrics at a large scale while also providing easy access to the metrics and resource information . Its key features include: an HTTP REST interface, horizontal scaling, multi-tenant, resource history, structured resources and metric aggregation. Gnocchi can handle a large amount of aggregate data without compromising performance and in addition offers scalability and fault tolerance [58].

Aodh

Aodh is the Telemetry Alarm service which triggers alarms when the defined rules are broken by the collected event data [56].

The Telemetry service is made up of the following components [56]:

- **An API server (aodh-api)** - which runs on management server(s) and provides access to alarm information that is in the data store.
- **An alarm evaluator (aodh-evaluator)** - which runs on management server(s) and determines when alarms go off due to statistics crossing a set threshold over a period of time.
- **A notification listener (aodh-listener)** - which runs on a central management server and determines when alarms should be fired. The alarms are generated according to rules defined against events collected by Ceilometer's notification agents.
- **An alarm notifier (aodh-notifier)** - which runs on central management server(s) and allows alarms to be set based on evaluating the threshold for some samples.

[1]An aggregate is a data point tuple composed of a time stamp and a value

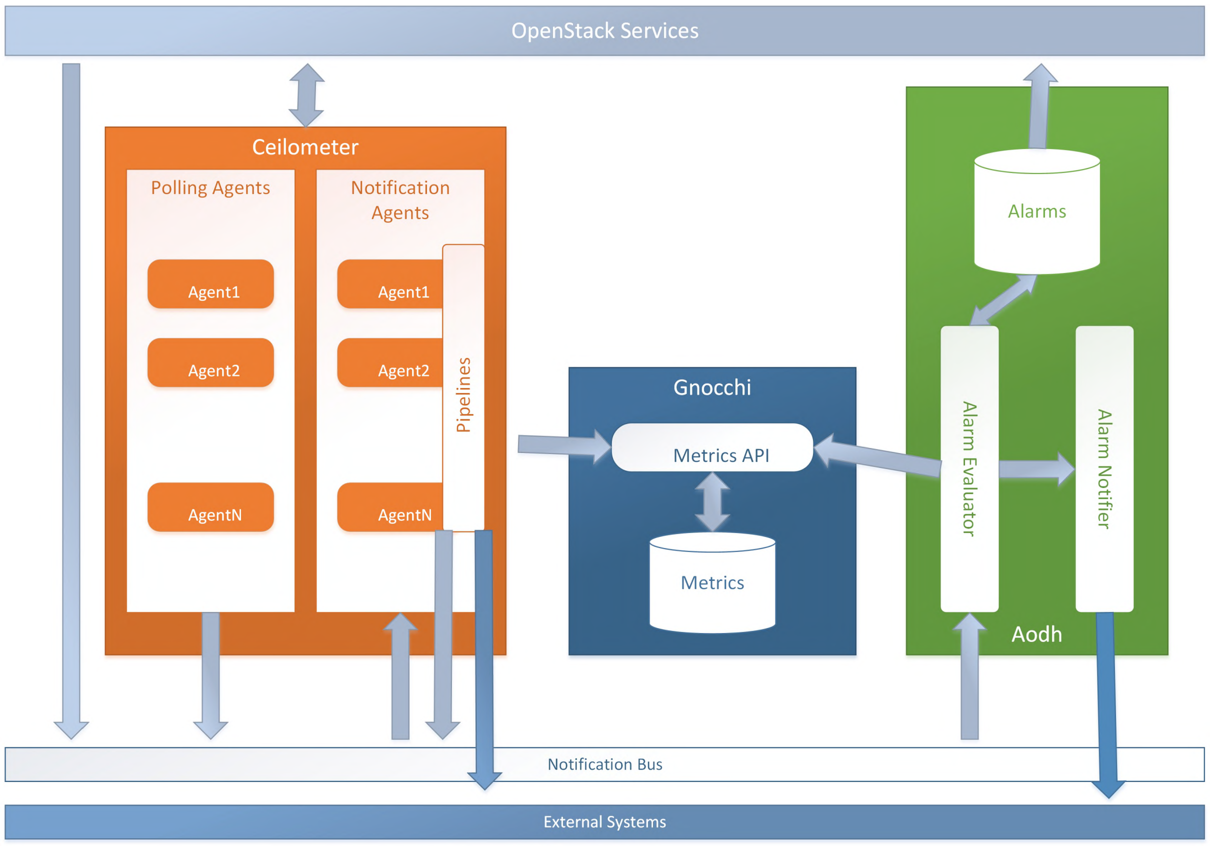


Figure 4.3: Telemetry Architecture. Adapted from: [9]

4.2.10 Telemetry: High-Level System Architecture

Ceilometer and Aodh work together to collect data across all OpenStack services and use it to create alarms.

Figure 4.3 shows a logical representation of the interaction between Ceilometer, Gnocchi and Aodh. Together these services allow for data to be collected from OpenStack services, for the data to be stored in such a way that it can be easily queried and lastly make it possible to set alarms based on event data that has been captured by the data collection service.

4.3 OpenStack Strengths and Challenges

OpenStack is gaining traction as enterprises adopt the open source solution to support their NFV projects [59]. In their paper, Vogel et al. [60] did a comparative analysis of private IaaS clouds namely, OpenNebula, CloudStack and OpenStack. Based on their experiments, they concluded that workloads running on OpenStack were the most stable. Trilio identified some reasons for the increase in OpenStack adoption to be [61]:

- **Tenant-Level Control**: Multi-tenancy makes it easier to monitor and maintain the health of the cloud as an administrator. Internal users are able to manage their own environment and consume resources according to their needs.
- **Flexibility**: OpenSatck simplifies virtualisation of network functions using NFV and provides containerised Compute, Network and Storage on demand. It scales up with ease and can run large numbers of instances. Administrators can scale up computing resources when additional capacity is needed and scale down when not needed.
- **Low Cost of Ownership**: It has no expensive software licensing fees and is compatible with commodity hardware which reduces management costs.
- **Openness and Modularity**: OpenStack is an API-driven cloud. Users and vendors build off the same APIs so services are standardised across the platform. The modular architecture helps to avoid bottlenecks using a plug-n-play setup. This also provides the opportunity to minimise vendor lock-in. This openness and modularity makes evolving and expanding a bit more seamless.
- **Agility**: The self-service portal takes away the need to wait for centralised IT to get to your ticket in the workflow.
- **Mature Community**: The OpenStack Foundation is driven by experienced vendors who can offer guidance to avoid some of the challenges other open source projects have faced before.

The weaknesses of OpenStack are mainly due to its complexity. It requires a lot of testing in order to get it to work well with other networking protocols in the telco infrastructure [59].

Users who have adopted OpenStack include Walmart, T-Mobile, AT&T, American Airlines and University of Edinburgh [62]. It also has a strong backing of industry sponsors that support The OpenStack Foundation. This makes OpenStack a big open source project. It being an open source project gives the flexibility to modify the APIs for better integration, subsystem usage and the extention of resource orchestration. However, the challenge presented by the open source nature of OpenStack is that it is a challenge to find OpenStack specialists, therefore, potentially increasing deployment time [63].

An increasing trend among Enterprises is the adoption of the hybrid cloud. This has resulted in many public cloud vendors embracing having their hybrid clouds via OpenStack. As a result, many public cloud vendors are enhancing their compatibility with OpenStack. A potential challenge in using OpenStack for a hybrid cloud solution is that there is no out-of-the-box APIs for integrating with leading public cloud vendors [63].

4.4 Summary

OpenStack is an open source cloud operating system which can be used to deploy public and private cloud environments. It provides Infrastructure as a Service (IaaS) that gives access to compute, networking and storage resources. It allows users to create networks, instances and run applications easily via a GUI dashboard. The different functions provided by OpenStack are realised through the integration of various individual services. The services do not only provide compute, networking and storage resources to OpenStack instances, but they also include management tools that can be used by administrators to monitor and orchestrate instances within the OpenStack environment. This makes OpenStack a robust cloud operating system.

Chapter 5

Phase 1: Building the Infrastructure As A Service

The previous chapters provide the readers with related information that gives a foundation for the work presented in this thesis.This chapter begins by describing the actual experimentation around the building of the telco cloud in which to virtualise network services in order to establish in detail the possibility of doing so exclusively through open source components.

1. Building the OpenStack Cloud,
2. Adding an Orchestrator,
3. Monitoring,
4. Virtualising IMS and testing Autoscaling and Fault Management Functions

This chapter discusses the first phase of the implementation where the starting point is the hardware naturally. This is the standard server (or servers) from which the platform will get its physical resources. Physical resources are the computation power, network resources and storage capacity that the server hardware will provide to the platform [64]. In order to get virtual resources from the hardware, a virtualisation layer is needed, typically in the form of a hypervisor. A cloud operating system is then required to provide IaaS for the platform. Once the basic cloud infrastructure is established, VM instances and networks can be deployed on the platform.

The infrastructure design overview used to build the deployment is given in Figure 5.1. The physical server hosting the platform is connected to the Rhodes University network via an ethernet connection. It runs a hypervisor which provides the virtualisation layer on which the router, IaaS and MANO nodes are hosted. The IaaS nodes are what gives rise to the VIM component of the NFV MANO implementation and interact with the NFV MANO node which implements the NFVO and NFVM components.

In the following sections, more light will be shed on both the hardware and software components chosen for building the cloud environment, why they were chosen and how these components were configured to work together to produce a fully functional cloud platform using open source components.

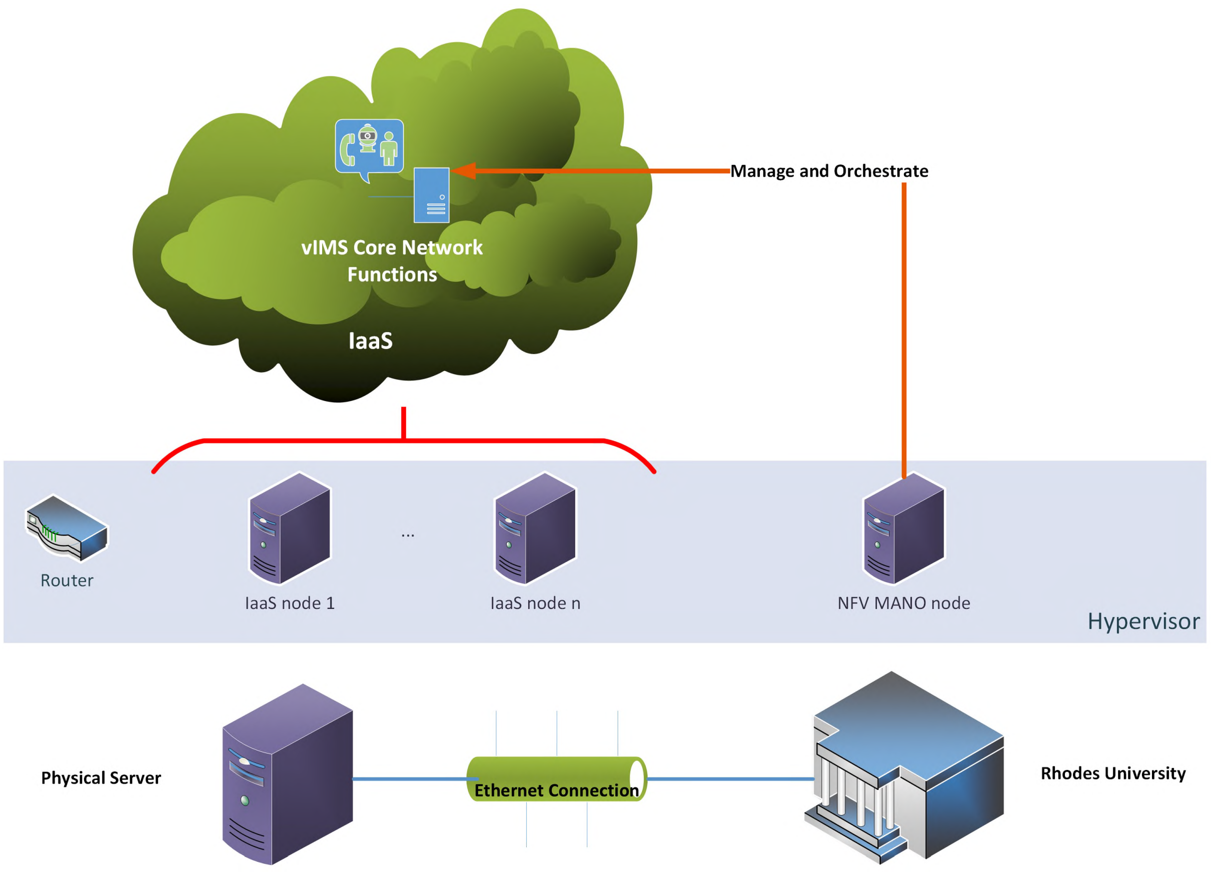


Figure 5.1: Telco Cloud Infrastructure Design Overview

5.1 Server

NFV aims to take network functions from monolithic dedicated hardware appliances so that they are run as software on standard servers. This allows for easier scalability and management of the network [1]. This thesis introduces a scalable and reliable cloud platform for IMS virtualised network functions and to achieve this, a physical server was needed to provide the underlying infrastructure. NFV reduces the operational cost by taking away the need to constantly buy new hardware components when changing the network topology [64]. However, hardware is still an important component that lies at the foundation of building a cloud and virtual environments. The main difference is that instead of having different dedicated hardware components running network functions such as routing and firewall functions, the network will make use of standard servers instead for all its hardware requirements.

The cloud deployment presented in this thesis makes use of a single physical server with the following specifications:

- **Manufacturer** - Supermicro
- **Model** - Super Server X11 SSL
- **CPU** - 4 CPUs x Intel(R) Xeon(R) CPU E3-1220 v5 @ 3.00GHz
- **Memory** - 64GB
- **Storage** - 2TB

5.2 Hypervisor - VMware vSphere (ESXi)

The server provides physical resources needed by the cloud environment. In order to be able to get IaaS resources from this hardware, a hypervisor is needed so that multiple VM nodes can be created with their own operating system and software so that they can host the services that give rise to the cloud environment.

The hypervisor, also known as the Virtual Machine Manager (VMM), is software that abstracts hardware from the operating system and makes it possible to have multiple operating systems on the same hardware. Hypervisors were developed to allocate resources from the physical computing resources provided by the server to the virtual machines on top of it. The abstraction of hardware from the operating system is done using a standard set of instructions that interface with the physical hardware [38]. There are two types of hypervisors namely, type 1 and type 2 hypervisors. Figure 5.2 shows the differences in logical structure between the two types of hypervisors. Type 1 hypervisors are what is

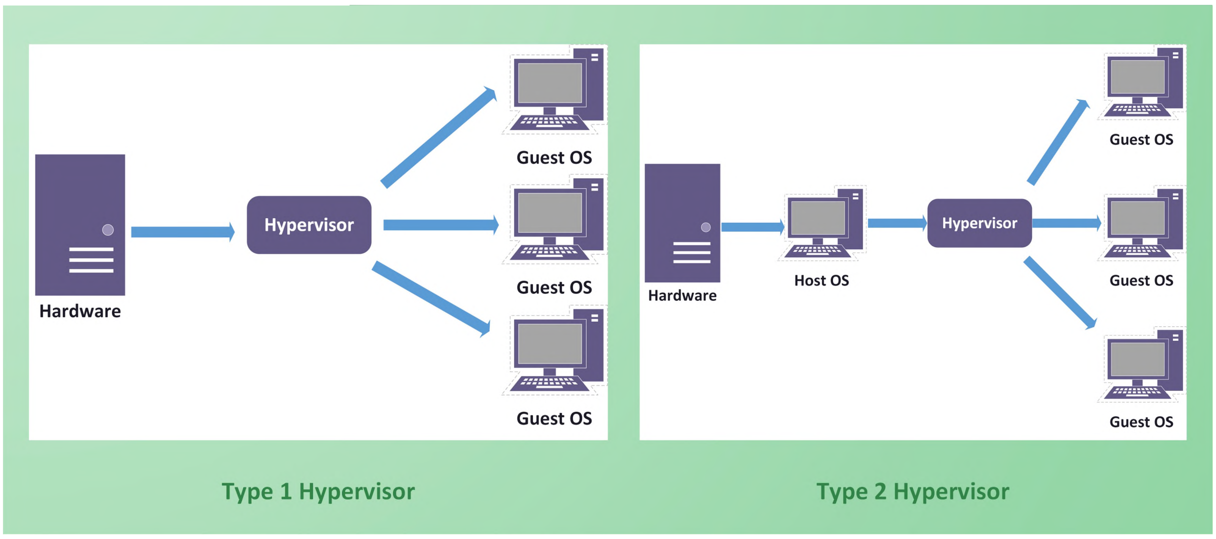


Figure 5.2: Type 1 and Type 2 Hypervisor Architecture.

called 'bare metal' hypervisors. They run directly on top of the hardware and are responsible for scheduling and allocating resources to VMs since there is no operating system. The hypervisor provides device drivers to the VM instances so that they have access to the underlying hardware. Type 2 hypervisors are known as 'hosted' hypervisors and run as an application in a normal operating system known as the host operating system [65]. A bare metal hypervisor offers better scalability, stability and performance since it has direct access to hardware resources [66]. For this reason, the implementation design for this work makes use of a bare metal hypervisor.

VMware offers a bare metal hypervisor solution called VMware ESXi. ESXi is available for free when using the free vSphere Hypervisor edition but it is also available as part of a paid vSphere edition [20, 65]. VMware ESXi installs directly onto the physical server and therefore gives direct access and control of underlying resources. Being a bare metal hypervisor, ESXi offers better efficiency than a hosted hypervisor because it consolidates hardware for improved utilization of resources [20]. For this research, VMware vSphere Hypervisor (ESXi) 6.5 was deployed on the server.

Inside VMware, a virtual switch was configured as shown in Figure 5.3. There is a single network adapter which connects to the university's network. There is a *management network* which maps the hypervisor to the university's network and makes it possible to connect to the hypervisor through a client application from which the hypervisor and any infrastructure inside it can be managed. The *provider network* provides floating IP addresses to VMs. It maps to physical networks that exist in the data center (university in the context of this work). There are several nodes that are connected to these subnets and these are explained in more detail in the sections below.

5.3 OpenStack Operating System - IaaS

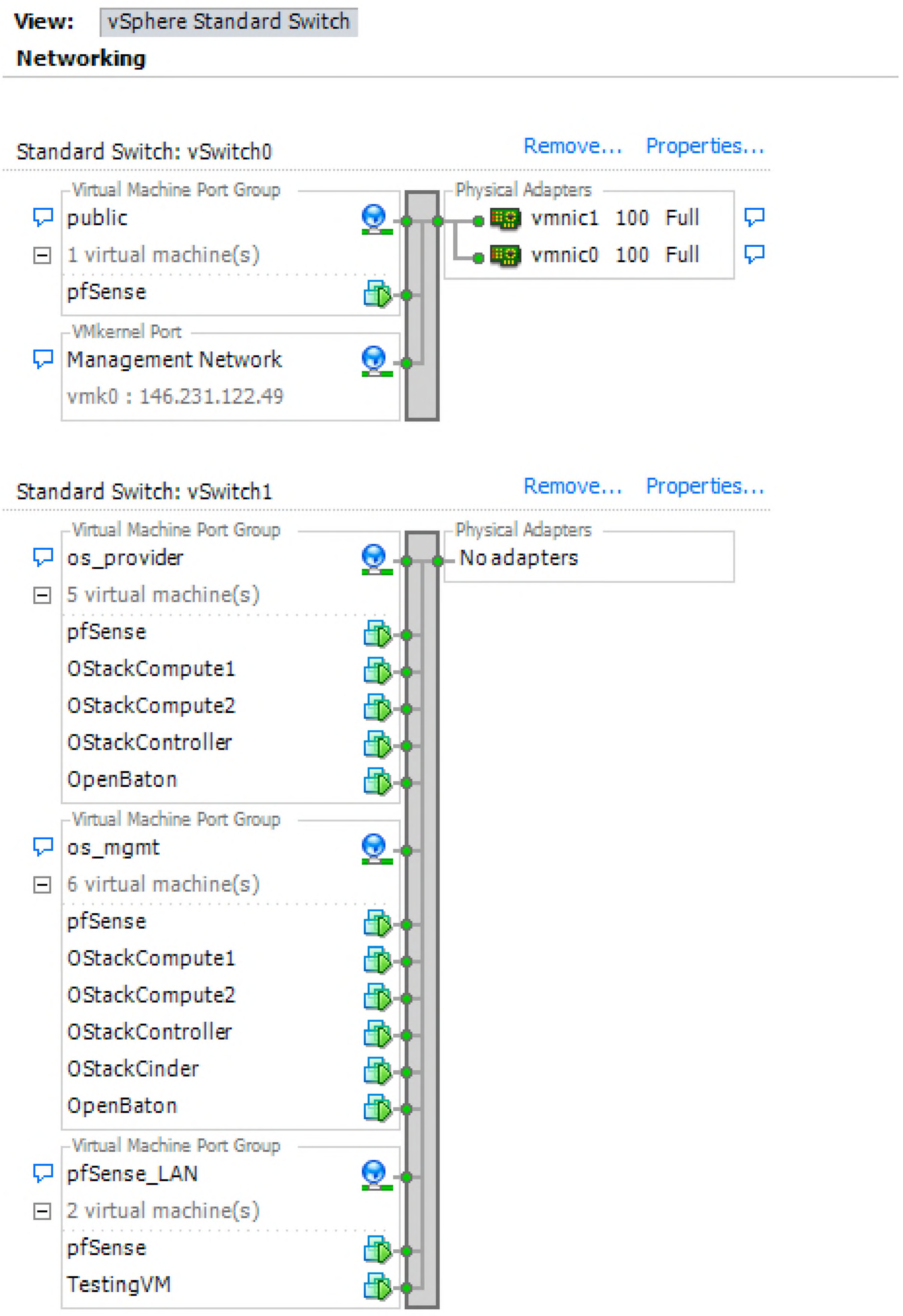
As seen in the previous chapter, OpenStack is a prominent open source cloud operating system that provides IaaS for both the private and public cloud environments. OpenStack offers a number of services that users can take advantage of in implementing an architecture that best suits their needs [9]. OpenStack is a popular choice for cloud deployment and a lot of products being developed for the cloud are designed to integrate well with OpenStack [36].

The design of the telco cloud platform presented in this thesis was guided by the ETSI NFV MANO specification. The specification requires the presence of a Virtual Infrastructure Manager (VIM). OpenStack fulfills the role of a VIM and makes it possible for developers to be able to manage virtual resources and allocate them to the different VMs, networks and services. So, OpenStack was the VIM of choice due to it being a popular open source cloud operating system that complies with the MANO specification.

A minimal OpenStack deployment requires at least two nodes (hosts) [9]. One node has the role of a *Controller* and the other node is the *Compute* node. The OpenStack architecture chosen for this research makes use of more than just the minimal requirements and makes use of four OpenStack nodes. These nodes make up the building blocks of the OpenStack deployment and run the services that make the cloud platform functional. The nodes were launched as VMs inside VMware and can be seen in Figure 5.3. The different roles of the nodes deployed are explained below, as well as the general requirements for deploying a simple OpenStack implementation.

5.3.1 Controller

The controller node runs a number of major services namely the Identity service, manages the Compute service, Image service, the Dashboard and it manages the Networking



View: vSphere Standard Switch

Networking

Standard Switch: vSwitch0 Remove... Properties...

Virtual Machine Port Group
public
1 virtual machine(s)
pfSense

VMkernel Port
Management Network
vmk0 : 146.231.122.49

Physical Adapters
vmnic1 100 Full
vmnic0 100 Full

Standard Switch: vSwitch1 Remove... Properties...

Virtual Machine Port Group
os_provider
5 virtual machine(s)
pfSense
OStackCompute1
OStackCompute2
OStackController
OpenBaton

Virtual Machine Port Group
os_mgmt
6 virtual machine(s)
pfSense
OStackCompute1
OStackCompute2
OStackController
OStackCinder
OpenBaton

Virtual Machine Port Group
pfSense_LAN
2 virtual machine(s)
pfSense
TestingVM

Physical Adapters
No adapters

Figure 5.3: Networking within VMware

and various Networking agents. If the OpenStack architecture deployed includes optional services such as Block storage, Orchestration, Telemetry services and Object storage, portions of these services are also run by the controller node. Additionally, the controller runs supporting services such as the message queue, an SQL database and NTP (Network Time Protocol).

Specification requirements: 1-2 CPUs, 8GB RAM, 100GB Storage and 2 network interfaces.

Specifications for deployment: A single Controller node was deployed as a VM named OStackController in VMware vSphere with 2 vCPUs, 12GB Memory, 100GB Storage and 2 network interfaces.

5.3.2 Compute

Compute nodes runs the hypervisor (KVM hypervisor is the default) that operates instances. There can be more than one Compute node in an OpenStack deployment. It runs a network service agent so that virtual instances are able to connect to virtual networks and have access to firewall services through security groups.

Specification requirements:2-4+ CPUs, 8+GB RAM, 100+GB Storage and 2 network interfaces.

Specifications for deployment: Two Compute nodes namely, OStackCompute1 and OStackCompute2, were deployed as VMs in VMware vSphere with each node having 4 vCPUs, 12GB Memory, 100GB Storage and 2 network interfaces.

5.3.3 Block

Block node is an optional node and contains disks provisioned for instances by the Block Storage and Shared File System services. An OpenStack deployment can have more than one Block node.

Specification requirements: 1-2 CPUs, 4BG RAM, 100+GB Storage and 1 network interface.

Specifications for deployment: A Block node named OStackCinder, was deployed in VMware vSphere with 2 vCPUs, 8GB Memory, 3 Hard disks (2 with 300GB and 1 with 50GB) and 1 network interface.

OpenStack Ocata was installed following the step-by-step installation and configuration instructions as provided in the documentation. Ocata is not the latest OpenStack version of OpenStack currently available but it was the latest fully documented OpenStack implementation when work on the experimental setup commenced. The CPU, memory, storage and network interface allocations were decided on based on the hardware requirements in the documentation, as well as monitoring how CPU and memory resources responded to instances running on the platform.

5.3.4 Overview of OpenStack Cloud

OpenStack offers a number services that users can take advantage of in their implementations. The deployment in this thesis only makes use of some of the available services. Once the major services are up and running, it is possible to install other additional services as required. Figure 5.4 shows the basic OpenStack architecture deployed for this research which is made up of mainly the major services.

The different nodes running in VMware vSphere are what give us a working OpenStack cloud platform that is accessible via a dashboard that is accessible through a web browser. A view of the OpenStack dashboard can be seen in Figure 5.5 which shows the number of instances, vCPU, RAM, Floating IP, Security Groups, Volumes and Volume Storage resources available. These are just the currently configured limits but administrators can adjust them as required.

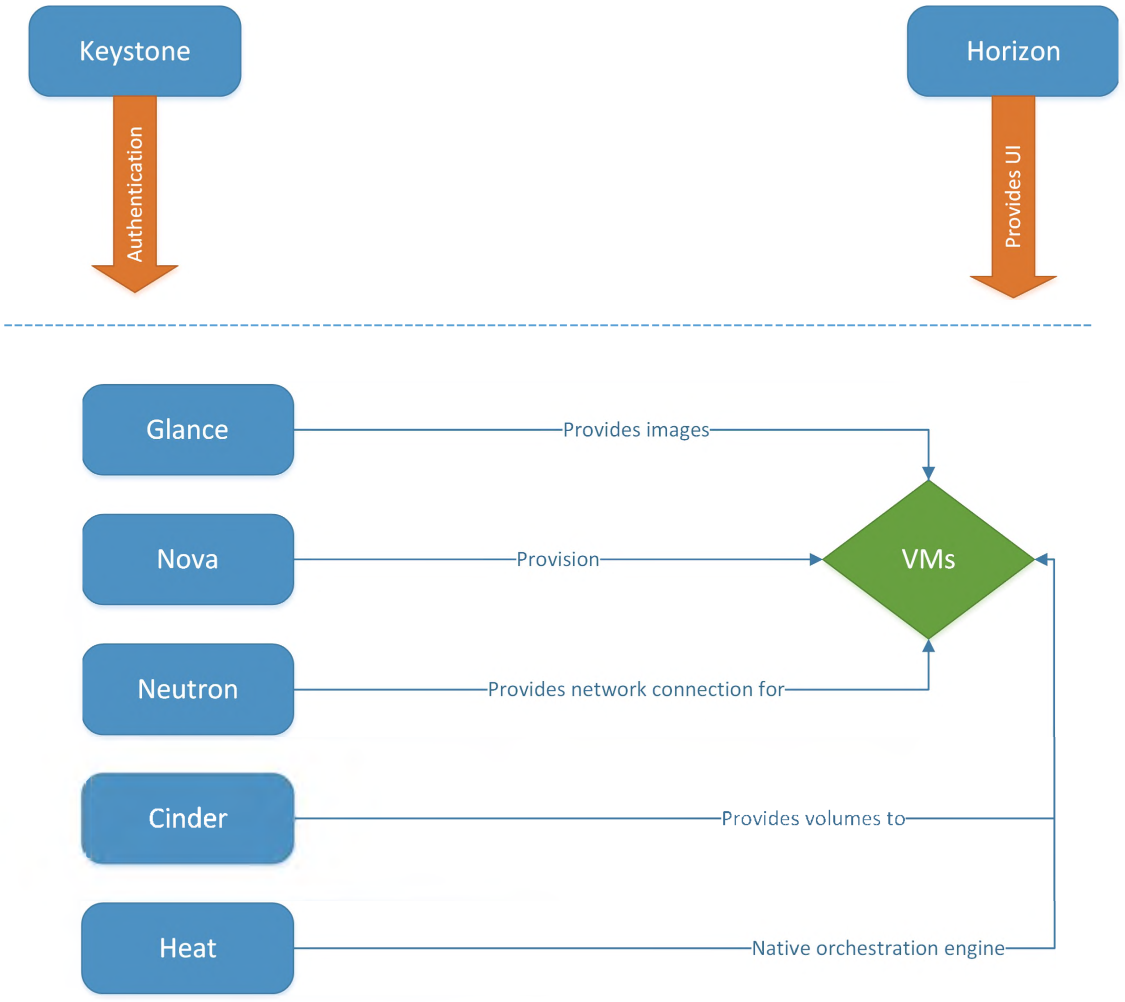
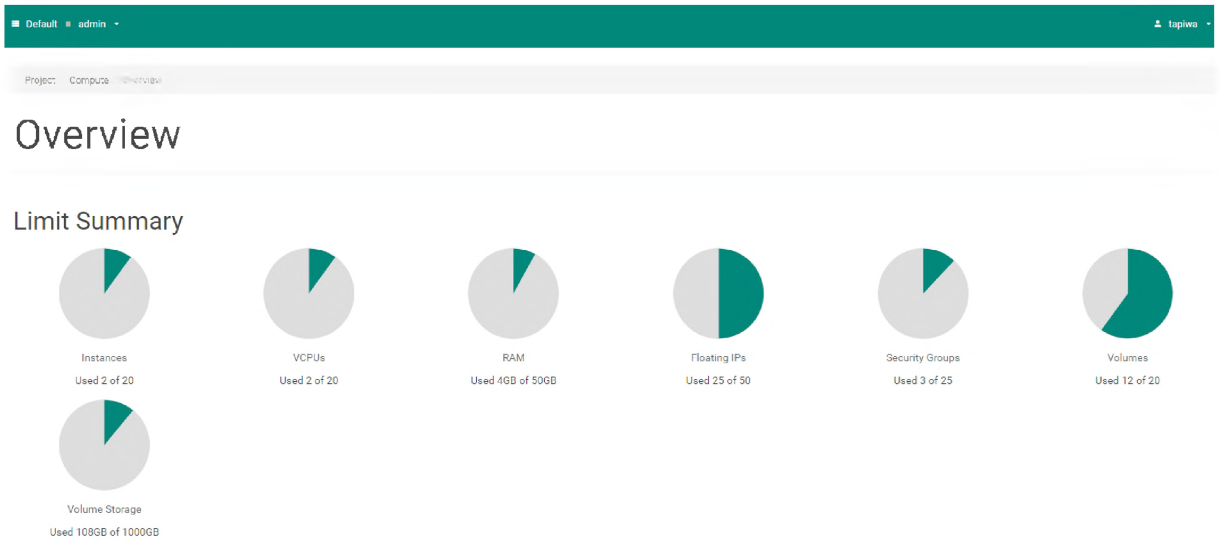


Figure 5.4: Architecture of the deployed OpenStack Services



Overview

Limit Summary

Instances
Used 2 of 20

VCPUs
Used 2 of 20

RAM
Used 4GB of 50GB

Floating IPs
Used 25 of 50

Security Groups
Used 3 of 25

Volumes
Used 12 of 20

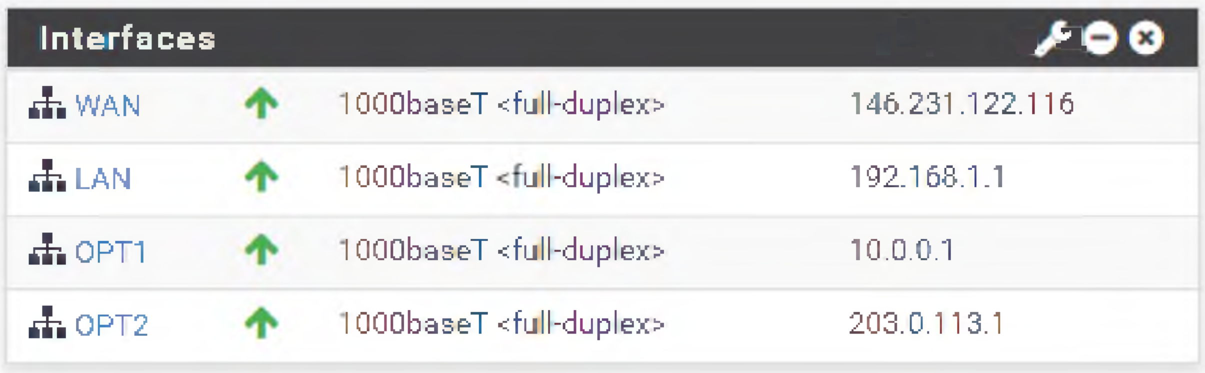
Volume Storage
Used 108GB of 1000GB

Figure 5.5: OpenStack Dashboard

5.4 Router

A final key element to get a functional cloud platform is a router. A router is needed to allow communication within the internal OpenStack network but also between OpenStack instances and the Internet.

The choice of router for this implementation is the pfSense router. pfSense software is free and open source that can be tailored to be used as a firewall and router that is managed via a web interface. It is a powerful and flexible firewall solution based on FreeBSD and has multiple related features which makes it a perfect routing solution for a dynamic platform [67]. There are 4 network interfaces configured on the router as show in Figure 5.6. The WAN interface is the public IP address that the physical server is connected to. The LAN network is the one the private IP addresses inside OpenStack belong to. The OPT1 network is the network to which the nodes that host the OpenStack cloud belong. The OPT2 network is the one that provides floating IPs to the instances hosted in the OpenStack cloud. Figures 5.7 to 5.10 show the firewall rules defined to allow communication between network entities and access to resources needed to manage the platform.

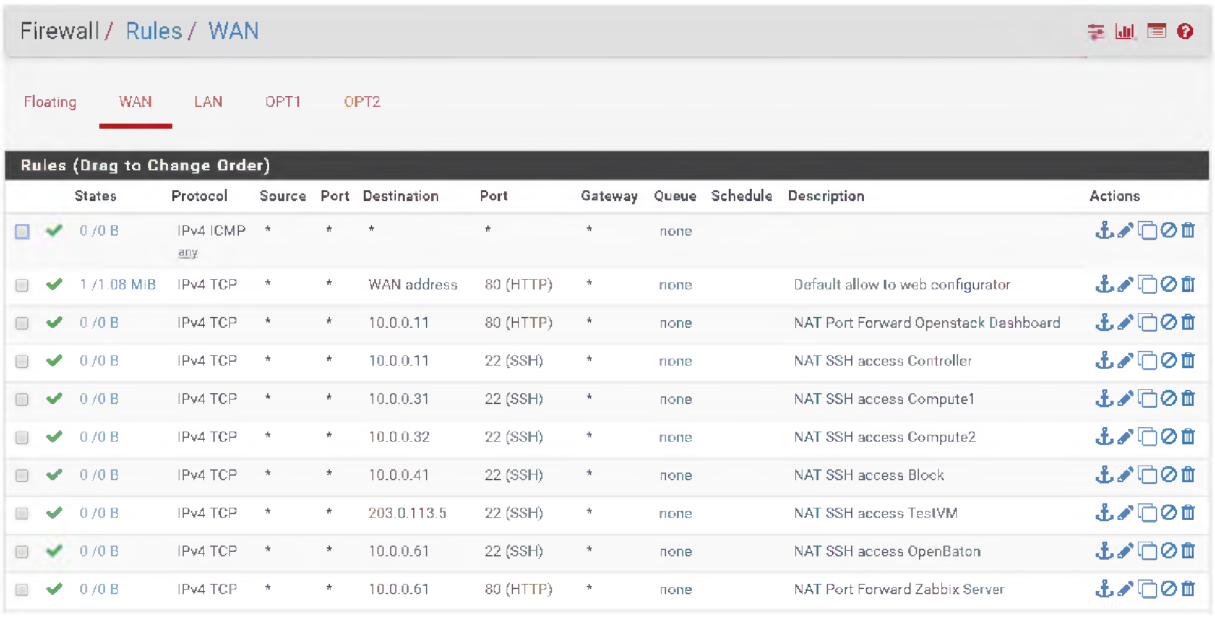


WAN	1000baseT <full-duplex>	146.231.122.116
LAN	1000baseT <full-duplex>	192.168.1.1
OPT1	1000baseT <full-duplex>	10.0.0.1
OPT2	1000baseT <full-duplex>	203.0.113.1

Figure 5.6: The Four Interfaces Configured.

5.5 Summary

Implementing a cloud environment for virtualised IMS requires building the platform from bottom up. This means starting from the choice of server that provides physical resources for the cloud platform and ending with a discussion about virtual machines on which the VNFs will run. A server with 4 CPUs, 64GB memory and 2TB of storage was

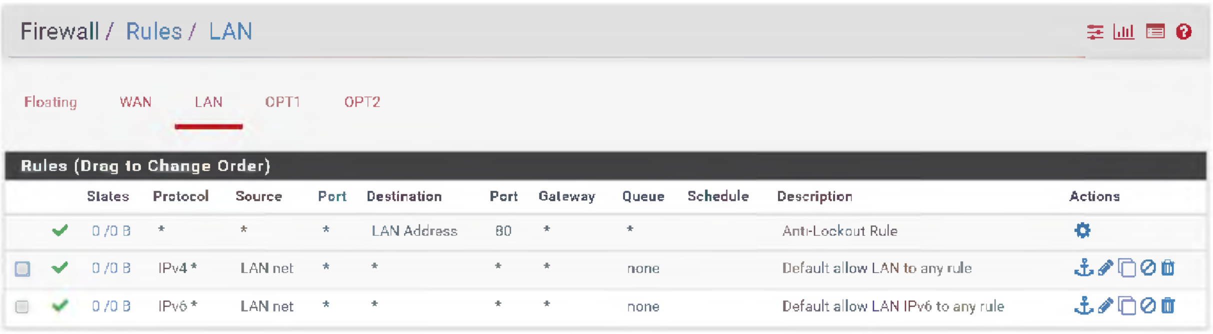

Firewall / Rules / WAN

Floating WAN LAN OPT1 OPT2

Rules (Drag to Change Order)

		States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
	✓	0 /0 B	IPv4 ICMP any	*	*	*	*	*	none			
	✓	1 /1.08 MiB	IPv4 TCP	*	*	WAN address	80 (HTTP)	*	none		Default allow to web configurator	
	✓	0 /0 B	IPv4 TCP	*	*	10.0.0.11	80 (HTTP)	*	none		NAT Port Forward Openstack Dashboard	
	✓	0 /0 B	IPv4 TCP	*	*	10.0.0.11	22 (SSH)	*	none		NAT SSH access Controller	
	✓	0 /0 B	IPv4 TCP	*	*	10.0.0.31	22 (SSH)	*	none		NAT SSH access Compute1	
	✓	0 /0 B	IPv4 TCP	*	*	10.0.0.32	22 (SSH)	*	none		NAT SSH access Compute2	
	✓	0 /0 B	IPv4 TCP	*	*	10.0.0.41	22 (SSH)	*	none		NAT SSH access Block	
	✓	0 /0 B	IPv4 TCP	*	*	203.0.113.5	22 (SSH)	*	none		NAT SSH access TestVM	
	✓	0 /0 B	IPv4 TCP	*	*	10.0.0.61	22 (SSH)	*	none		NAT SSH access OpenBaton	
	✓	0 /0 B	IPv4 TCP	*	*	10.0.0.61	80 (HTTP)	*	none		NAT Port Forward Zabbix Server	

Figure 5.7: WAN Rules.

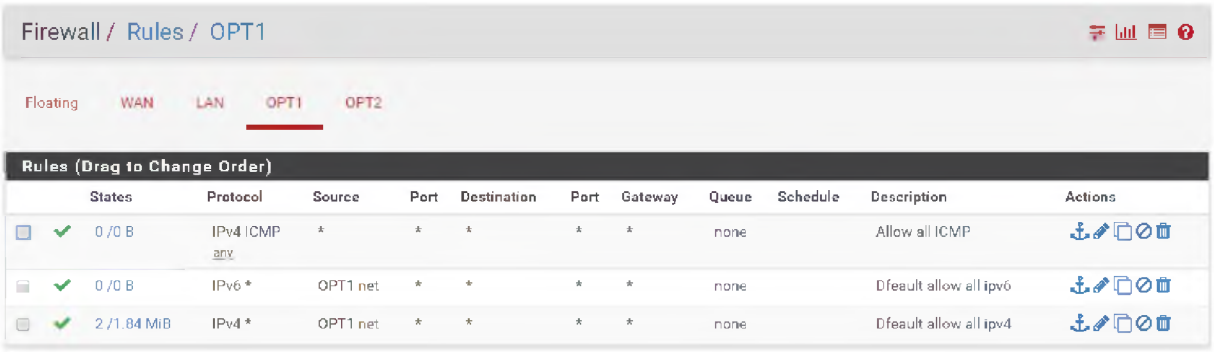

Firewall / Rules / LAN

Floating WAN LAN OPT1 OPT2

Rules (Drag to Change Order)

		States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
	✓	0 /0 B	*	*	*	LAN Address	80	*	*		Anti-Lockout Rule	
	✓	0 /0 B	IPv4 *	LAN net	*	*	*	*	none		Default allow LAN to any rule	
	✓	0 /0 B	IPv6 *	LAN net	*	*	*	*	none		Default allow LAN IPv6 to any rule	

Figure 5.8: LAN Rules.

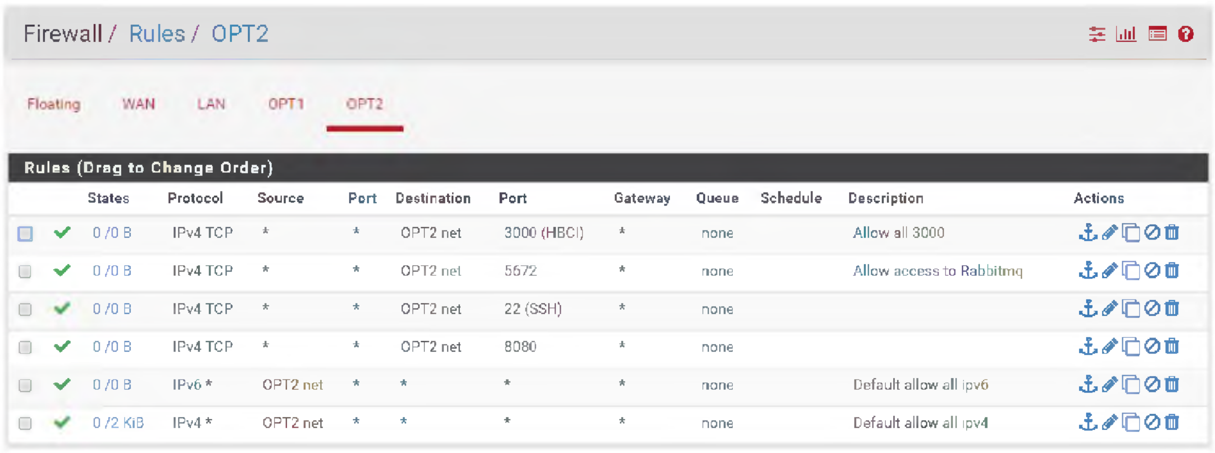

Firewall / Rules / OPT1

Floating WAN LAN OPT1 OPT2

Rules (Drag to Change Order)

		States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
	✓	0 /0 B	IPv4 ICMP any	*	*	*	*	*	none		Allow all ICMP	
	✓	0 /0 B	IPv6 *	OPT1 net	*	*	*	*	none		Dfeault allow all ipv6	
	✓	2 /1.84 MiB	IPv4 *	OPT1 net	*	*	*	*	none		Dfeault allow all ipv4	

Figure 5.9: OPT1 Rules.


Firewall / Rules / OPT2

Rules (Drag to Change Order)

		States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
	✓	0 /0 B	IPv4 TCP	*	*	OPT2 net	3000 (HBCI)	*	none		Allow all 3000	
	✓	0 /0 B	IPv4 TCP	*	*	OPT2 net	5672	*	none		Allow access to Rabbitmq	
	✓	0 /0 B	IPv4 TCP	*	*	OPT2 net	22 (SSH)	*	none			
	✓	0 /0 B	IPv4 TCP	*	*	OPT2 net	8080	*	none			
	✓	0 /0 B	IPv6 *	OPT2 net	*	*	*	*	none		Default allow all ipv6	
	✓	0 /2 KiB	IPv4 *	OPT2 net	*	*	*	*	none		Default allow all ipv4	

Figure 5.10: OPT2 Rules.

chosen. To get the virtual resources for the cloud, a virtualisation layer is required in the form of a hypervisor. The VMware vSphere ESXi Hypervisor 6.5 was installed on to the server. This is an open source Type 1 hypervisor that installs directly onto the server and gives VMs a more direct access to physical resources from the server. Inside VMware, several VMs were deployed to host the services needed to create an OpenStack cloud environment as well as VMs used for testing and orchestrasting the network services. Getting the VMware hypervisor installed and ready to host VMs was simple and was done without difficulty.

The work in this thesis is in the spirit of the ETSI NFV MANO specification. This means that the ETSI NFV MANO specification guided the design process. The specification highlights three major components that are required in the management and orchestration of VNFs namely VIM, VNFM and NFVO. OpenStack, the cloud operating system provides the IaaS and represents the VIM component of the specification. OpenStack Ocata was installed with the following nodes: a controller node, two compute nodes and one block node. All these components were easily configured owing to OpenStack is a well documented tool. A pfSense router was installed on its own VM inside VMware and on it four network interfaces (WAN, LAN, OPT1 and OPT2) were configured using the easy to use web interface. Some firewall rules were also defined for each of these networks.

Chapter 6

Phase 2: Management and Orchestration

Chapter 5 presented the implementation of the OpenStack cloud platform. The second part of the implementation is the VNF management and orchestration. The orchestrator is an important part of the implementation as it responsible for the creation of the network service and managing the network services.

This chapter will focus on the deployment of the Open Baton orchestrator. It will provide details on how the orchestrator is configured and the different components offered by Open Baton to give a robust management and orchestration platform.

6.1 NFV Orchestrator

According to the ETSI NFV MANO specification, the orchestrator is the main component that makes decisions for how services and infrastructure interact. The importance of the role of an orchestrator as the main decision point can be seen in Figure 6.1. It is the only component in the architecture with a complete view of the infrastructure resources and allocates them to the different services based on their requirements. The orchestrator also maintains the resources of the network services during their life-cycle management [11].

Open Baton was installed and configured on the OpenBaton node inside VMware. The Open Baton VM was created with 2 CPUs, 8GB memory and 100GB of storage. The

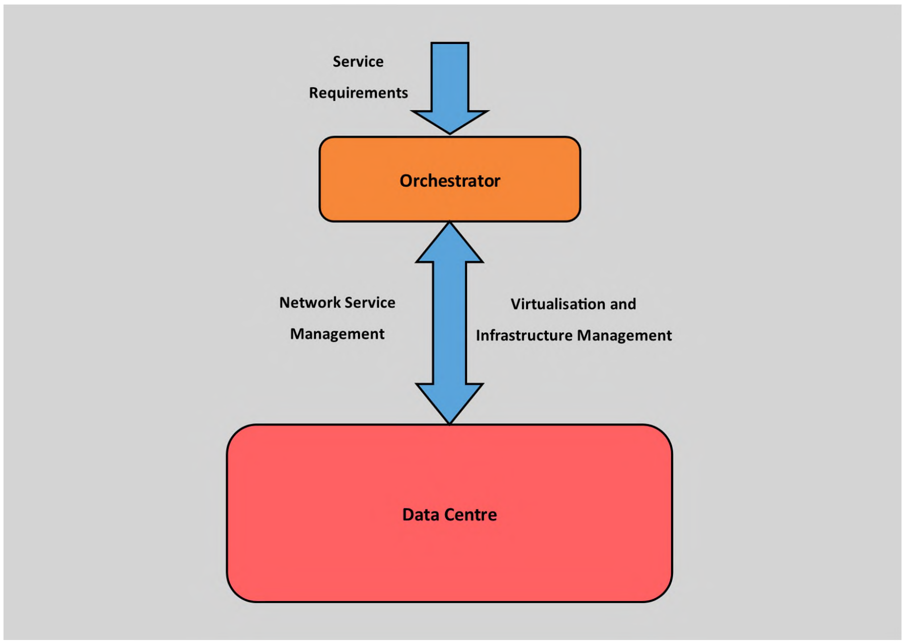

Service Requirements
Orchestrator
Network Service Management
Virtualisation and Infrastructure Management
Data Centre

Figure 6.1: The Orchestrator as the Main Decision Point. Adapted from [11]

NFVO is the main function of Open Baton. It has a modular architecture and achieves its different functions through a set of modules [11]. Figure 6.2 shows the Open Baton dashboard through which several actions can be performed. Most of these can also be done via the CLI, however the dashboard provides a nice visual of the different elements in the Open Baton environment and their statuses.

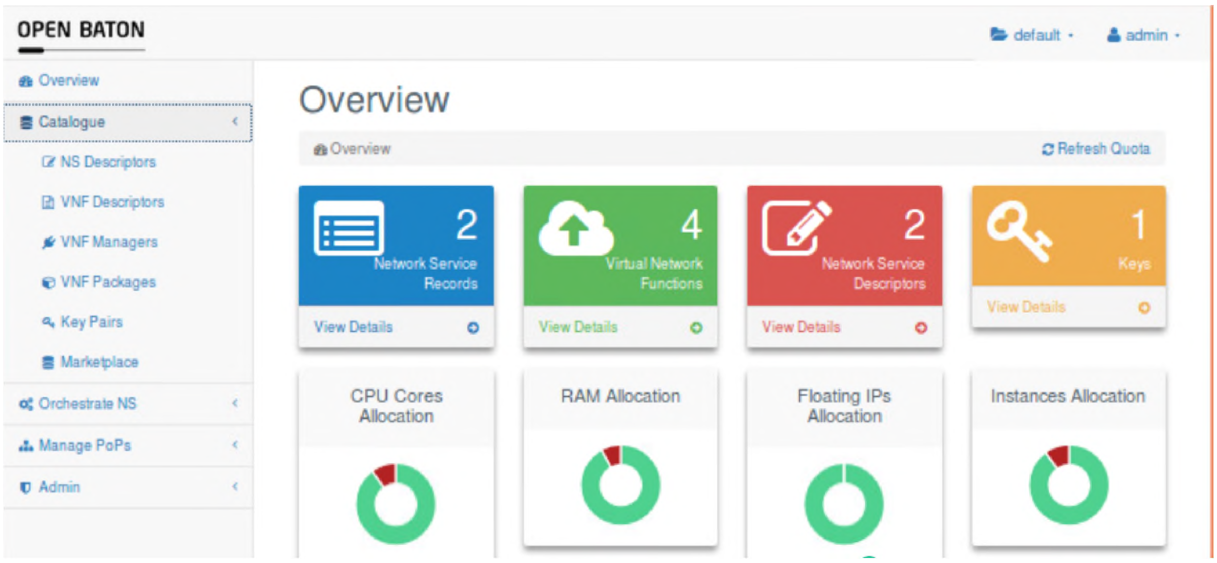


Figure 6.2: Open Baton Dashboard.

6.2 VNF Manager

The VNFM is an independent component that is able to communicate with the NFVO and other components via a message bus or RPC protocols [36]. There are three ways in which a VNFM can be integrated to work with the Open Baton NFVO.

1. Using the Generic VNFM or the Juju VNFM which are part of the Open Baton project. This is the simplest and quickest approach that makes it easy to get started with the VNF package implementation.
2. Using another VNFM that connects to the NFVO using the REST API or AMQP.
3. Building a VNFM using the SDK in Java or Python called vnfm-sdk. Communication between the NFVO and VNFM (Or-Vnfm) can be done using vnfm-sdk-amqp or vnfm-sdk-rest depending on whether the user prefers AMQP or REST.

For ease of deployment, the Generic VNFM was chosen for this work. It is a VNFM implementation that follows the ETSI MANO specification. It is the component that sits between the NFVO and the VMs on which the VNF software is installed. The VNFM also communicates with the EMS which is an agent inside the VMs that is responsible for running VNF scripts. Figure 6.3 shows the communication between the NFVO, Generic VNFM and EMS. This communication is done using the AMQP protocol over RabbitMQ. The VNFM is responsible for sending commands to the EMS running on the VM. The EMS then runs these commands on the VNF component (VNFC) [11].

6.3 VIM Integration - Point of Presence

The VIM was described in more detail in Chapter 2. It is the component in the NFV MANO framework whose main task is to provide access to the NFVI resources and manages these resources. Open Baton mainly provides the Open Baton NFVO and has implementations of the Generic VNFM or Juju VNFM. However, it needs access to virtualised resources from the virtualisation host [36]. This is the role of a VIM, which is an external component that has to be added to the system.

Open Baton implements VIM drivers for interacting with VIMs. The plugin interfaces are Remote Procedure Call (RPC) based. The NFVO is installed with four default drivers

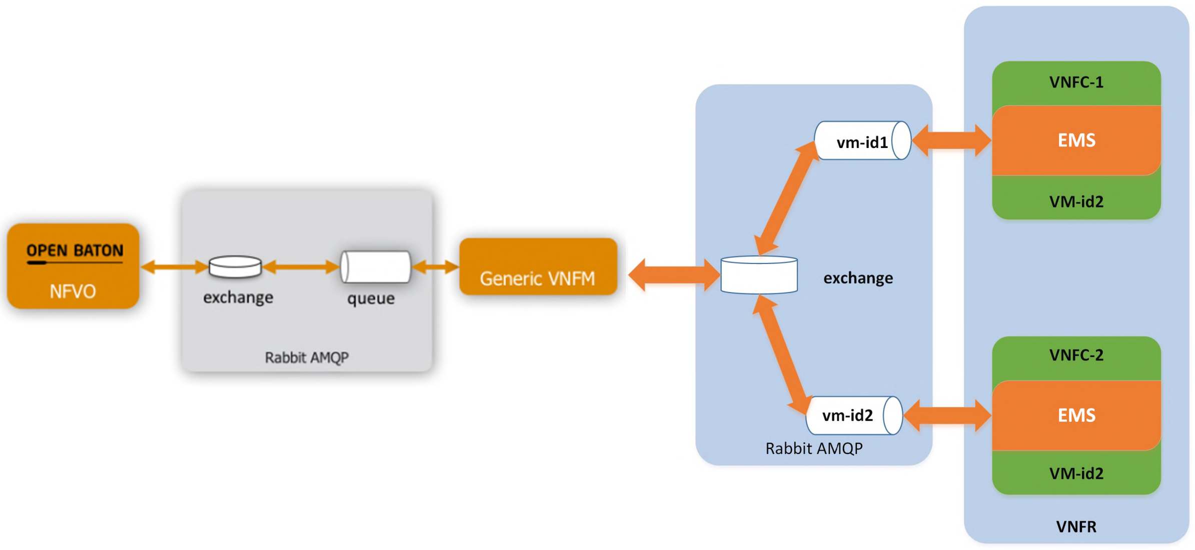


Figure 6.3: Communication between the NFVO, Generic VNFM and EMS. Source [11]

namely, *openstack*, *amazon*, *docker* and *test* (for testing a VIM mockup) [11]. Most deployments of Open Baton typically use OpenStack as their VIM of choice. The OpenStack driver uses the OpenStack4J library which provides Java APIs for calling OpenStack REST APIs and enables the NFVO and VNFM to interoperate with a Point of Presence (PoP) that is managed by OpenStack. Listing 6.1 shows the JSON file used to register the OpenStack VIM on the NFVO via the dashboard under Manage PoPs.

Listing 6.1: JSON for registering OpenStack PoP

```
{
    "name":"vim-instance",
    "authUrl":"http://controller:5000/v3/",
    "tenant":"cc06e792b8494fc78db6109f77356071",
    "username":"admin",
    "password":"openbaton",
    "keyPair":"openbaton-keypair",
    "securityGroups":[
        "default"
    ],
    "type":"openstack",
    }
}
```

Table 6.1 explains the parameters used in the JSON file that is used to register the PoP. The registration of a PoP completes the requirements of an ETSI MANO compliant system and network services can be deployed on the OpenStack cloud using the Open Baton NFVO and Generic VNFM.

Parameters	Meaning
name	The name of the Point of Presence.
authUrl	Keystone APIs endpoint.
tenant	The version of the NSD.
username	OpenStack username for an account with admin rights.
password	The password for the OpenStack account.
keyPair	An optional keypair used to access all the VMs.
securityGroups	The security group with filtering rules to be applied to instances.
type	The type of PoP used.

Table 6.1: PoP Registration Parameters. Source [11].

6.4 Network Service Deployment Files

Creating VNFs on the OpenStack VIM using the Open Baton orchestrator is done through several scripts and files. Files for some common open source network services are available on the Open Baton Marketplace. The two resources needed to create a network service are VNF packages and a Network Service Descriptor (NSD). For simple network services, the NSD alone can be used to deploy the network.

6.4.1 VNF Packages

A VNF package is either a TAR or CSAR archive with configuration information needed to create the network service. It defines the parameters that match the resources available in the PoP as well as the network requirements of the given network service. A VNF package can also include policies on how the VNF should be managed during runtime. The TAR format, which is the one used for this thesis, follows the ETSI NFV specification for VNF packages and descriptors. VNF packages typically include:

- **VNF descriptor (VNFD)**: contains the information needed by the NFVO to deploy the VNF.
- **Metadata file**: contains information for the NFVO to understand the contents of the package.
- **Scripts Folder**: contains scripts used for lifecycle management.

VNFD

The VNFD is a specification template from the VNF creator that details the virtual resource requirements of a given VNF. The template is used by the NFV MANO functions for VNF lifecycle operations including instantiation [68]. In Open Baton it is a JSON file which can be added directly into the NSD file especially for simpler networks services. For more complex network services, the the VNFD is referenced by its ID in the NSD. As suggested by the names, the VNFD contains information about a specific VNF in a network service and the NSD contains information about the entire network service. An example of a VNFD can be seen in listing 6.2 which shows the VNFD of the OpenIMSCore p-cscf component (available from the Open Baton Marketplace).

Listing 6.2: JSON for OpenIMSCore P-CSCF VNFD

```
{
    "name":"pcscf",
    "vendor":"fokus",
    "version":"5.1.0",
    "lifecycle_event":[
        {
            "event":"CONFIGURE",
            "lifecycle_events":[
                        "bind9_relation_joined.sh"
            ]
        },
        {
            "event":"INSTANTIATE",
            "lifecycle_events":[
                        "pcscf_install.sh"
            ]
        },
        {
            "event":"START",
            "lifecycle_events":[
                        "pcscf_generate_config.sh",
                        "pcscf_start.sh"
            ]
        }
    ],
    "configurations":{
        "name":"client-configuration",
        "configurationParameters":[
            {
                "confKey":"port",
                "value":"4060"
            },
            {
                "confKey":"name",
                "value":"pcscf"
            }
        ]
    },
    "vdu":[
        {
            "vm_image":[],
            "scale_in_out":1,
            "vnfc":[
                {
                    "connection_point":[
                        {
                            "floatingIp":"random",
                            "virtual_link_reference":"mgmt"
                        }
                    ]
                }
            ]
        }
    ],
    "virtual_link":[
        {
            "name":"mgmt"
        }
    ],
    "deployment_flavour":[
        {
            "flavour_key":"m1.small"
        }
    ],
    "type":"pcscf",
    "endpoint":"generic"
}
```

Table 6.2 shows the main parameters that need to be defined in the VNFD.

Parameters	Meaning
name	The name of the VNF.
vendor	The provider of the VNF.
version	The version of the VNF.
lifecycle_events	A life_cycle event is made up of an Event and a list of script names that need to run in that specific Event. The supported event names are INSTANTIATE, CONFIGURE, START, TERMINATE and SCALE_IN.
configurations	A configuration object contains a list of key-value parameters that can be used in the scripts.
vdu	This is the virtual deployment unit. It has its own set of parameters which are listed in Table 6.3.
virtual_link	It is also known as the internal virtual link which points to a virtual link descriptor in the NSD.
deployment_flavour	The deployment flavour must correspond to a flavour on the VIM, OpenStack in this case.
type	A string that states the type of the VNF. It is referenced in the dependency parameter in the script files.
endpoint	Defines the VNFM responsible for the VNF.

Table 6.2: VNF Descriptor Parameters. Source [11].

Parameters	Meaning
vm_image	A list of image names or IDs available in the PoP.
vimInstanceName	A list of PoP where the VNF Components of the VDU will be deployed. In the case where there is more than one, one is selected at random.
scale_in_out	The maximum number of nodes that can be launched within the VDU at runtime.
vnfc	The list of VNF Components to be created when the VDU is instantiated.

Table 6.3: Virtual Deployment Unit (VDU) Parameters. Source [11].

Metadata YAML file

The Metadata file that contains VNF properties stored in simple <key>:<value> pairs. Listing 6.3 is the corresponding Metadata file for the OpenIMSCore p-cscf component whose VNFD is shown in listing 6.2.

Listing 6.3: Metadata.yaml file for OpenIMSCore pcscf

```
name: pcscf
provider: fokus
nfvo_version: 5.1.0
description: "A Proxy-CSCF (P-CSCF) is a SIP proxy that is the
first point of contact for the IMS terminal. It can be located
either in the visited network (in full IMS networks) or in the
home network (when the visited network is not IMS compliant yet)."
image:
    upload: "false"
    names:
        - openims
    link: http://marketplace.openbaton.org:8082/api/v1/images/
        52e2ccc0-1dce-4663-894d-28aab49323aa/img
image-config:
    name: openims
    diskFormat: qcow2
    containerFormat: bare
    minCPU: 0
    minDisk: 0
    minRam: 0
    isPublic: true
vim_types:
    - openstack
```

The following parameters are to be defined in the Metadata file:

- **name**: The name of the VNF package.
- **provider**: The provider of the VNF.
- **nfvo_version**: The version of the NFVO which supports this package. The version is given in the format X.Y.Z and the X.Y value is what is used to check whether the VNF package is supported by the NFVO during on-boarding. If the version is not supported, an error will appear when trying to on-board the package.
- **description**: A description of the VNF in human readable form.
- **image**:
    - **upload**: The options are: true, false or check. Where *true* means that the defined image must be uploaded on all VIM instances, regardless of whether an image with that name already exists. *False* means that the image is assumed

to already exist on the VIM. *Check* is useful if there might be need to upload a new image from a *link*.

  - **ids**: The list of image IDs used to fetch the correct image from the VIM instance. If no image matching the ID is found, it will check using image names. If multiple images match, an exception will be thrown because of ambiguity.
  - **names**: The list of image names used to fetch the correct image from the VIM instance. The image selection is done in the same way as with the *ids*. In the example in listing 6.3, no are defined so the *names* option was checked instead.
  - **link**: the link points to a URL where an image file is available for upload on the VIM instance.

- **image-config**:
  - **name**: The name of the image to be uploaded. Available either directly in the VNF package or via a URL link.
  - **diskFormat**: The format of the disk type the image is stored in.
  - **containerFormat**:The format of the container type the image is stored in.
  - **minCPU**: Minimum number of CPU cores required for using this image.
  - **minDisk**: Minimum amaount of disk space required for using this image.
  - **minRam**: Minimum amount of RAM required for using this image.
  - **isPublic**: Defines whether the image will be available to all tenants on the VIM instance.
- **vim_types**: The list of VIM types supported by the VNF package.

Scripts

The scripts folder contains scripts needed to instantiate, configure and start a VNF on a VM or container. These scripts are referenced in the lifecycle_events defined in the VNFD and are executed based on the order of definition. A *scripts-link* can be used in the Metadata file to reference a URL link of where the scripts are located. If this parameter is defined, the scripts folder is ignored because the scripts-link has a higher priority over the folder.

At the time of this research, the scripts folder cannot have subfolders and all scripts are at the same level within the scripts folder. The script files are either Python scripts or shell scripts. Depending on the VNF Manager used, there are multiple ways in which runtime parameters can be passed to the scripts.

6.4.2 Network Service Descriptor

The Network Service Descriptor is a template file that follows the ETSI MANO Specification that can be used to launch network services. There are two formats that can be used for an NSD namely, JSON and TOSCA. The JSON format is the representation specified by ETSI and therefore the one implemented in this thesis.

Listing 6.4 shows the condensed NSD file for the OpenIMSCore network service. The NSD is the file that is used to launch the network service and create a Network Service Record (NSR) on the NFVO. For this reason, the NSD needs to reference all the VNFDs for all the VNFs in the network service and this is done by adding a list of VNFD IDs in the *vnfd* parameter. A network service is the result of different network functions working together effectively and this implies some dependency between the VNFs in the network. These dependencies can be defined in the NSD and make a big part of the NSD especially for complex networks.

Listing 6.4: Sample Network Service Descriptor (NSD) JSON file

```
{
    "name":"OpenIMSCore Bind9 FHoSS",
    "vendor":"fokus",
    "version":"2.0",
    "vnfd":[...],
    "vld":[
        {
            "name":"mgmt"
        }
    ],
    "vnf_dependency":[
        {
            "source":{
                "name":"bind9"
            },
            "target":{
                "name":"fhoss"
            },
            "parameters":[
                "useFloatingIpsForEntries",
                "realm",
                "mgmt",
                "mgmt_floatingIp"
            ]
        },
        ...
        {
            "source":{
                "name":"icscf"
            },
            "target":{
                "name":"scscf"
            },
            "parameters":[
                "name",
                "mgmt",
                "mgmt_floatingIp"
            ]
        }
    ]
}
```

The NSD has several parameters that provide information to the NFVO about the service being deployed. The parameters are explained in Table 6.4.

Parameters	Meaning
name	The name of the network service.
vendor	The provider of the network service.
version	The version of the NSD.
vnfd	The list of VNF Descriptors for all the VNFs that need to be deployed. This can be the JSON representation of the VNFD or simply the ID of the VNFD. However, if VNF packages are on-boarded for the NS, there is no need to add the JSON again in the NSD and its more convenient to use the IDs generated by the NFVO.
vld	The list of virtual links referenced in the vld sections of the VNF Descriptors which are used to define network connectivity.
vnf_dependency	An optional list of dependencies between VNFs.

Table 6.4: Network Service Descriptor Parameters. Source [11].

6.5 Creation of the Network Service Record (NSR)

Once the NSD is launched, the process shown in Figure 6.4 begins. The sequences of events shows how messages are sent between the NFVO, Generic VNFM and Generic EMS to orchestrate the deployment of the NS which creates an NSR. The successful deployment of the NS is indicated by the NSR getting into an ACTIVE state. The sequence of interactions is explained further in the steps below that correspond to the steps shown in Figure 6.4.

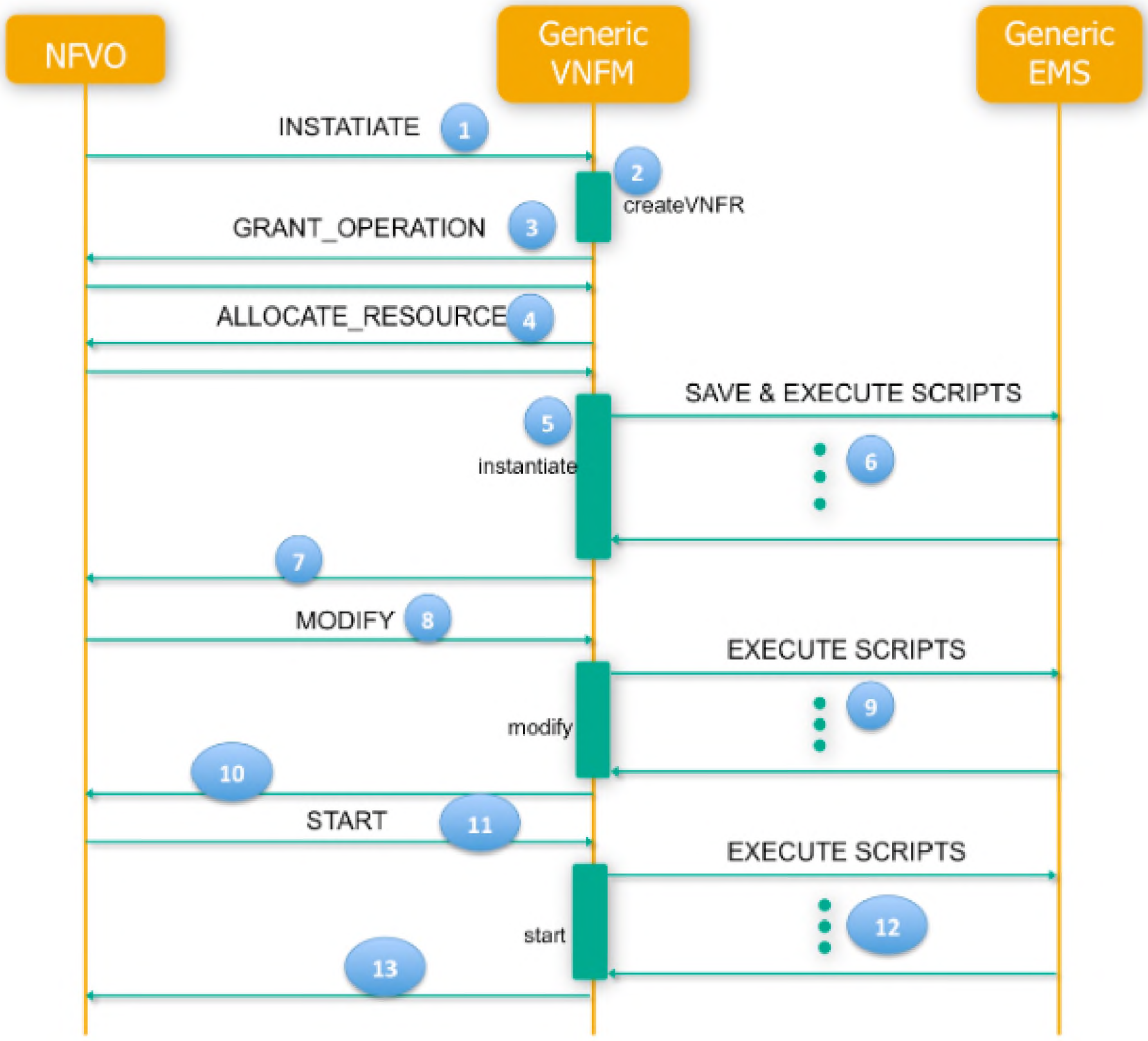


Figure 6.4: Sequence for the creation of a Network Service Record using Open Baton. Source [11].

1. The process begins with the NFVO sending an INSTANTIATE message to the Generic VNFM. The INSTANTIATE message contains information needed to create

the VNF Record. This information comes in the form of the VNFD and other parameters.

2. The Generic VNFM then creates the VNF Record.
3. A GrantOperation message is sent to the NFVO. This message is so that the NFVO checks if there is enough resources to create the VNF Record. If there are enough resources, a GrantOperation message with an updated VNF Record is sent back to the VNFM.
4. The VNFM creates an AllocateResource message using the VNF Record and sends it back to the NFVO.
5. The NFVO creates the VMs in the VIM, OpenStack for this thesis, and sends back the AllocateResources message to the VNFM. The instantiate method is then called by the VNFM.
6. The scripts from the VNF package are sent to the EMS and stored directly on the VM. The execution of the scripts is called by then VNFM as defined in the VNF Descriptor.
7. Once scripts have successfully been executed, the VNFM sends back the instantiate method the NFVO.
8. If the VNF has some dependencies, the Modify message is sent from the NFVO to the VNFM.
9. The VNFM executes scripts in the Configure lifecycle event as defined in the VNF Descriptor.
10. If there are no errors, the Modify message is sent back to the NFVO.
11. The NFVO then sends the Start message to the VNFM.
12. The VNFM calls the EMS to execute scripts defined in the Start lifecycle event.
13. Lastly, the Start message is sent back to the NFVO if there were no errors.

VNF Record and NSR States

Once an NSD is launched, the corresponding NSR is created in a NULL state. Figure 6.5 shows a diagram of the possible VNFR and NSR states which reflect the ETSI NFV states

[68] and the events resulting in the transition between states. Each VNF also has its own record referred to as a VNFR. The initial state for all NSR and VNFR is the NULL state. After the INSTANTIATE method is finished running in the VNFM, the VNFR state becomes INSTANTIATED. Once all the VNFRs are in the INSTANTIATED state, the NSR also goes into an INSTANTIATED state. In cases where the VNF is a target of a dependency, the MODIFY message is sent to the VNFM and eventually comes back to the NFVO which then sets the VNFR state to INACTIVE. When all the VNFRs are in the INACTIVE state, the NSR state is also set to INACTIVE. If there is a START message, it is sent to the VNFM and when it comes back to the NFVO, the state is changed to ACTIVE by the NFVO. Once all the VNFRs are ACTIVE the NSR also gets into an ACTIVE state [11]. The ACTIVE state indicates that the deployment was successful and the NS is now ready for use.

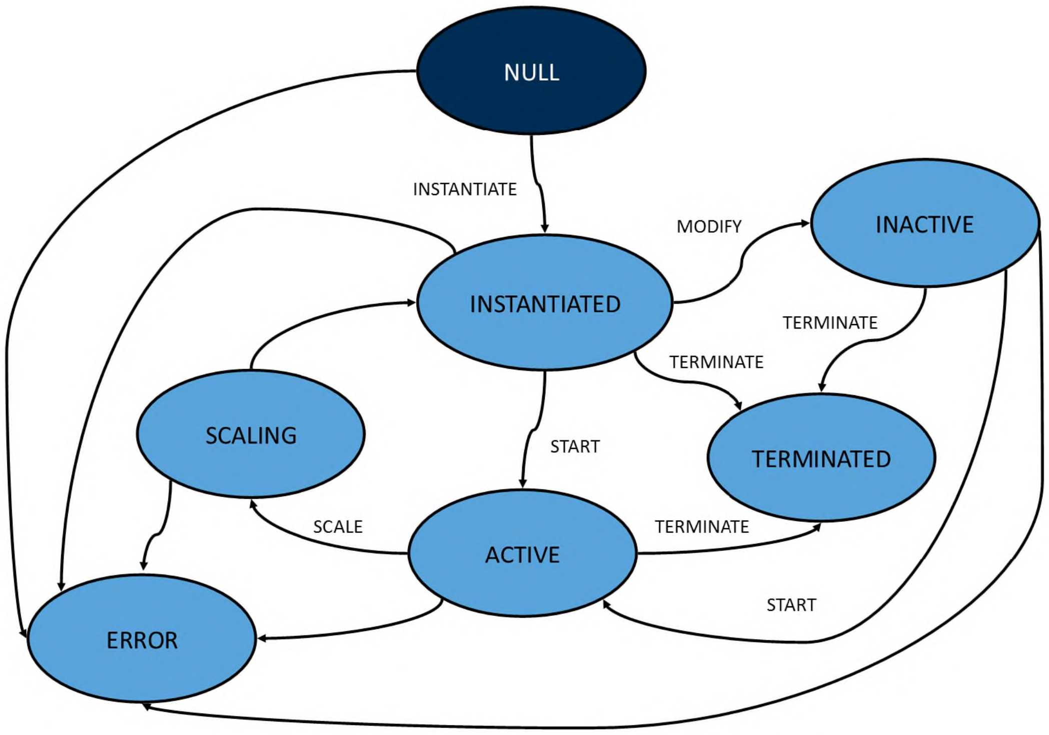


Figure 6.5: VNF Record States and Transitions. Source [11].

If something goes wrong during the deployment of the NS and any of the VNFR end up in an ERROR state, the NSR will get into an ERROR state. When an NSR is deleted, the TERMINATE method is called and the NSR gets into a TERMINATED state. The NFVO then sends a TERMINATE message to all the VNFM. The VNFR state is then set to TERMINATED on the NFVO and when all the VNFRs are in a TERMINATED state, it means that the NSR has been completely deleted from the system [11].

6.6 Services

One of the aims of Open Baton is to be an extendable platform where new services can be added using the Open Baton SDK as a tool for development. Two such services that are already available are the Autoscaling Engine (ASE) and the Fault Management System (FMS). Both systems were explored as they introduce the functions that are need to maintain QoS as well as make the NS more resilient under load or in the event of a fault.

6.6.1 Autoscaling Engine - ASE

The ASE is an external component that integrates with Open Baton to provide autoscaling services. This means that the VNF components that make up a network service can be scaled up or down based on a set threshold. The ASE works based on AutoScalePolicy that can be defined in the VNF Descriptor. AutoScalePolicy are defined using several parameters which are explained in Table 6.5.

Listing 6.5 shows an example of an AutoScalePolicy that can be added in the VNFD. The AutoScalePolicy defines a scaling-out operation that should happen if the measurements from the metric are greater than the threshold. In other words, the VNF is to be scaled out if the CPU utilisation by user processes exceeds 50%.

Listing 6.5: Example of an Autoscaling Policy

```
"auto_scale_policy":[
  {
    "name":"scale-out",
    "threshold":100,
    "comparisonOperator":">=",
    "period":30,
    "cooldown":60,
    "mode":"REACTIVE",
    "type":"WEIGHTED",
    "alarms": [
        {
          "metric":"system.cpu.load[percpu,avg1]",
          "statistic":"avg",
          "comparisonOperator":">",
          "threshold":0.70,
          "weight":1
        }
      ],
      "actions": [
        {
          "type":"SCALE_OUT",
          "value":"2",
          "target":"<target>"
        }
      ]
    }]
```

As explained in Table 6.5, alarms are what trigger the actions defined within an AutoScalePolicy. Alarms are defined using their own set of parameters which are explained in Table 6.6. Actions can be defined using three parameters, namely:

- **type** - This defines the type of action that should be executed. There are four types of scaling actions.
  - SCALE_OUT - scales out by the given number of instances.
  - SCALE_OUT_TO - scales out to a specifc number of instances.
  - SCALE_IN - scales in by the given number of instances.
  - SCALE_IN_TO - scales in to a specific number of instances.
- **value** - This is the value associated with the type of action. For example, a SCALE_OUT requires a value of the number of instances that should be scaled out. SCALE_OUT_TO requires the number to which instances should be scaled out.
- **target** - Target allows other VNFs to be scaled based on conditions of the VNF under consideration being met. If multiple VNFs have the same type, only one of the VNFs will execute the scaling action. If no target is defined, the scaling action is executed on the VNF that the policy is defined on.

Parameters	Meaning
name	The human-readable name of the AutoScalePolicy.
threshold	The percentage of sub alarms that should be fired before firing the AutoScalePolicy high-alarm.
comparisonOperator	The comparison operator used to check the percentages of alarms that have been fired. Where a value of 100 would mean that all weighted alarms have to be thrown before the AutoScalePolicy takes action and value of 50 means only half.
period	The period in which the conditions of the AutoScalePolicy are checked. For example, a period of 60 means that the conditions defined in the AutoScalePolicy are checked every 60 seconds.
cooldown	This is the time period that the VNF has to wait between scaling actions to ensure that the scaling operation takes effect first. This means that any other scaling operations will be rejected during this time.
mode	The way in which the alarms and conditions are recognised in the AutoScalePolicy. This can be *REACTIVE*, *PROACTIVE* and *PREDICTIVE*.
type	This defines how alarms are handled. Options are *VOTED*, *WEIGHTED* and *SIMPLE*.
alarms	This is a list of alarms that belong to the same AutoScalePolicy. The alarms are affected by the mode and type of the AutoScalePolicy and this affects the checks that lead to the triggering of the AutoScalepolicy.
actions	These are the actions that get executed as a result of the conditions and alarms defined being met. Once the conditions are met, corresponding actions are implemented.

Table 6.5: AutoScalePolicy Parameters. Source [11].

6.6.2 Fault Management System - FMS

The Fault Management System is a rule-based external component that uses alarms from the VIM to perform switch-to-standby or heal functions. It uses rules called *fault management policies* to generate alarms and define how to react to the alarms. The rule that defines how to react to the alarms is called a Drools rule. As with the AutoScalePolicy, the Fault Management Policy is also defined in the VNFD, particularly in the VDU. Listing 6.6 shows an example of a Fault Management Policy defined to check whether the P-CSCF server is available by pinging the server every 10 seconds. The parameters used in the Fault Management Policy are explained in Table 6.7.

Parameters	Meaning
metric	This is the metric to be considered when checking conditions and is made available through the monitoring system. Metric values include CPU idle time, memory utilisation and network traffic.
statistic	This defines how the final measurement result should be calculated over the group of instances. The possible values are: sum, min, max, avg and count.
comparisonOperator	This defines how the final measurement result should be compared against the threshold. The possible values are: >, <, >=, <=, = and !=.
threshold	This defines the value to which the final measurement result of the given metric is compared.
weight	This defines the weight of an alarm in relation to the rest of the rest of the alarms defined in the AutoScalePolicy. This allows for high priority alarms to be given a higher weight value.

Table 6.6: Alarm Parameters for Autoscaling. Source [11].

Listing 6.6: Example of a Fault Management Policy

```
"fault_management_policy":[
              {
                "name":"P-CSCF Server not available",
                "isVNFAlarm": true,
                "criteria":[
                  {
                    "parameter_ref":"agent.ping",
                    "function":"nodata(1m)",
                    "vnfc_selector":"at_least_one",
                    "comparison_operator":"=",
                    "threshold":"1"
                  }
                ],
                "severity":"CRITICAL",
                "period":10
              }
            ]
```

The criteria has its own set of parameters explained in Table 6.8.

Parameters	Meaning
name	The human-readable name of the Fault Management Policy.
isVNFAlarm	This is true if the alarm is of type VNF.
criteria	The criteria that defines a monitoring parameter and a threshold beyond which an alarm will get fired.
severity	The severity of the alarm.
period	How frequently the criteria is checked (in seconds).

Table 6.7: Fault Management Policy Parameters. Source [11].

Parameters	Meaning
parameter_ref	This a reference to a monitoring parameter already defined in the VDU. An example of how the monitoring parameters are added is shown in Listing 6.7.
function	The functions that is to be applied to the parameter.
vnfc_selector	This sets whether the criteria is met when all the VNF components cross the threshold or when there is at least one that crosses the threshold. The possible values are either *all* or *at_least_one*.
comparison_operator	The comparison operator for the threshold.
threshold	The value compared against the value from the parameter_ref.

Table 6.8: Fault Management Policy Parameters. Source [11].

Listing 6.7: Adding monitoring parameters to VDU for use in Fault Management Policy

```
"monitoring_parameter":[
                "agent.ping",
                "system.cpu.load[all,avg5]",
                "net.tcp.listen[80]",
                "system.cpu.util[,user]"
              ]
```

Zabbix is the only monitoring system currently supported by Open Baton therefore the monitoring parameters that can be referred to in a VDU and Fault Management Policy are the ones defined for the zabbix agent[1]. A Zabbix server communicates with the Zabbix agent in order to gather data about the VNF [11, 69]. The function used in the criteria also needs to be one of the trigger functions[2] defined by Zabbix. The example in Listing 6.6 uses the *nodata()* function which checks if no data has been received in the given time

[1]Available at: https://www.zabbix.com/documentation/3.0/manual/config/items/itemtypes/zabbix_agent
[2]Available at: https://www.zabbix.com/documentation/3.0/manual/appendix/triggers/functions

period, where a value of 1 means no data has been received and 0 means otherwise. The Zabbix monitoring system is discussed in more detail in the next chapter.

The FMS follows the workflow shown in Figure 6.6. The NFVO sends a message to initiate the deployment of the VNF by sending an INSTANTIATE message to the VNFM followed by the rest of the messages needed to get the VNF started and working. The VNFM uses the the messages to instantiate, configure and start the VNF. Once the the VNF is started, the VNFM informs the NFVO, which then sends a message to the FMS if there is a fault management policy defined in the descriptor. The FMS sends a CREATE FM message to the OpenStack VIM to create a fault management (FM) job on the VNF, followed by a CREATE THRESHOLD message to set the threshold. When this threshold is crossed, an alarm is set off and a message is sent to the FMS so that it uses the FM policy for that VNF to send the appropriate ACTION message to the NFVO which then implements the ACTION on the VNF. The possible actions are:

- **Heal**: A Heal lifecycle event must be included in the VNFD. Scripts in the Heal lifecycle event will be executed by the VNFM,
- **Switch to standby VNFC (stateless)**: A Switch to Standby action can be performed if there is a VNFC on standby in the VNF. The service is switched from the failing VNFC to the VNFC on standby automatically. A *high_availability* parameter can be added to the VDU to have a VNFC on standby as shown in Listing 6.8.

Listing 6.8: Adding high availability parameter for Switch to Standby

```
"high_availability":{
        "resiliencyLevel":"ACTIVE_STANDBY_STATELESS",
        "redundancyScheme":"1:N"
}
```

6.7 Summary

Open Baton fulfilled the roles on the NFVO as well as the VNFM. Nodes required to run the OpenStack Cloud and Open Baton were all deployed in VMware ESXi. Open Baton offers an external service, an Autoscaling Engine (ASE) that provides autoscaling of

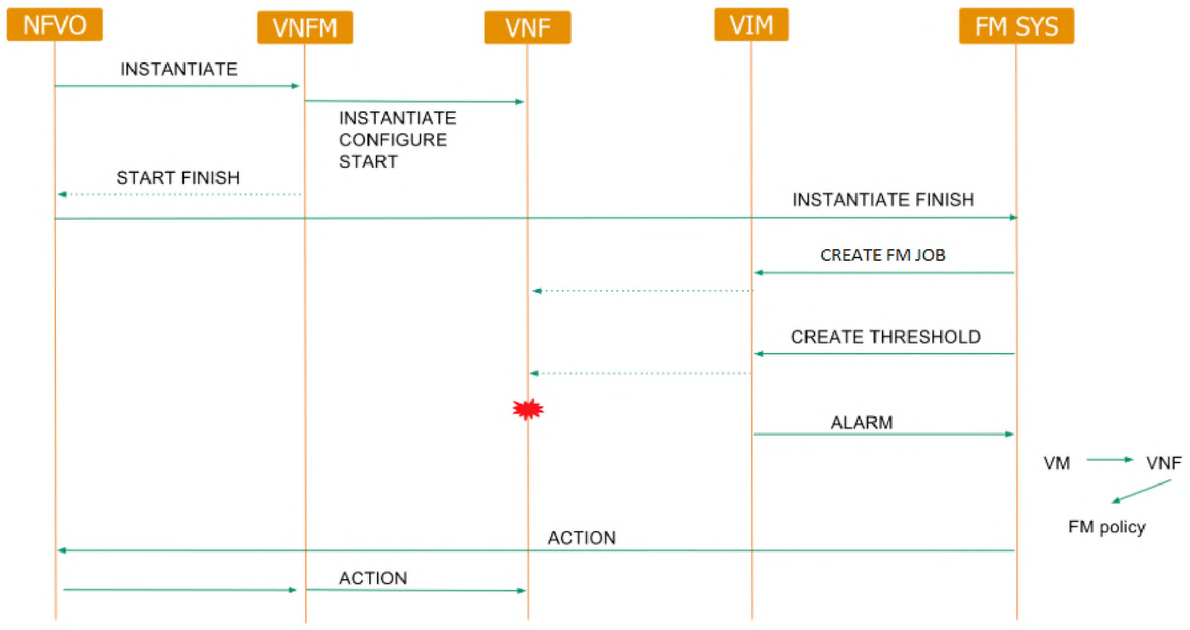


Figure 6.6: Fault Management System Workflow. Source [11].

VNFs. ASE requires the presence of a monitoring system, therefore the Zabbix monitoring system needed to be installed.Its Installation and use will be described in the next chapter.

Chapter 7

Phase 3: Monitoring

In order to get information to orchestrate and for example, automatically scale the network services, the Open Baton ASE needs to be able to get information from the VNFs. As mentioned earlier, Open Baton provides a plugin mechanism that it uses to interface with monitoring systems. In particular, a readily available monitoring plugin is the Zabbix plugin which allows Open Baton to use Zabbix as its monitoring system and communicate with the Zabbix server.

Zabbix is an open-source distributed monitoring solution for monitoring networks and applications, developed by Alexei Vladishev and is now developed and supported by Zabbix SIA [69, 70]. Zabbix can monitor different kinds of network devices including virtual machines, which is convenient for more complex network architectures. It uses a client-server model through the use of a Zabbix agent that runs on the host being monitored and gathers data to send to the Zabbix server. The Zabbix server is the central process of the monitoring system and interacts with Zabbix agents and proxies and sends the necessary notifications to subscribers. Communication between the Zabbix agent and Zabbix server supports encryption starting from Zabbix version 3. Configuration parameters and statistics can be accessed and configured via a web-based frontend for convenience [70].

This chapter will give details of how the Zabbix monitoring system was deployed and configured to work with the Open Baton deployment. It will also give insight into some initial experimentation done with monitoring the performance of VNFs running a simple network service.

7.1 Zabbix Server

Before the Zabbix plugin can be installed, a Zabbix server needs to already be installed and running. The two versions of the Zabbix server supported are Version 2.2 and Version 3.0. Zabbix server 3.0.22 was installed following instructions in [71]. Even though the Zabbix server may be installed on a server separate from the NFVO, it can simply be installed on the same server. Of course, when installing the Zabbix server on the same node as the NFVO, it is important that the machine used is powerful as both systems are heavy consumers of memory and CPU resources [71]. The Zabbix server used for this work was installed on the same server as the NFVO and did not cause performance problems because the server used is powerful enough to run both systems. Figure 7.1 shows the architecture of the deployed Zabbix system inside Open Baton.

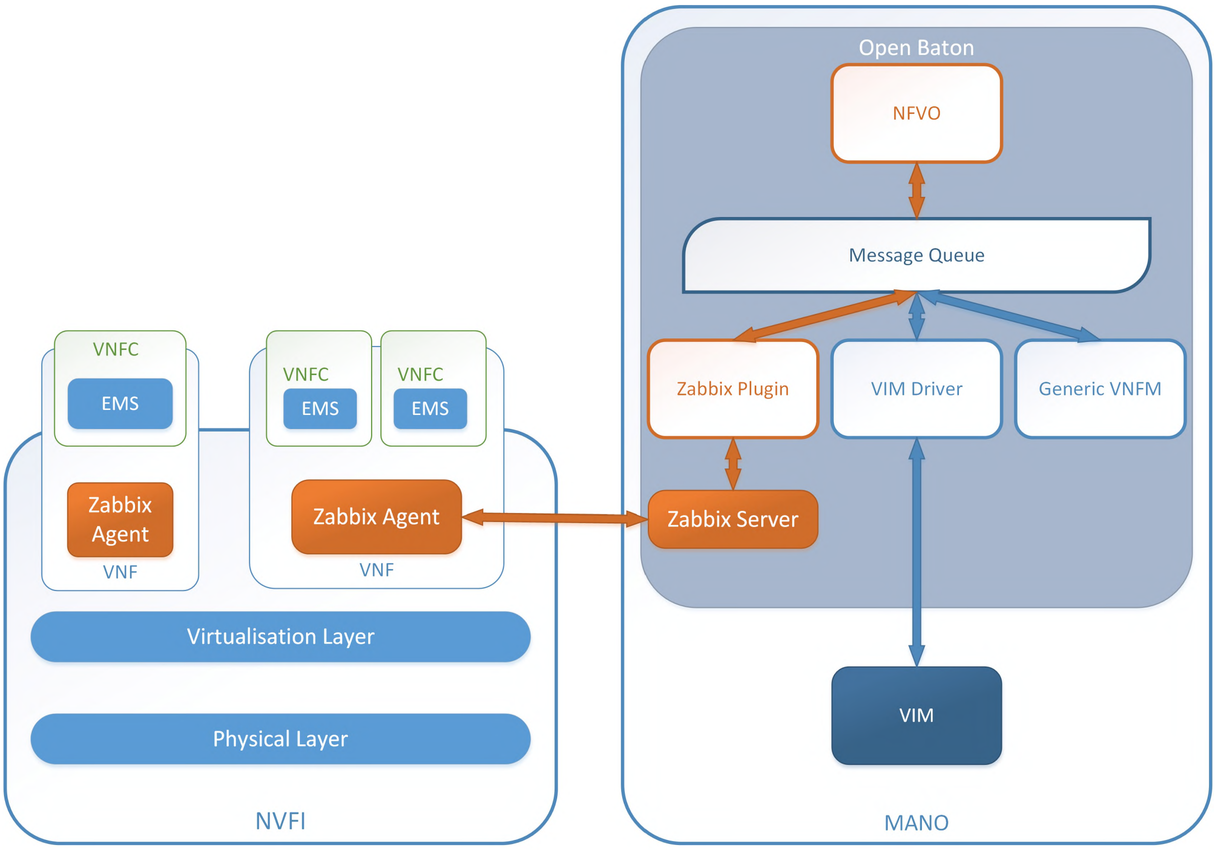
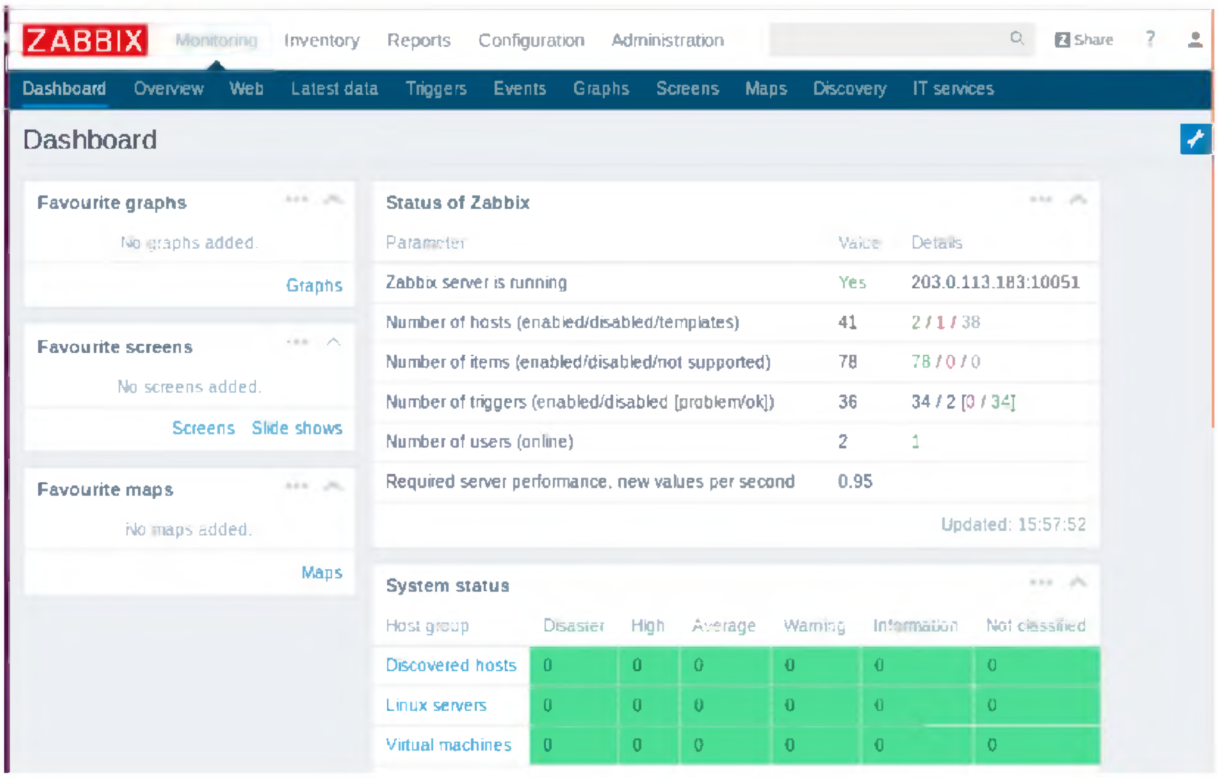


Figure 7.1: Deployed Zabbix Architecture.

Once the Zabbix server was installed and running, the Zabbix plugin could be installed on the same server as it is supposed to be installed on the server with the NFVO. From the Open Baton node, the Zabbix server web console is then available at http://localhost/zabbix and a snapshot of the dashboard can be seen in Figure 7.2. From the snapshot it can



Dashboard

Favourite graphs

No graphs added.

Graphs

Favourite screens

No screens added.

Screens Slide shows

Favourite maps

No maps added.

Maps

Status of Zabbix

Parameter	Value	Details
Zabbix server is running	Yes	203.0.113.183:10051
Number of hosts (enabled/disabled/templates)	41	2 / 1 / 38
Number of items (enabled/disabled/not supported)	78	78 / 0 / 0
Number of triggers (enabled/disabled [problem/ok])	36	34 / 2 [0 / 34]
Number of users (online)	2	1
Required server performance, new values per second	0.95	

Updated: 15:57:52

System status

Host group	Disaster	High	Average	Warning	Information	Not classified
Discovered hosts	0	0	0	0	0	0
Linux servers	0	0	0	0	0	0
Virtual machines	0	0	0	0	0	0

Figure 7.2: Snapshot of Zabbix Web Interface

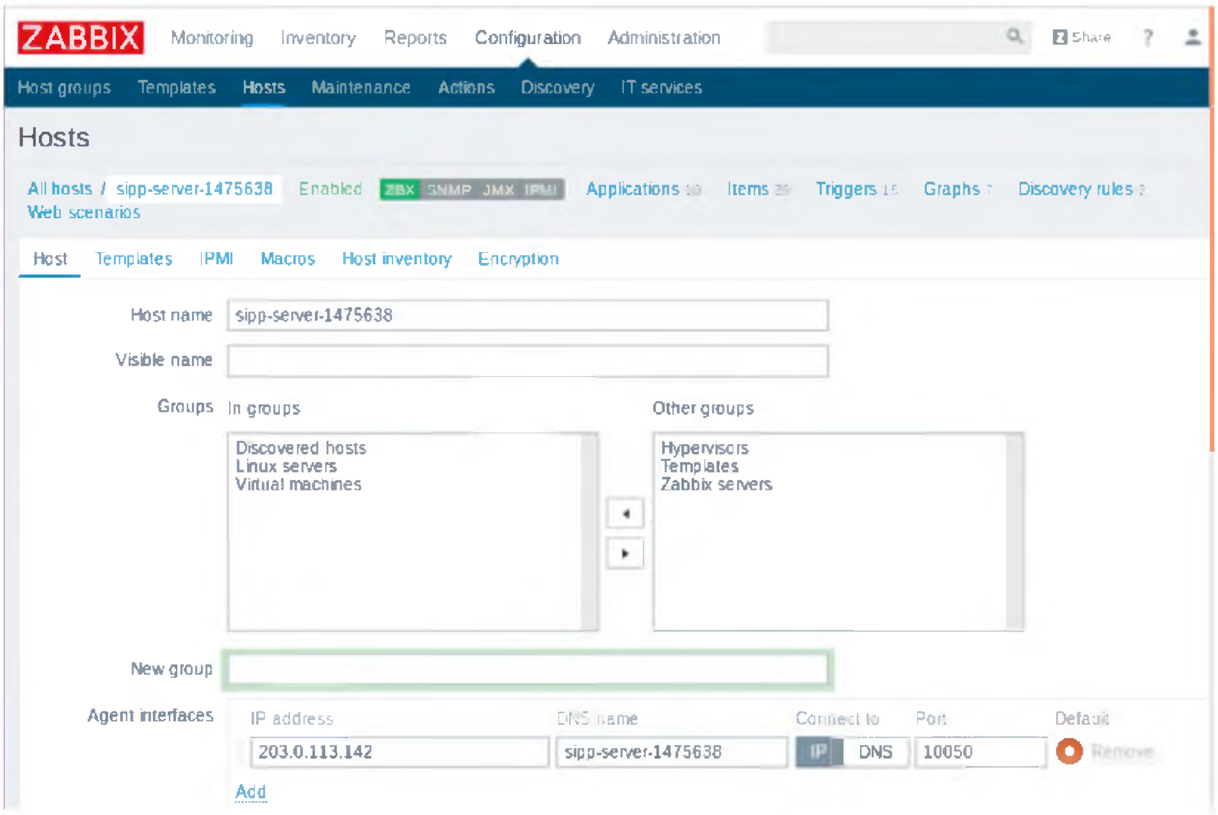
be seen that the Zabix server is up and running and its monitoring is available via port 10051. The snapshot also shows the number of hosts being monitored divided into the ones that are enabled, disabled and templates (entities that can be applied to hosts). Several configurations can be made through this Web Console including adding hosts to be monitored and adding triggers to get notified of events that are of interest.

7.2 Experiments - Configuring Zabbix

To become familiar with monitoring VNFs using Zabbix, a simpler network service was deployed. The IMS core network has a number of functional components, making it a more complex network service. The SIPp network service available from the Open Baton Marketplace was deployed for this purpose. The SIPp network service is simple and is created using just the NSD and does not require any separate VNF packages. There are two options for deploying the SIPp NS, one uses only private IPs and the other uses both private IPs and floating IPs. The NSD using floating IPs was chosen for this experiment. The network is made up of two VNFs namely, the SIPp server and the SIPp client. SIPp is a SIP performance tester. It is used to test the SIP protocol using basic SipStone user agent scenarios namely, UAC and UAS. The user agents establish and release calls using the INVITE and BYE request methods [72]. In the deployed SIPp NS, the SIPp client VNF represents the UAC and the SIPp server VNF represents the UAS.

7.2.1 Zabbix agent

When a network service is deployed using Open Baton and the Zabbix monitoring plugin is configured, a Zabbix agent is automatically installed on each VNF. The Zabbix agent monitors local applications and resources on the monitoring target it is deployed on. Once an agent is created and configured on the host, it still has to be registered with the server and this can be done via the dashboard. A host can simply be added by going to Configuration → Hosts → Create host. Figure 7.3 shows a snapshot of the configurations for adding the SIPp server.



Hosts

Host name	sipp-server-1475638
Visible name	
Groups — In groups	Discovered hosts, Linux servers, Virtual machines
Groups — Other groups	Hypervisors, Templates, Zabbix servers
New group	

Agent interfaces

IP address	DNS name	Connect to	Port	Default
203.0.113.142	sipp-server-1475638	IP DNS	10050	●

Figure 7.3: Adding the SIPp Server Host for monitoring in Zabbix

The *Host name* and *DNS name* are set to the name assigned to the SIPp server host in OpenStack. The host can be associated with certain host groups. *Groups* are used to logically group hosts and these groups are used when assigning access rights to hosts in different host groups [69]. The groups that were assigned to the SIPp server are: Discovered hosts, Linux servers and Virtual Machines. This is because it fits into all three categories. The *IP address* was set to the floating IP address that was attached to the host in OpenStack. The *DNS name* could have been omitted since the *Connect to* option is set to IP. Zabbix agents connect to the Zabbix server using *Port* 10050. The Template tab allows users to link *Templates* with a host. A template is a set of entities that can be applied to hosts. These include triggers, graphs and items which will be discussed further in following sections. There are several templates available to choose from but the template that was linked on the SIPp server host as shown in Figure 7.4 is the Template App Zabbix Agent. At this point the host can be added and saved. Both Figure 7.3 and 7.4 show that the status for the host is *Enabled* meaning that monitoring is enabled for that host. The *ZBX* option is in green which means that the host can be monitored using the Zabbix agent only, SNMP and the other options are unavailable for that host.

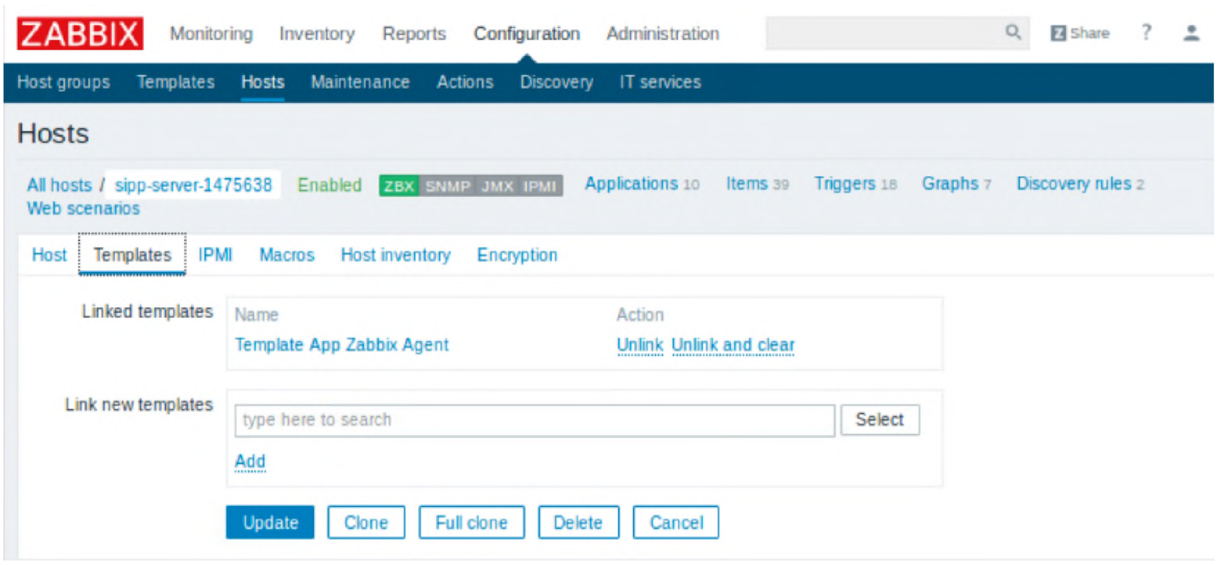
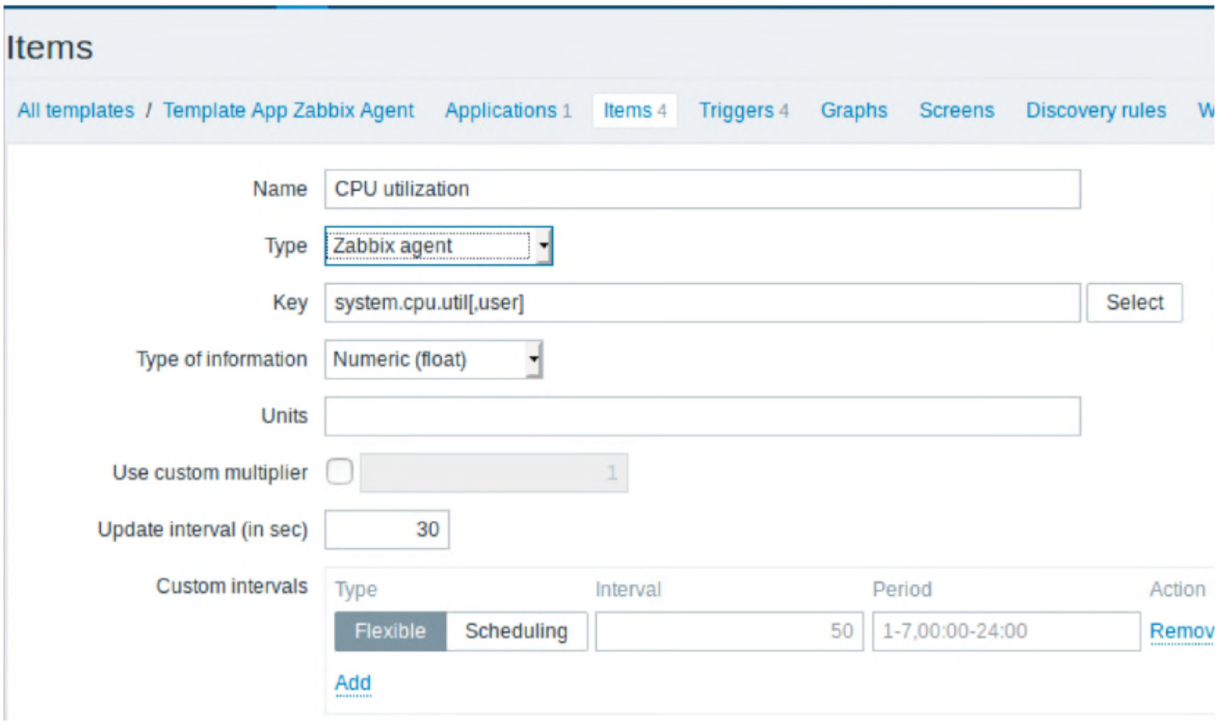


Figure 7.4: Adding a Template to the SIPp Server Host

7.2.2 Items

Items define the metric data that is to be received from a host [69]. Items can be created directly on hosts but a more efficient approach is to create Items inside a Template. Available Templates can be found under the Templates tab. Selecting Template App Zabbix Agent will display all the Items inside the Template as well as the option to create a new Item.

One of the important metrics when checking the performance of a server under load is the CPU utilisation. If CPU resources are getting overloaded, the server is under-performing. For this reason, this initial experiment focused on looking at CPU utilisation data. An Item with the configurations shown in Figure 7.5 was added to Template App Zabbix Agent. The *Name* is simply the name displayed for this Item and is set to CPU utilization. The *Type* is set to Zabbix agent, which means that communication with the Zabbix agent will be used to collect data. The *Key* has to be one of the pre-defined keys that come with the Zabbix agent is set as the Type parameter. It is a technical representation of the name of the piece of information that is to be gathered. The *Key* parameter was set to system.cpu.util[,user] meaning that information about CPU utilisation by user applications was to be collected. The Key definition for CPU utilisation is system.cpu.util[<cpu>, <type>, <mode>] The *cpu* parameter is the CPU number and defaults to all, the *type* is the specific CPU consumer for which data is to be collected. Possible values are user, idle, nice, interrupt, system, iowait, steal, softirq, guest, guest_nice. User is the parameter used to collect information of CPU utilisation by the applications running on the host. The possible values for *mode* are avg1 (default), avg5 and avg15 which say get the average over a given time period in minutes. The Item configured in this experiment gives CPU utilisation information for all CPUs, for the *type* user and using one minute averages.


Items

All templates / Template App Zabbix Agent | Applications 1 | Items 4 | Triggers 4 | Graphs | Screens | Discovery rules | W

Name	CPU utilization
Type	Zabbix agent
Key	system.cpu.util[,user]
Type of information	Numeric (float)
Units	
Use custom multiplier	1
Update interval (in sec)	30

Custom intervals

Type	Interval	Period	Action
Flexible Scheduling	50	1-7,00:00-24:00	Remov

Figure 7.5: Adding an Item to a Template

7.2.3 Triggers

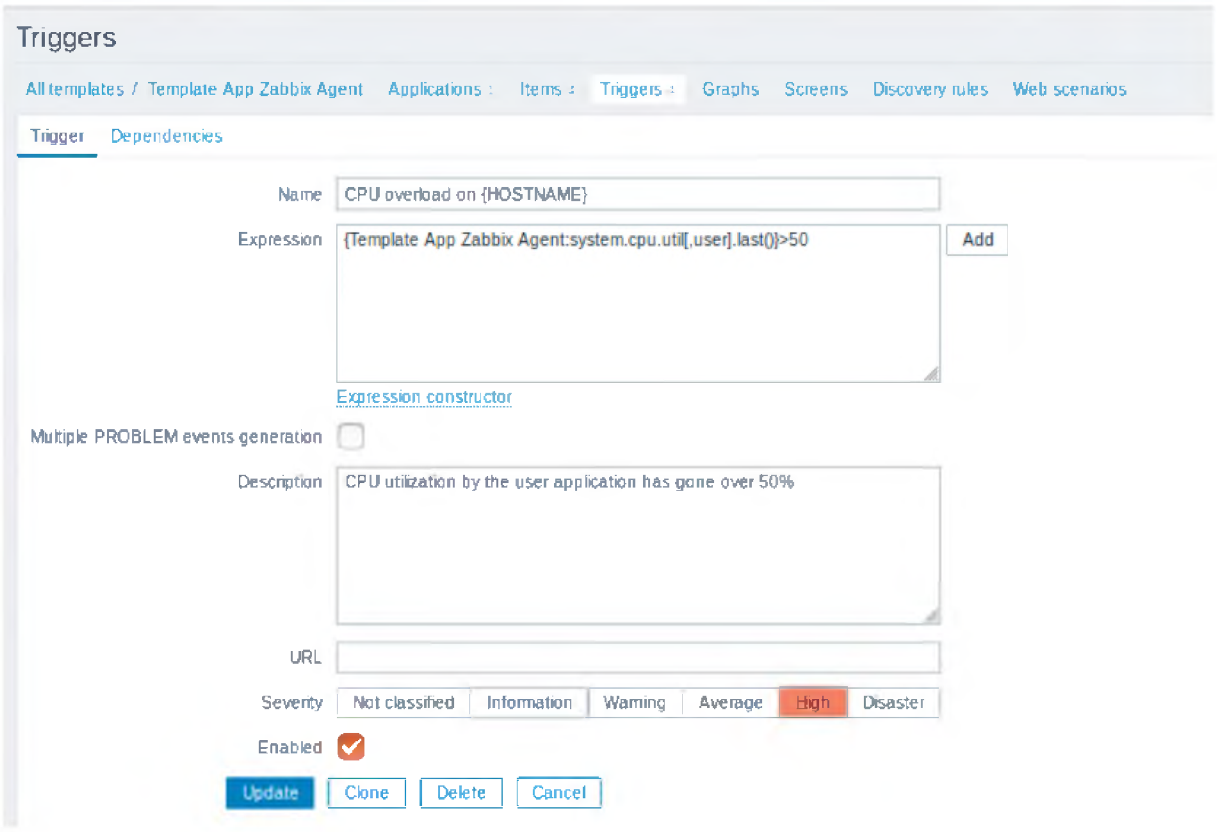
Items are very useful as they make it possible to receive data on a specific metric. However, this is the only thing they do: they collect data and do not do any evaluation. To add the ability to automatically evaluate incoming data, *Triggers* can be defined based on the Items. A Trigger gives an expression used to define a threshold for acceptable data values for when the host is in an OK state. When the incoming data exceeds the set threshold, the Trigger goes off and get into a PROBLEM state. If the data goes back to more acceptable values, the Trigger goes back to an OK state [69].

A Trigger was defined as per the configurations shown in Figure 7.6. The *Name* of the Trigger was set to '*CPU overload on {HOSTNAME}*' and this is the name that will represent the Trigger inside Zabbix. Zabbix supports the use of variables referred to as Macros. The syntax for Macros is {MACRO}, which resolves to a valid value within the context of that deployment of Zabbix [69]. In this case, {HOSTNAME} will resolve to the name of the specific host being monitored. The syntax for Trigger expressions can simply be expressed as:

{<*server*>:<*key*>.<*function*>(<*parameter*>)}<*operator*><*constant*>

Where,

- **server** is the host or Template the Trigger is for.
- **key** is the key for the Item whose data is to be analysed.
- **function** is used to refer to specific values from the collected data. Possible values include avg() for an average of the values, last() for the last value and sum() for a sum of the values.
- **parameter** provides a parameter value because most of the numeric functions accept a parameter. For example, **avg(300)** gives the average of all values within 300 seconds.
- **operator** is mainly standard mathematical operators, such as: *, /, +, -, < and >.
- **constant** is the value against which the operator evaluates the data from the agent {<*server*>:<*key*>.<*function*>(<*parameter*>)}.



Name: CPU overload on {HOSTNAME}

Expression: {Template App Zabbix Agent:system.cpu.util[,user].last()}>50

Multiple PROBLEM events generation: ☐

Description: CPU utilization by the user application has gone over 50%

URL:

Severity: Not classified, Information, Warning, Average, [✓ High], Disaster

Enabled: ✓

Figure 7.6: Creating a Trigger.

As mentioned earlier, Templates are in essence sets of entities such as Items and Triggers. The expression used for this experiment creates a Trigger for Template App Zabbix Agent. This means that all hosts linked to this template will inherit this Trigger. An advantage of doing this is that the Trigger can be created once and automatically applied to all new hosts that use this Template. The *Expression* parameter uses the Item key to refer to the Item the Trigger is for, in this case, system.cpu.util[,user]. The function used is last(), meaning that only the last value is being looked at. The 'greater than' operator is used with a somewhat arbitrary constant of 50. The *Severity* of the trigger can be classified as (in order of severity): Not classified, Information, Warning, Average, High and Disaster. In this case it was set to a severity of High.

7.3 Experiments - Running the SIPp Scenarios

In order to monitor the SIPp server, call scenarios need to be sent to it from the SIPp client that has traffic to handle. The client emulates a number of users calling the server. The SIPp executable has some basic scenarios embedded and ready for execution. The most common scenarios are simply named **uas** and **uac**. The server listens for request messages from the client and the server scenario, uas, was run using the command:

```
screen -d -m -S server sipp -sn uas
```

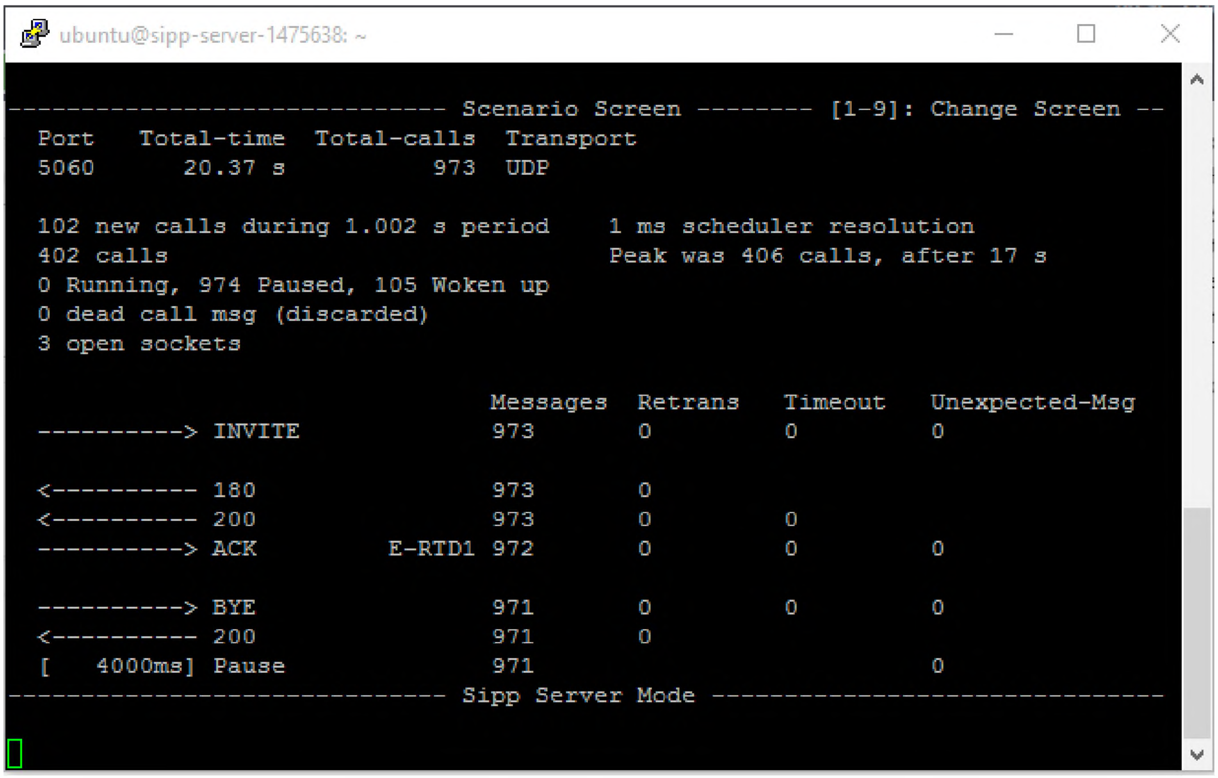
The client sends requests to the server and the client scenario, uac, was running using the command:

```
screen -d -m -S client sipp -sn uac 192.168.100.26:5061 -d 10 -r 100 -rp 1000 -rate_increase
10 -fd 10 -rate_max 1000 -trace_logs
```

The client scenario have several parameters added to it to control the behaviour of the traffic sent to the server. The server is available at the private IP address of 192.168.100.26 and the server is listening on port 5061. The other configurations are explained below [73]:

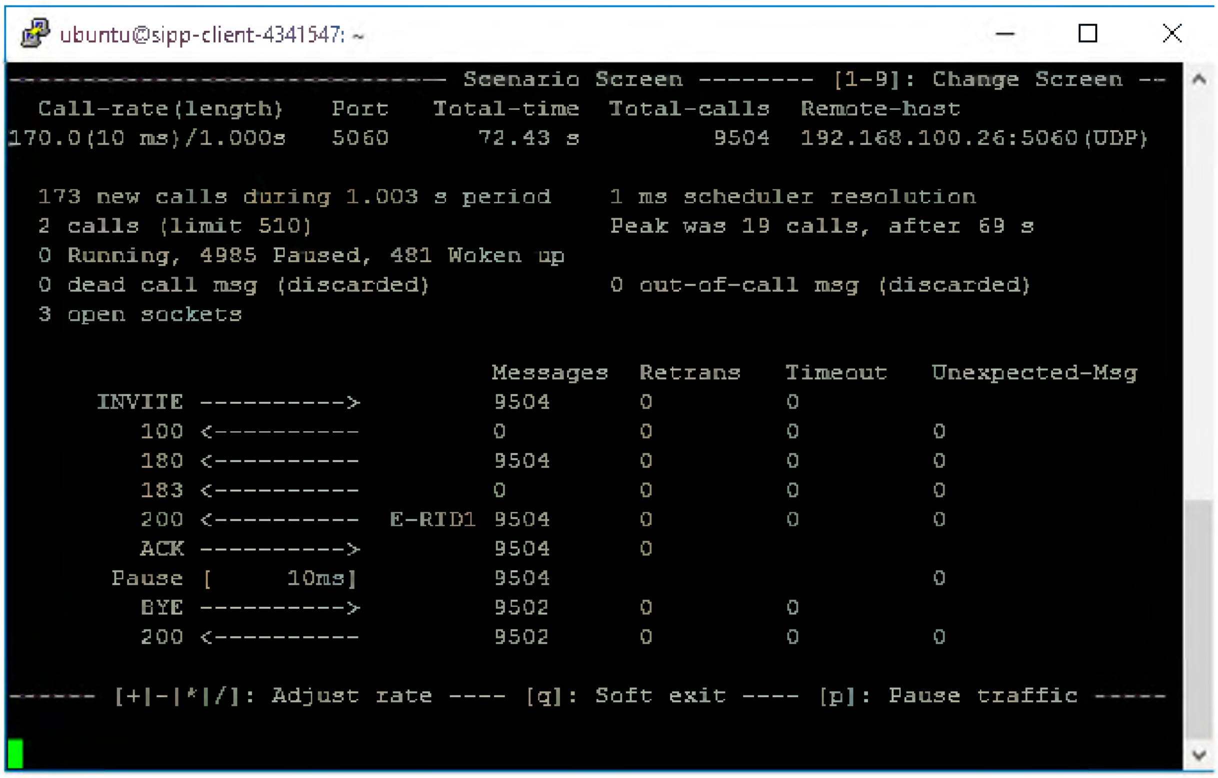
- **-d** Sets the length of the 'pause' that represents the length of the call (in milliseconds). The default value is 0.
- **-r** Sets the call rate (in calls per seconds).
- **-rp** Specifies the rate period (in milliseconds) for a given call rate. The default value is 1 second. If for example, the call rate is set to 10 and the rate period set to 3000, i.e -r 10 -rp 3000, it means 10 calls every 3 seconds.
- **-rate_increase** Specifies the rate increase every -fd seconds, allowing for load to be increased for different logging periods.
- **-fd** Sets the statistics dump log report frequency (in seconds). The default value is 60. If for example, the rate increase is set to 15 and the frequency set to 20, i.e -rate_increase 15 -fd 20, it means that calls must be increase by 15 every 20 seconds.
- **-rate_max** If the rate_increase is set, rate_max sets the maximum rate after which it should quit.
- **-trace_logs** Allows logs to be traced and saved in <scenario file name>_<pid>_logs.log.

Screenshots of the scenario screens can be seen in Figure 7.7 and 7.8.



```
------------------------------ Scenario Screen -------- [1-9]: Change Screen --
  Port   Total-time  Total-calls  Transport
  5060      20.37 s          973  UDP

  102 new calls during 1.002 s period    1 ms scheduler resolution
  402 calls                              Peak was 406 calls, after 17 s
  0 Running, 974 Paused, 105 Woken up
  0 dead call msg (discarded)
  3 open sockets

                               Messages  Retrans   Timeout   Unexpected-Msg
  ----------> INVITE           973       0         0         0

  <---------- 180              973       0
  <---------- 200              973       0         0
  ----------> ACK        E-RTD1 972      0         0         0

  ----------> BYE              971       0         0         0
  <---------- 200              971       0
  [   4000ms] Pause            971                           0
------------------------------ Sipp Server Mode --------------------------------
```

Figure 7.7: Server Listening for SIP calls.



```
------------------------------ Scenario Screen -------- [1-9]: Change Screen --
  Call-rate(length)   Port   Total-time  Total-calls  Remote-host
170.0(10 ms)/1.000s   5060      72.43 s         9504  192.168.100.26:5060(UDP)

  173 new calls during 1.003 s period    1 ms scheduler resolution
  2 calls (limit 510)                    Peak was 19 calls, after 69 s
  0 Running, 4985 Paused, 481 Woken up
  0 dead call msg (discarded)            0 out-of-call msg (discarded)
  3 open sockets

                               Messages  Retrans   Timeout   Unexpected-Msg
      INVITE ---------->       9504      0         0
         100 <----------       0         0         0         0
         180 <----------       9504      0         0         0
         183 <----------       0         0         0         0
         200 <---------- E-RTD1 9504     0         0         0
         ACK ---------->       9504      0
       Pause [      10ms]      9504                          0
         BYE ---------->       9502      0         0
         200 <----------       9502      0         0         0

------ [+|-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----
```

Figure 7.8: Client Releasing Calls.

7.4 Experiments - Monitoring with Zabbix

In Section 7.2 the SIPp server was registered as a host in the Zabbix server and from that point, the SIPp server was being monitored by Zabbix. Under the Monitoring tab in Zabbix, there is an option to view information in a graph format. Figure 7.9 shows a screenshot of a graph showing the CPU utilisation trend as traffic is going through the SIPp server.

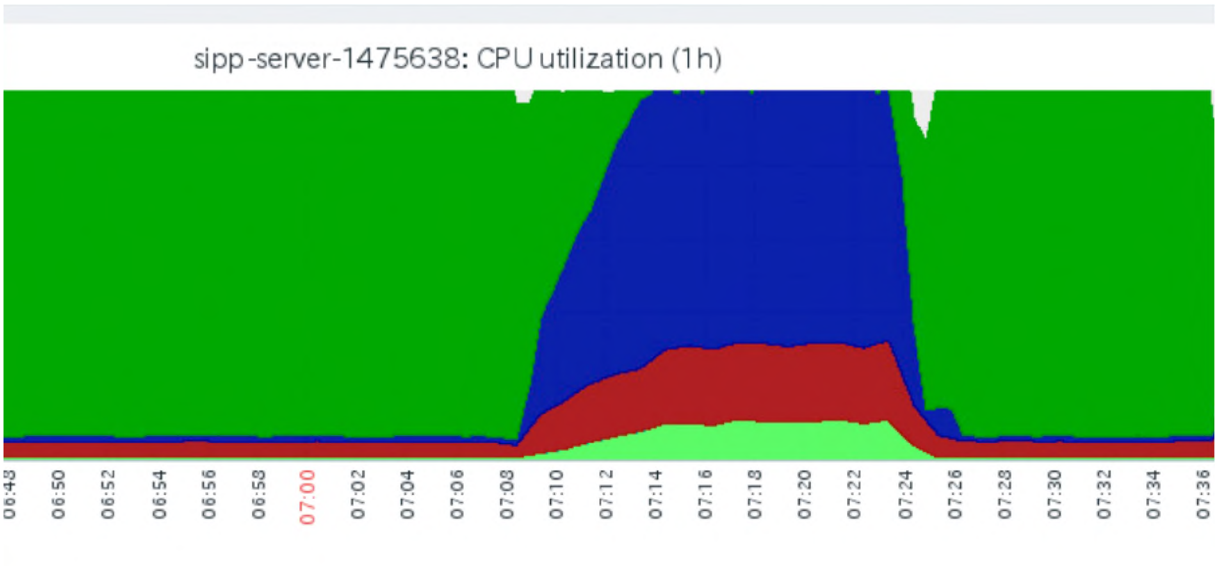
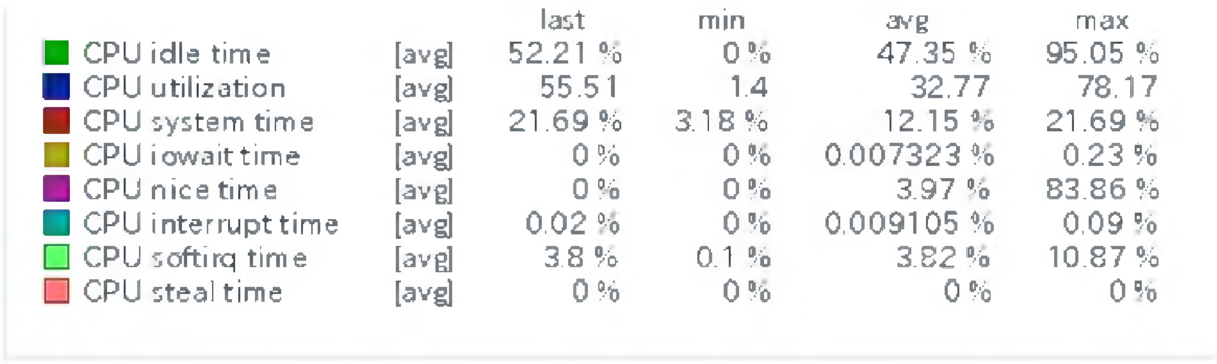


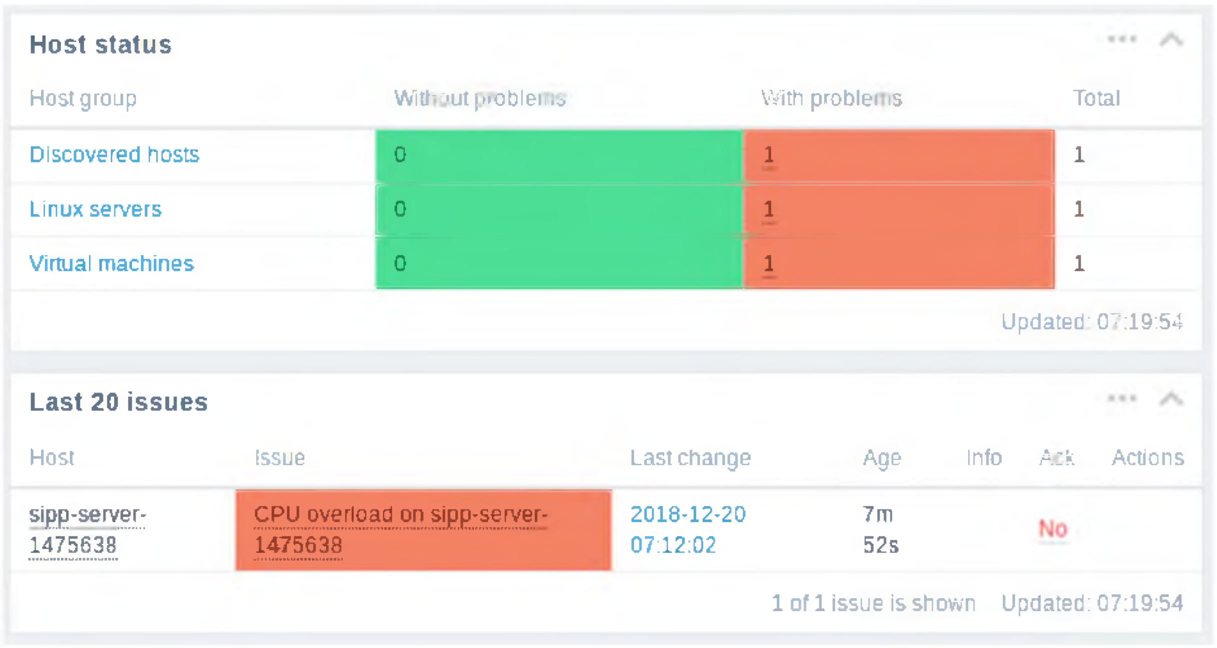
Figure 7.9: Screenshot of Graph showing CPU utilisation on SIPp server Host.

Figure 7.10 shows a key for the graph and some statistics for when the CPU utilisation had gone over the threshold. It can be seen that the *last* value (which is our value of interest) for CPU utilisation at this point was 55.51 which is over the threshold of 50. This means that we expect the trigger that we set to go off and go into a 'PROBLEM' status. Figure 7.11 shows a screenshot of the monitoring dashboard. On it, it can be seen that there is an issue with the SIPp server. The issue flagged by the dashboard corresponds with the Trigger configured and the message shown is "CPU overload on sipp-server-1475638". The graph in Figure 7.9 shows that the CPU utilisation goes down when the maximum call rate is reached as expected. As mentioned in Section 7.2.3, the Trigger should automatically go back into an 'OK' state once the values are back within an acceptable range. Figure 7.12 shows this expected behaviour and it can be seen that the dashboard nolonger flagged any issues once CPU utilisation was within an acceptable range.



		last	min	avg	max
🟩 CPU idle time	[avg]	52.21 %	0 %	47.35 %	95.05 %
🟦 CPU utilization	[avg]	55.51	1.4	32.77	78.17
🟥 CPU system time	[avg]	21.69 %	3.18 %	12.15 %	21.69 %
🟨 CPU iowait time	[avg]	0 %	0 %	0.007323 %	0.23 %
🟪 CPU nice time	[avg]	0 %	0 %	3.97 %	83.86 %
🟦 CPU interrupt time	[avg]	0.02 %	0 %	0.009105 %	0.09 %
🟩 CPU softirq time	[avg]	3.8 %	0.1 %	3.82 %	10.87 %
🟥 CPU steal time	[avg]	0 %	0 %	0 %	0 %

Figure 7.10: Monitoring Graph key showing CPU Utilisation Statistics when CPU was Overloaded.



Host status

Host group	Without problems	With problems	Total
Discovered hosts	0	1	1
Linux servers	0	1	1
Virtual machines	0	1	1

Updated: 07:19:54

Last 20 issues

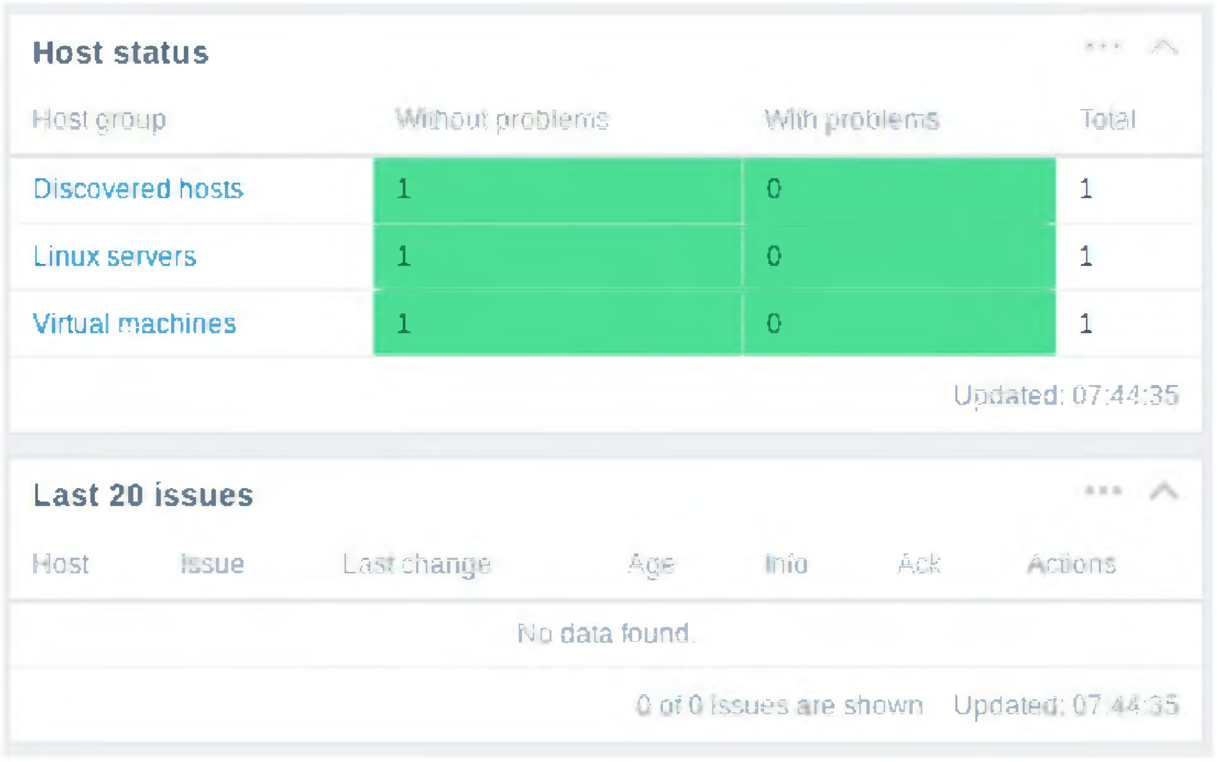
Host	Issue	Last change	Age	Info	Ack	Actions
sipp-server-1475638	CPU overload on sipp-server-1475638	2018-12-20 07:12:02	7m 52s		No	

1 of 1 issue is shown Updated: 07:19:54

Figure 7.11: Monitoring Dashboard when Trigger for CPU Utilisation is set off on SIPp server.

7.5 Summary

In order to perform scaling and fault management, the Open Baton Autoscaling Engine and Fault Management Systems need to receive statistics on how the different VNFs are consuming resources such as CPU, memory and storage. If these resources are under-provisioned, it will result in the VNF nodes under-performing. This creates the need to incorporate a monitoring system to the deployment. Open Baton provides a plugin mechanism for interfacing with a monitoring system. Currently, the available monitoring plugin is the Zabbix plugin through which Open Baton interfaces with the Zabbix monitoring system.



Host status

Host group	Without problems	With problems	Total
Discovered hosts	1	0	1
Linux servers	1	0	1
Virtual machines	1	0	1

Updated: 07:44:35

Last 20 issues

Host	Issue	Last change	Age	Info	Ack	Actions

No data found.

0 of 0 issues are shown Updated: 07:44:35

Figure 7.12: Monitoring Dashboard when Trigger for CPU Utilisation returns to OK State on SIPp server.

This chapter looked at how Zabbix monitoring system was incorporated into Open Baton and some experimentation that was done to familiarise with the monitoring system. The Zabbix server was installed on the same server as Open Baton, for simplicity. The Zabbix server can be accessed via a web console and this can be used to configure and view different elements inside the Zabbix environment. Zabbix uses a client-server model and client end is in the form of a Zabbix agent. If a network service is deployed using the Open Baton NFVO, the Zabbix agents are installed when the VNF nodes are created. Meaning that the VNFs are ready to be registered with the server. This can simply be done by configuring a new host host via the Zabbix web interface after-which the VNF node can be monitored.

Zabbix uses elements called Items that allow data to be collected from a host. This is done by using a Key which is one of several pre-defined keys for collecting a specific metric of data. An Item can be created for a specific host or it can be created for a Template. A Template is a grouping of several Items and other elements that need to be applied across multiple hosts. This simplifies the process as it takes away the need to create Items on each host individually. Furthermore, in order to set thresholds for an acceptable range of values for a particular Item, a Trigger has be created using the Item Key and other parameters to create an Expression. The Expression is then used to check whether the host is in an ‘OK’ or ‘PROBLEM’ status.

For simple experimentation, the SIPp network service was launched using Open Baton and deployed the SIPp server and SIPp client VNFs inside the OpenStack cloud. The SIPp server was registered as a host for monitoring in Zabbix and linked to a Template named Template App Zabbix Agent. An Item to get data on CPU utilisation was added to the Template. Lastly, a Trigger was created to flag if the CPU utilisation has gone beyond the acceptable range. When increasing traffic was received from the SIPp client, the Zabbix dashboard flags that there is a host with problems and shows the name of the SIPp server and what the issue is. It was also shown that eventually when the load decreases, the Trigger automatically goes back to an ‘OK’ state when data values normalise.

Chapter 8

Phase 4: IMS Virtualisation, Autoscaling and Fault Management

Chapter 3 introduced two open source implementations of the IMS core network namely, Open IMS Core and Project Clearwater. The Open IMS core network is simple in its architecture and makes use of the least number of VNF components. Project Clearwater is more complex as it includes a few more VNF components than the standard IMS core network.

A mapping of the components of Open IMS core and Project Clearwater is shown in Figure 8.1. Open IMS Core is made up strictly of the standard IMS network functions and is a more mature implementation. Project Clearwater is made up of the standard IMS core functions but also includes other additional features such as HSS cache, XDMS and a user provisioning portal.

8.1 Initial Experiments with IMS

There are several options available for deploying an IMS network. A simple and basic deployment can have all the different core functions on a single instance. Other deployments have an instance for each network function. Each node can be installed and configured individually, in an "hands on" fashion. A faster approach is to use scripts to deploy the network using an orchestrator. All theses different approaches were utilised in our experiments.

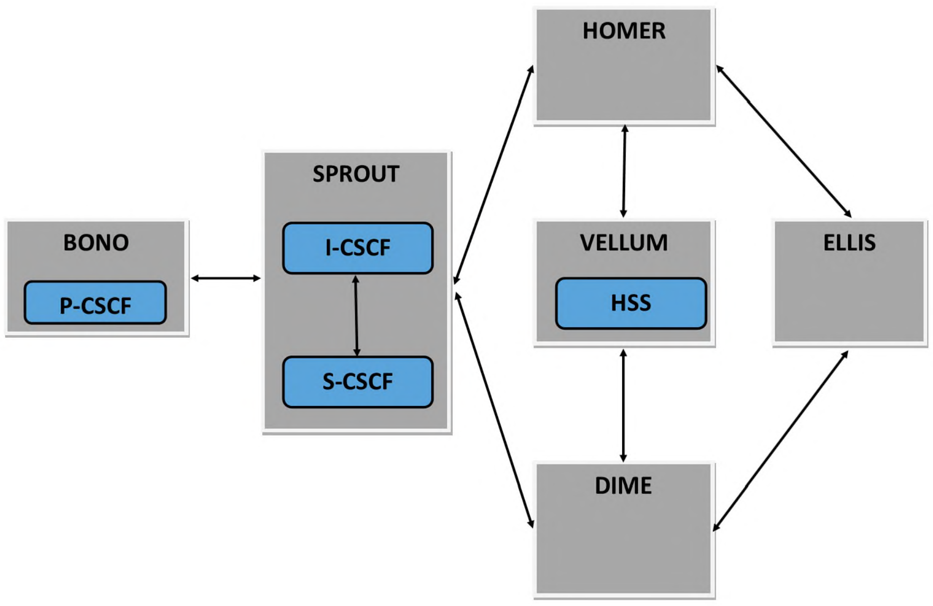


Figure 8.1: Relationship between Open IMS Core and Project Clearwater.

8.1.1 Simple Open IMS Core Deployment

A basic Open IMS Core network was installed as a single instance in OpenStack. This gave a quick and easy way to get familiar with the IMS core network. The main advantage of deploying the IMS network this way is that it is simple. However, a very big disadvantage is that it does not support network flexibility and network function scalability. In order to automatically scale the network appropriately, the network functions need to be implemented on separate instances so that auto-scaling can be based on the behaviour of each individual network function.

8.1.2 Deploying Project Clearwater

Clearwater can be installed in various ways, the simplest being the all-in-one image that can be used to install all the Project Clearwater functions onto a single node. The installation is done using an image with AMI (Amazon Machine Image), Amazon EC2 or an OVF (Open Virtualization Format) image for VMware and VirtualBox. This is similar to the installation of Open IMS Core on one node and therefore has the same limitation of not being scalable and having limited performance. However, it is also great for familiarising with the Clearwater network.

Another alternative is using the Chef orchestrator. Chef is an open source orchestrator that defines a system's infrastructure as code and continuously evaluates the system against a desired state and tries to correct any shift from this state [74]. This is currently only supported on Amazon EC2 and requires the DNS to be handled by Amazon's Route53. This option allows the deployment of a larger scale network that can be scaled. However, it is not supported for the OpenStack environment in which the network is to be deployed.

Clearwater can also be installed manually by using Debian packages to hand-configure each VNF instance. This is the recommended option for an environment that does not support Chef or if the DNS is not Amazon's Route53. For this option, all the virtual machines, DNS entries and firewalls are manually configured. This deployment proved not to be easy nor simple. As stated, the Clearwater network is more complex than the Open IMS Core network because of all extra unconventional functional parts. As a result, it is very easy to run into problems during the configuration of the nodes, making it a challenge to get a functional IMS network service.

8.1.3 Deployment using the Heat Orchestrator

Using an orchestrator to create a network service is a more efficient approach as most of the configurations are done automatically and the chances of misconfiguring components are lower. Additionally, this tends to be faster as the orchestrator can handle the installation and configuration of multiple VNFs simultaneously. This is very important in NFV because one of the aims of NFV is to simplify network service deployment and management.

OpenStack provides an Orchestration service called Heat. The Heat orchestration engine was introduced in Chapter 4 and is used to launch cloud applications based on templates. The templates are text files containing the network description and are executed to create the defined network by creating the necessary VMs [55]. Heat templates can be used to create simple networks with VMs that do not host any VNFs when they are deployed, an example of such is shown in Chapter 4.

Metaswitch provides Heat templates for deploying Clearwater in OpenStack [75]. The templates handle different elements of the deployment:

- **clearwater.yaml** - the top-level template
- **bono.yaml** - configures the Clearwater bono node
- **dime.yaml** - configures the Clearwater dime node
- **ellis.yaml** - configures the Clearwater ellis node
- **homer.yaml** - configures Clearwater homer node
- **sprout.yaml** - configures the Clearwater sprout node
- **vellum.yaml** configures the Clearwater vellum node
- **dns.yaml** - configures the DNS server exposing dynamic DNS using DNSSEC (Domain Name System Security Extensions)
- **network.yaml** - configures the network for Clearwater to use
- **security-groups.yaml** - configures security groups needed by Clearwater. Security groups are firewall rules to open certain ports for communication.

The Clearwater network was created by running:

openstack stack create clearwater -f clearwater.yaml -e cw-environment.yaml

where *cw-environment.yaml* contains the parameters needed by the top-level template to configure all the network components. The parameters are based on the service requirements and resources available in the OpenStack environment. The resulting network topology is shown in Figure 8.2. The nodes created have the minimum requirements of 1 vCPU, 2GB RAM and are running Ubuntu 14.04.

Figure 8.2 shows a provider network to which subnets connect to get access to the Internet. There are two routers each connected to a subnet, namely clearwater-private-management and the clearwater-private signalling. The private management subnet is responsible for management services such as SNMP, SSH and provisioning. The private signalling subnet is responsible for signalling services such as SIP, HTTP and Diameter communication. Each node has separate virtual network interfaces that connect to each subnet, with the exception of the Ellis node which is only part of the private management subnet because it is only used for provisioning.

Once the Clearwater network is running, numbers can be allocated via the ellis web URL. Figure 8.3 shows the Clearwater login page available using the ellis URL.

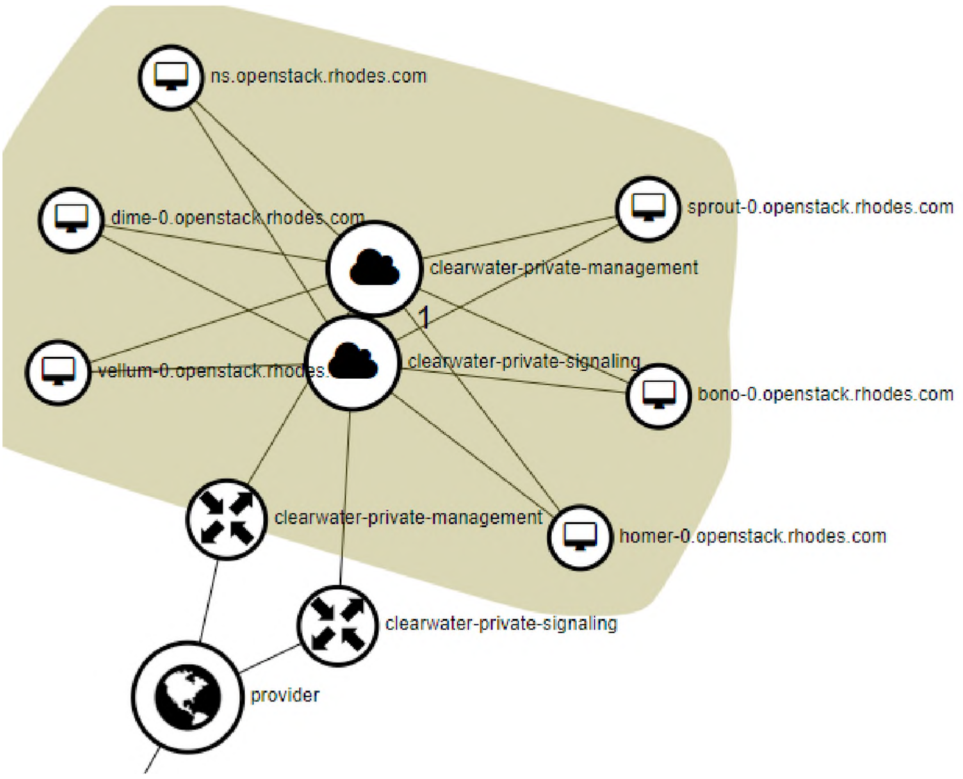
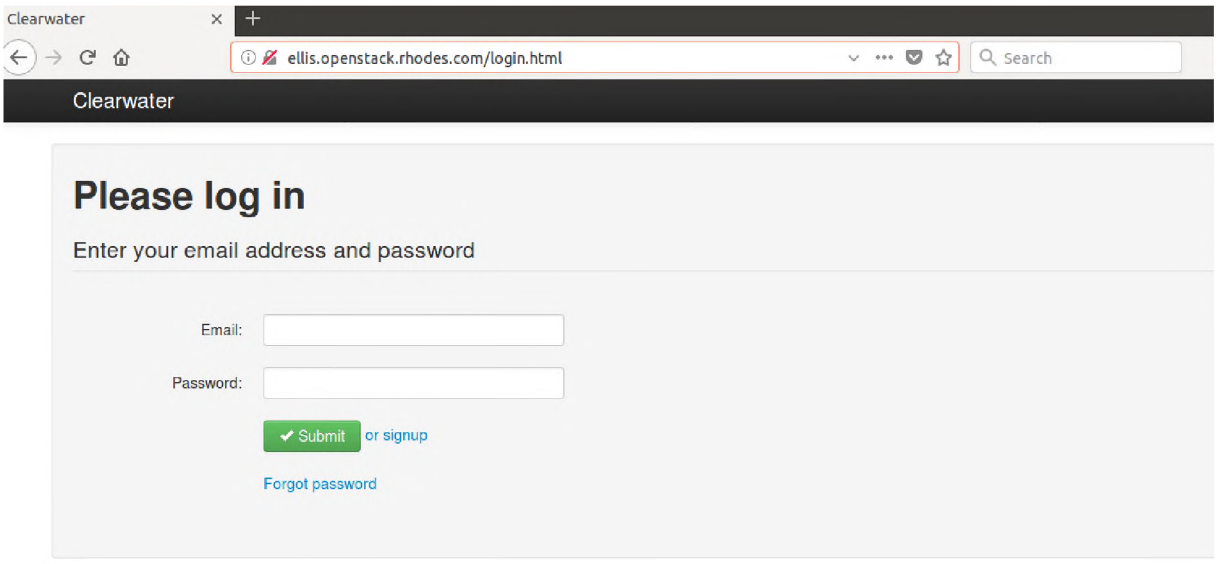

ns.openstack.rhodes.com
dime-0.openstack.rhodes.com
sprout-0.openstack.rhodes.com
clearwater-private-management
1
vellum-0.openstack.rhodes.com
clearwater-private-signaling
bono-0.openstack.rhodes.com
clearwater-private-management
homer-0.openstack.rhodes.com
clearwater-private-signaling
provider

Figure 8.2: Screenshot of Clearwater network topology deployed using Heat.



Please log in

Enter your email address and password

Email:

Password:

Figure 8.3: Screenshot of Clearwater login interface.

8.1.4 Insight from experimental deployments

These experiments were done mainly to become familiar with both implementations of the IMS network. Open IMS core and Clearwater are both reasonably simple networks with Clearwater being the more complex of the two. The different approaches available for deploying the networks present different limitations and advantages. Deploying a network service using an orchestrator is more efficient and is generally the preferred option.

8.2 IMS and Open Baton

Section 8.1 gave some insight on experiments that were done to explore the different ways in which the IMS network can be installed. Creating the network service using an orchestrator proved to be the better approach. An orchestrator simplifies the configuration process by using scripts to encapsulate the installation of the network service and does not require manual configuration. As seen above, the orchestrator chosen is Open Baton. Open Baton has a Market Place which is a public space that provides useful components such as Network Service Descriptors (NSDs), VNFs, images and drivers for both the Open IMS Core and Clearwater network services.

The Open IMS Core network was chosen for its simplicity and maturity. The Open IMS Core Network Service Record is made up of five VNF components: icscf, pcscf, scscf, bind9 and fhoss. These are the standard CSCFs, FHoSS which is the HSS implementation and a bind9 DNS server. Whereas, the Clearwater network service is made up of the six Clearwater NFs as well as the bind9 DNS server and FHoSS server NFs.

8.2.1 VNF Packages

As mentioned in Chapter 6, the VNF packages are tar files containing a Metadata.yaml file, vnfd.json and a folder containing scripts needed to configure the VNFs. The five packages for the VNF components need to be correctly configured to align with the resources in the OpenStack environment. The Metadata file is similar for all the VNF packages, the only difference being the `name` and `description` which are based on the VNF. This is because the configuration information in the Metadata file describes the VNF environment which is the same for all the VNFs in this NS.

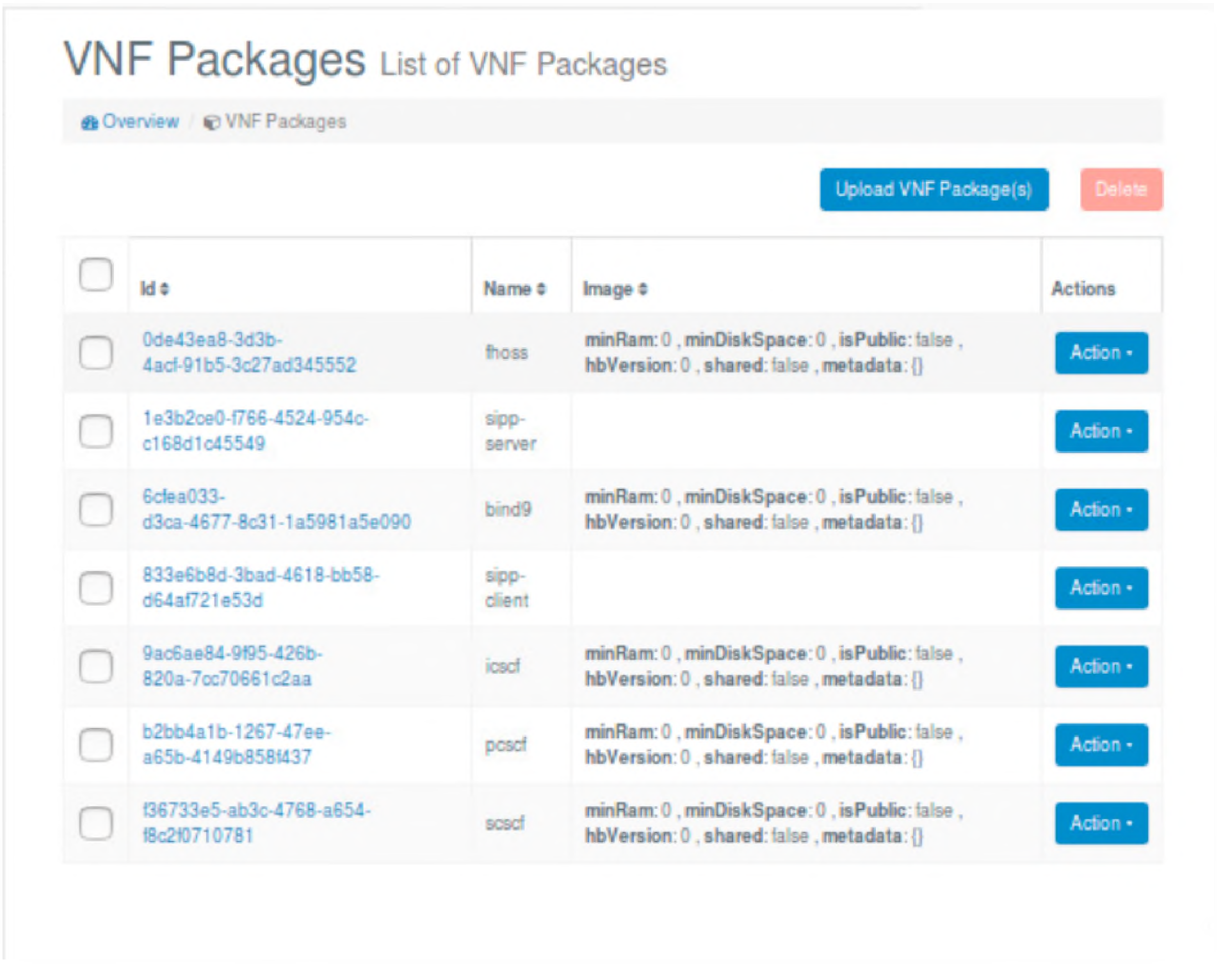
The `provider` of this VNF is Fokus and the target `nfvo_version` is 5.1.0. The `upload` option was set to "false" because an appropriate image is already available in the OpenStack VIM with the name openims. The openims image is an Ubuntu 14.04 image which is a requirement for this implementation of Open IMS Core VNFs. The `vim_types` configuration is set to openstack as this is the only VIM type implemented for this thesis.

Listing 8.1: Metadata.yaml for OpenIMSCore VNFs

```
name: VNF-name
description: "The description of the VNF"
provider: fokus
shared: true
nfvo_version: 5.1.0
image:
    upload: "false"
    names:
        - openims
image-config:
    name: openims
    diskFormat: qcow2
    containerFormat: bare
    minCPU: 0
    minDisk: 0
    minRam: 0
    isPublic: true
vim_types:
    - openstack
```

The vnfd files are unique for each VNF because these contain information specific for configuring each particular VNF. However all five VNFs are attached to the `mgmt` virtual network and the `deployment_flavour` is `m1.small` which has 1 vCPU, 2GB RAM and 10GB disk. Finally, the scripts folder in each VNF package contains the scripts needed to instantiate, configure, and start the VNF [11].

After the VNF packages were correctly configured, they were compressed into tar files that were uploaded onto the Open Baton dashboard, as shown in Figure 8.4. The list of VNF packages includes both the Open IMS Core packages as well as the SIPp packages for the SIPp NS deployed for the Zabbix monitoring experiments. Once these were successfully uploaded, VNFD IDs were automatically assigned and ready for use in the NSD. A list of the VNFD IDs is shown in Figure 8.5 which is a screenshot from the OpenBaton CLI which was used instead of the web interface as it was getting an error as shown in Figure 8.6.

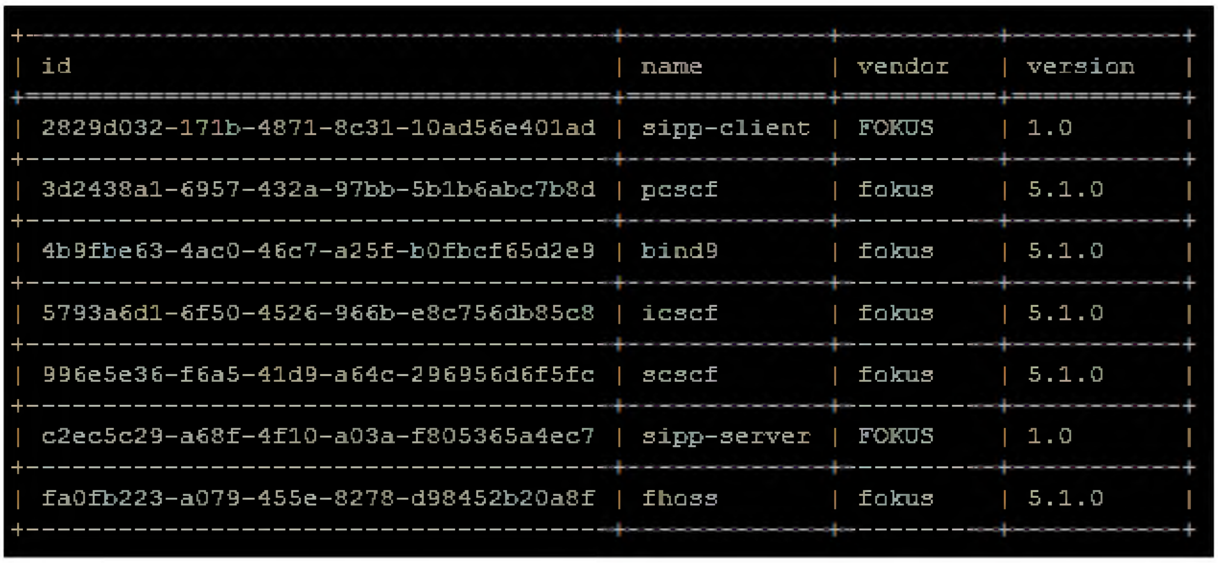


VNF Packages List of VNF Packages

Overview / VNF Packages

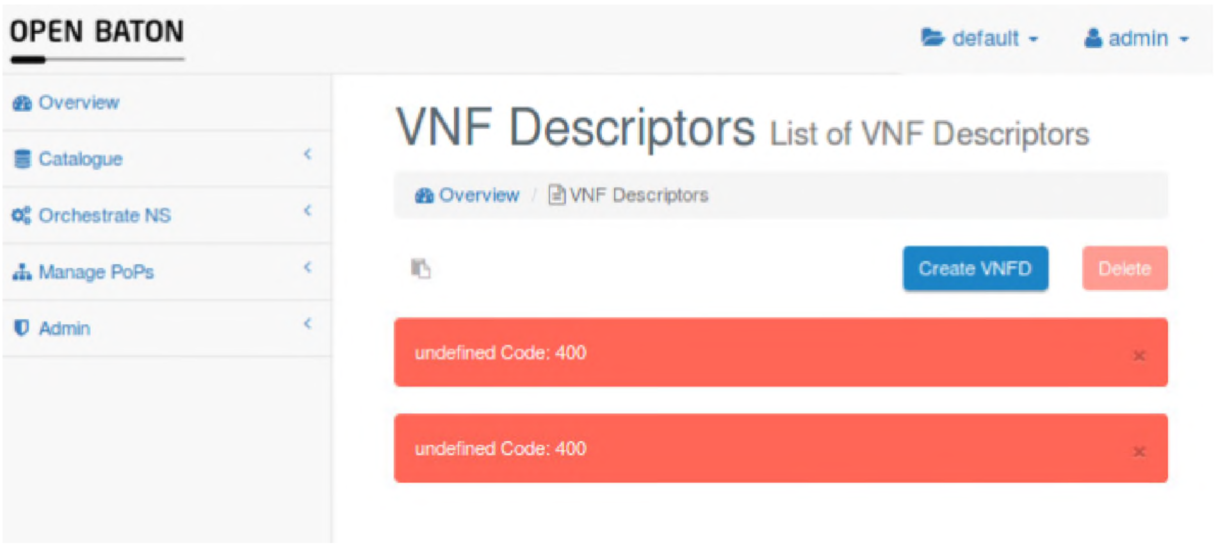
	Id	Name	Image	Actions
	0de43ea8-3d3b-4acf-91b5-3c27ad345552	fhoss	**minRam**: 0 , **minDiskSpace**: 0 , **isPublic**: false , **hbVersion**: 0 , **shared**: false , **metadata**: {}	Action
	1e3b2ce0-f766-4524-954c-c168d1c45549	sipp-server		Action
	6cfea033-d3ca-4677-8c31-1a5981a5e090	bind9	**minRam**: 0 , **minDiskSpace**: 0 , **isPublic**: false , **hbVersion**: 0 , **shared**: false , **metadata**: {}	Action
	833e6b8d-3bad-4618-bb58-d64af721e53d	sipp-client		Action
	9ac6ae84-9f95-426b-820a-7cc70661c2aa	icscf	**minRam**: 0 , **minDiskSpace**: 0 , **isPublic**: false , **hbVersion**: 0 , **shared**: false , **metadata**: {}	Action
	b2bb4a1b-1267-47ee-a65b-4149b858f437	pcscf	**minRam**: 0 , **minDiskSpace**: 0 , **isPublic**: false , **hbVersion**: 0 , **shared**: false , **metadata**: {}	Action
	f36733e5-ab3c-4768-a654-f8c2f0710781	scscf	**minRam**: 0 , **minDiskSpace**: 0 , **isPublic**: false , **hbVersion**: 0 , **shared**: false , **metadata**: {}	Action

Figure 8.4: Screenshot of the VNF Packages uploaded.



```
+--------------------------------------+-------------+----------+----------+
| id                                   | name        | vendor   | version  |
+======================================+=============+==========+==========+
| 2829d032-171b-4871-8c31-10ad56e401ad | sipp-client | FOKUS    | 1.0      |
+--------------------------------------+-------------+----------+----------+
| 3d2438a1-6957-432a-97bb-5b1b6abc7b8d | pcscf       | fokus    | 5.1.0    |
+--------------------------------------+-------------+----------+----------+
| 4b9fbe63-4ac0-46c7-a25f-b0fbcf65d2e9 | bind9       | fokus    | 5.1.0    |
+--------------------------------------+-------------+----------+----------+
| 5793a6d1-6f50-4526-966b-e8c756db85c8 | icscf       | fokus    | 5.1.0    |
+--------------------------------------+-------------+----------+----------+
| 996e5e36-f6a5-41d9-a64c-296956d6f5fc | scscf       | fokus    | 5.1.0    |
+--------------------------------------+-------------+----------+----------+
| c2ec5c29-a68f-4f10-a03a-f805365a4ec7 | sipp-server | FOKUS    | 1.0      |
+--------------------------------------+-------------+----------+----------+
| fa0fb223-a079-455e-8278-d98452b20a8f | fhoss       | fokus    | 5.1.0    |
+--------------------------------------+-------------+----------+----------+
```

Figure 8.5: List of VNFDs and their IDs



undefined Code: 400

undefined Code: 400

Figure 8.6: VNFD Error on Dashboard

8.2.2 Network Service Descriptor

The NSD template file is the main JSON file used by the NFVO to deploy a network service. For this reason, it references the VNF Descriptors for all the VNFs for that particular network service. In this case, the NSD uses the five VNFD IDs that belong to the IMS NS listed in Figure 8.5.

The NSD `name` is `OpenIMSCore Bind9 FHoSS` and the main sections that need to be defined are the `vnfd` section which has the VNFD IDs, the `vld` section which is the Virtual Link defined in the VNFD for network connectivity and finally the `vnf_dependency` section which lists all the dependencies between VNFs. There are several VNF dependencies between the VNFs because a functional IMS network service requires efficient messaging between the VNFs .

Listing 8.2: IMS NSD JSON File

```
{
    "name":"OpenIMSCore Bind9 FHoSS",
    "vendor":"fokus",
    "version":"2.0",
    "vnfd":[
        { "id":"3d2438a1-6957-432a-97bb-5b1b6abc7b8d" },
        { "id":"4b9fbe63-4ac0-46c7-a25f-b0fbcf65d2e9" },
        { "id":"5793a6d1-6f50-4526-966b-e8c756db85c8" },
        { "id":"996e5e36-f6a5-41d9-a64c-296956d6f5fc" },
        { "id":"fa0fb223-a079-455e-8278-d98452b20a8f" }
    ],
    "vld":[
        {
            "name":"mgmt"
        }
    ],
    "vnf_dependency":[
        {
            "source":{
                "name":"bind9"
            },
            "target":{
                "name":"fhoss"
            },
            "parameters":[
                "useFloatingIpsForEntries",
                "realm",
                "mgmt",
                "mgmt_floatingIp"
            ]
        },
        .
        .
        .
        {
            "source":{
                "name":"icscf"
            },
            "target":{
                "name":"scscf"
            },
            "parameters":[
                "name",
                "mgmt",
                "mgmt_floatingIp"
            ]
        }
    ]
}
```

Once the NSD is uploaded, it needs to be launched to start the deployment of the network service. The successful deployment of the network service is confirmed by the Network Service Record (NSR) getting into an ACTIVE state as shown in Figure 8.8. Figure 8.7 shows the logical architecture of the deployment while Figure 8.9 shows the network

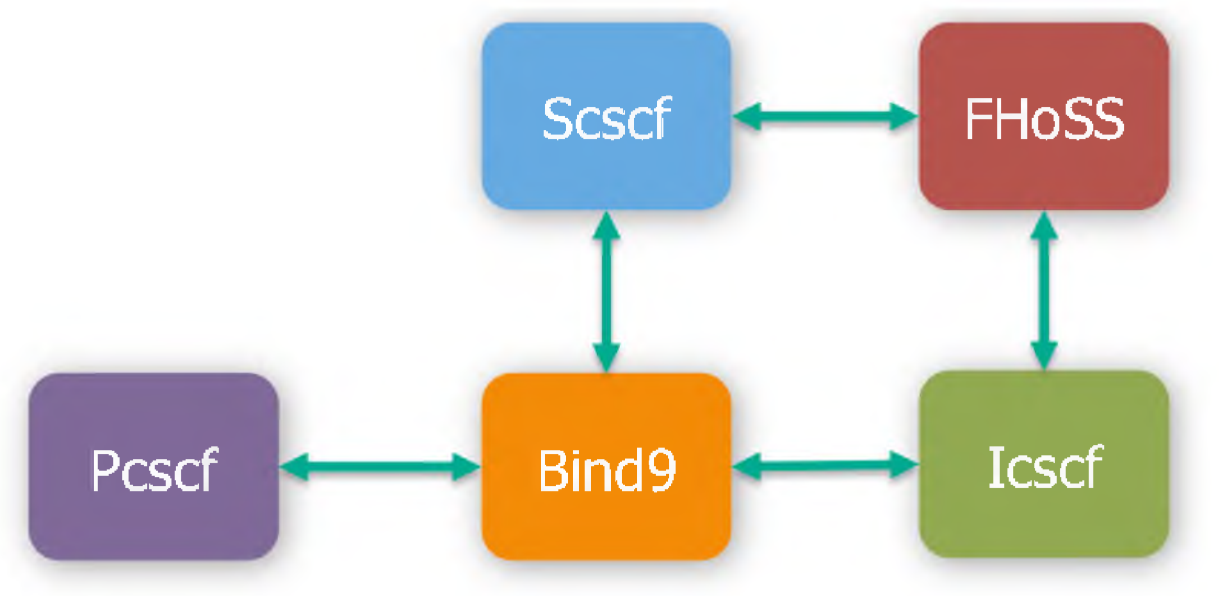
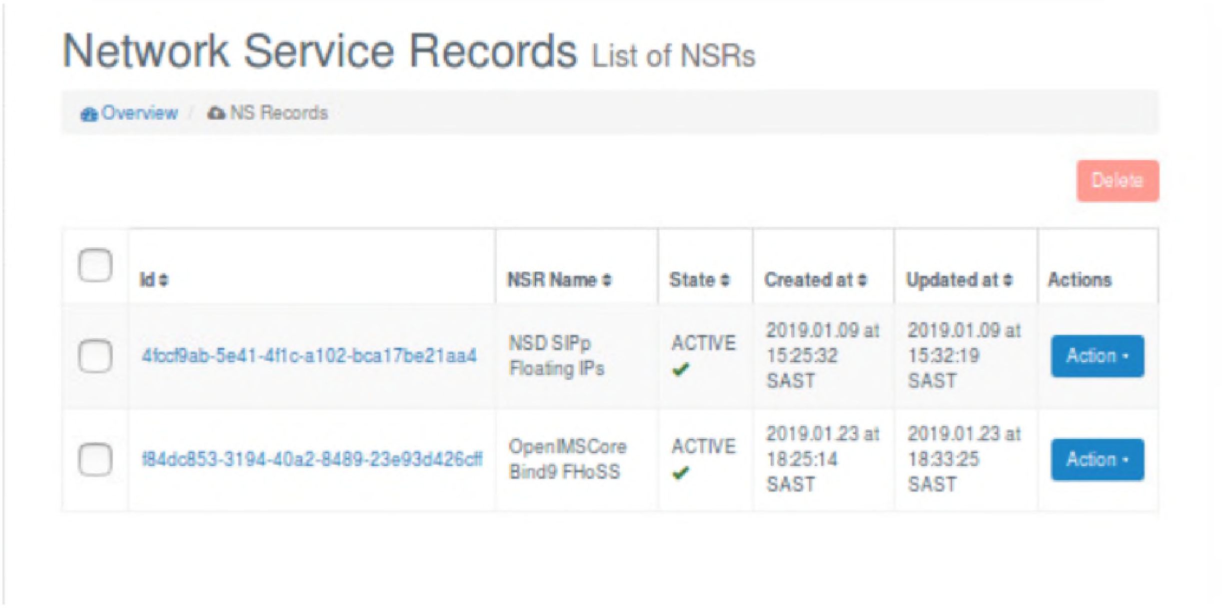


Figure 8.7: Architecture or OpenIMSCore-FHoSS-BIND Deployment. Source [11]

topology as deployed on OpenStack .



Network Service Records List of NSRs

Overview / NS Records

Delete

	Id	NSR Name	State	Created at	Updated at	Actions
	4fccf9ab-5e41-4f1c-a102-bca17be21aa4	NSD SIPp Floating IPs	ACTIVE ✓	2019.01.09 at 15:25:32 SAST	2019.01.09 at 15:32:19 SAST	Action
	f84dc853-3194-40a2-8489-23e93d426cff	OpenIMSCore Bind9 FHoSS	ACTIVE ✓	2019.01.23 at 18:25:14 SAST	2019.01.23 at 18:33:25 SAST	Action

Figure 8.8: List of the ACTIVE NSR.

8.3 Attempt to Perform Autoscaling

VNF autoscaling is a useful tool for maintaining a good QoS level. It is used to ensure that as the load experienced by the VNF increases and threatens to compromise the QoS

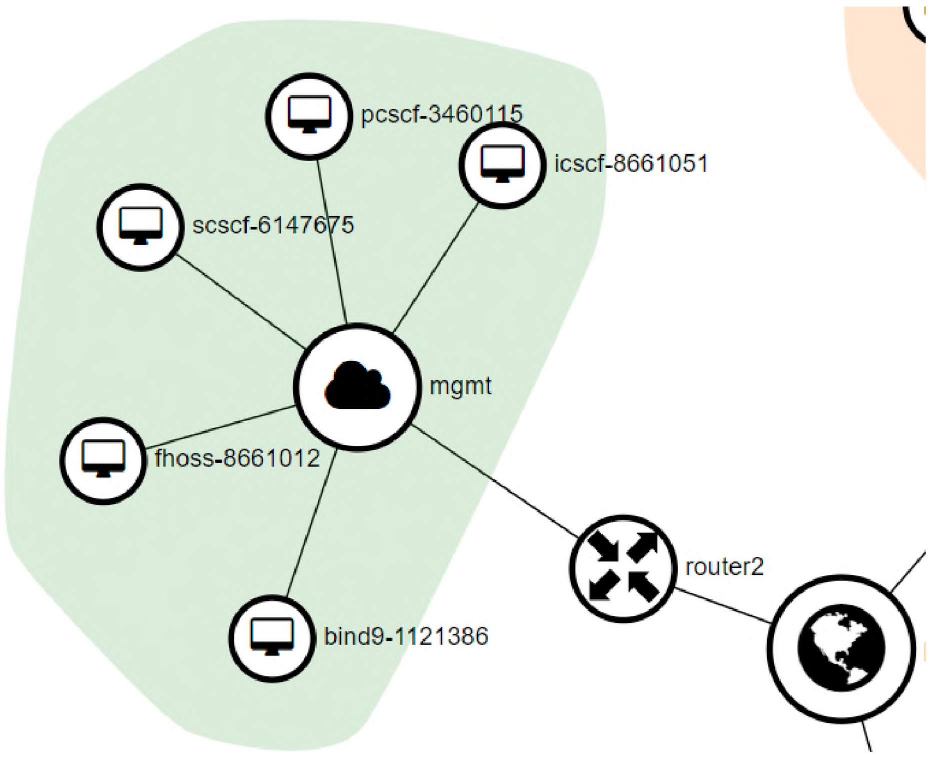


Figure 8.9: Topology of IMS deployment in OpenStack.

and user experience, the VNF will be scaled out based on a pre-defined scaling policy. The AutoScaling Engine was introduced in Chapter 6 and some concepts and policies on which it works were explained. One such concept is that the metrics that can be defined in the AutoScale Policy are based on the monitoring system used. In Chapter 7 some experiments were done on the Zabbix monitoring system using the SIPp NS. The VNF VMs were automatically registered as hosts on the Zabbix monitoring system after the NS was deployed. An *Item* and *Trigger* were added to test if the metrics were being monitored by checking if the Zabbix web interface reports when the VNF hosts are in a PROBLEM state. The results of the experimentation gave confirmation that the monitoring system was able to monitor the hosts and that the was defined correctly. The metric used was *system.cpu.util[,user]* and the threshold that was set based on observing the SIPp server as messages were received from the SIPp client. When CPU utilisation by user systems exceeded 50%, calls were no longer being handled efficiently and therefore the threshold was set at 50%.

The AutoScale Policy in Listing 8.3 was added to the P-CSCF VNFD. All the messages sent to the IMS network first go through the proxy and it is therefore the one likely to experience the most load. Hence, the P-CSCF was chosen as the first target for autoscaling. The alarm set in the Autoscaling Policy uses the same metric as the one experimented with on the Zabbix monitor.

Listing 8.3: Example of an Autoscaling Policy

```
"auto_scale_policy": [
                {
                    "name":"scale-out",
                    "threshold":100,
                    "comparisonOperator":">=",
                    "period":30,
                    "cooldown":60,
                    "mode":"REACTIVE",
                    "type":"WEIGHTED",
                    "alarms": [
                        {
                            "metric":"system.cpu.util[,user]",
                            "statistic":"avg",
                            "comparisonOperator":">",
                            "threshold":50,
                            "weight":1
                        }
                    ],
                    "actions": [
                        {
                            "type":"SCALE_OUT",
                            "value":"1"
                        }
                    ]
                }]
```

In order to test if the AutoScaling Engine was operating as expected, the logs were checked for the AutoScaling Engine and they indicated that there was an exception because the Autoscaling Engine was not receiving the data in the format it expected, giving the error below:

Listing 8.4: Error message from ASE logs

```
2019-01-23 18:49:00.398  INFO 3969 --- [ThreadPoolTaskScheduler-10]
o.o.a.core.detection.DetectionEngine       :
Get monitoring plugin with following parameters: 203.0.113.183
2019-01-23 18:49:00.399 ERROR 3969 --- [ThreadPoolTaskScheduler-10]
o.o.a.core.detection.task.DetectionTask    :
Exception occured while getting measurements ->
Expected a com.google.gson.JsonArray but was com.google.gson.JsonObject,
Trying again next time...
```

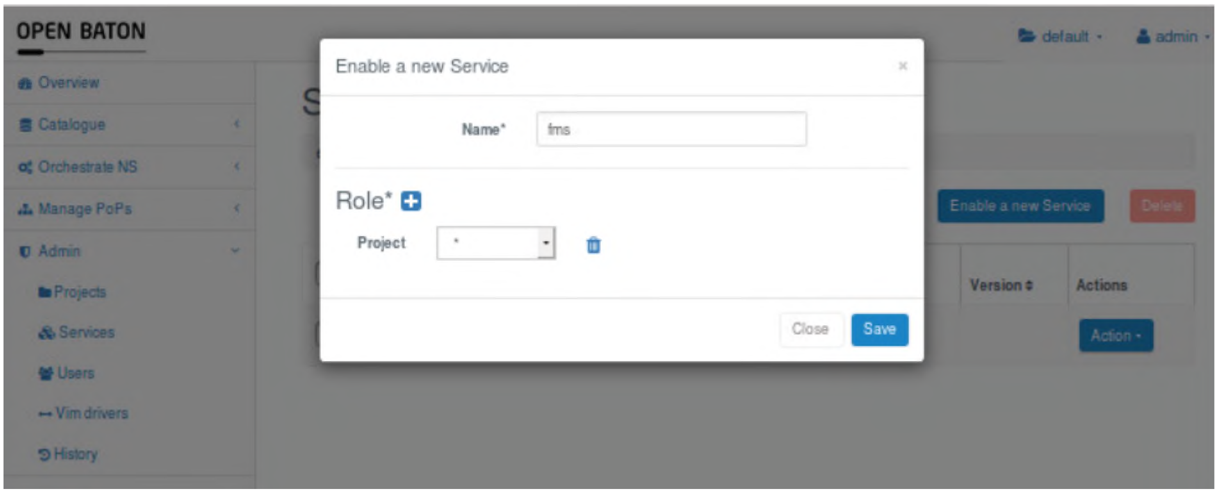
8.4 Attempt to Perform Fault Management

As with autoscaling, fault management is important for maintaining QoS. Autoscaling is a proactive approach to maintaining QoS because it anticipates behaviour that might result in the need for more resources before the VNF starts misbehaving. Fault management is the reactive approach that defines the actions that needs to be taken when the VNF malfunctions and something goes wrong and within Open Baton this can be fixed with a HEAL or SWITCH TO STANDBY action.

However, installing the service proved not to be a simple process. The FM system can be installed via a Debian package or from source code. Both options as per the documentation were unsuccessful. When a service is installed, it needs to be enabled by downloading a service key via the dashboard as shown in Figure 8.10. This way of adding a service via the dashboard fails as shown in Figure 8.11. If the service is installed using debian packages, the service key needs to be directly added into the configuration file. The source code approach worked for the AutoScaling Engine but did not work for the Fault Management System. As a result, no further experimentation could be done with fault management on the network.

8.5 Summary

The final phase of the implementation was discussed in this chapter. With the rest of the infrastructure in place, the IMS network service could now be deployed on the cloud. Some experiments were done to explore the different ways in which the IMS network can be deployed and to see which of those proves to be the most efficient. Using an



Enable a new Service

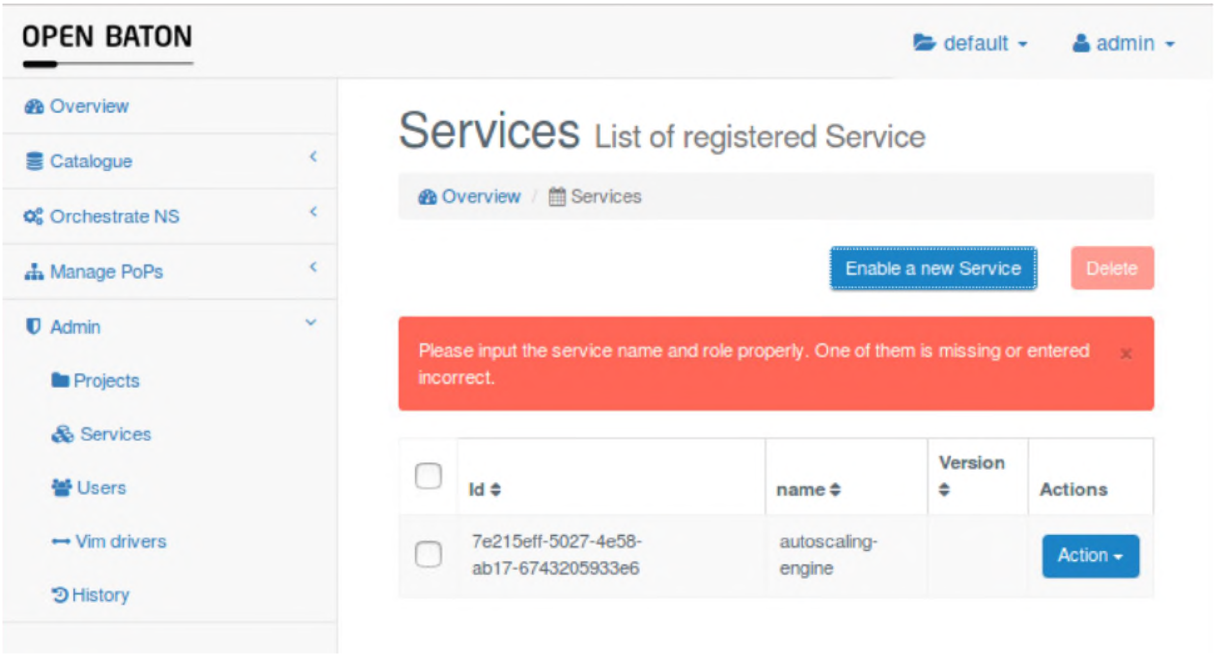
Name* fms

Role*

Project *

Close Save

Figure 8.10: Enabling the FMS service



Please input the service name and role properly. One of them is missing or entered incorrect.

	Id	name	Version	Actions
☐	7e215eff-5027-4e58-ab17-6743205933e6	autoscaling-engine		Action

Figure 8.11: Failure to Create the FMS service

orchestrator proved to be the most efficient and the final deployment was made using the Open Baton orchestrator. Attempts to perform autoscaling and fault management on the network services were unsuccessful. The insight from all the experiments done will be presented in the next chapter.

Chapter 9

Discussion

The aim of this research was to investigate the readiness of open source tools to build a telco cloud platform. The telecommunication industry is constantly evolving and is a field with great innovation potential. Open source tools encourage innovation and various open source tools are currently available that can be used to build a telco cloud environment. This research looks into some of these tools and investigates the maturity and how easy to use these tools are. This investigation is done from the perspective of an administrator who has experience in building systems using different tools while following the provided documentation. The aim of this work was not to study the source code and fix components that were not working as expected or poorly documented. The following sections discuss the readiness of the tools used in attempting to build the telco cloud.

9.1 VMware

VMware was the first software that was installed onto the server. Being a bare metal hypervisor, it was installed directly onto the server. Once VMware is installed and running, it is easy to use and everything is done via the web interface. It provides a simple environment in which the physical host, networking, storage and VMs can be managed by using the navigator, shown in Figure 9.1, to access the different components. Information about the physical server can be seen by going to the *Host* option. There all the information about the hardware, vmware configurations, system information and Performance graphs of the CPU and memory is made available as shown in Figure 9.2. Such information is often useful for system administrators who are managing the system and need such information readily and easily available.

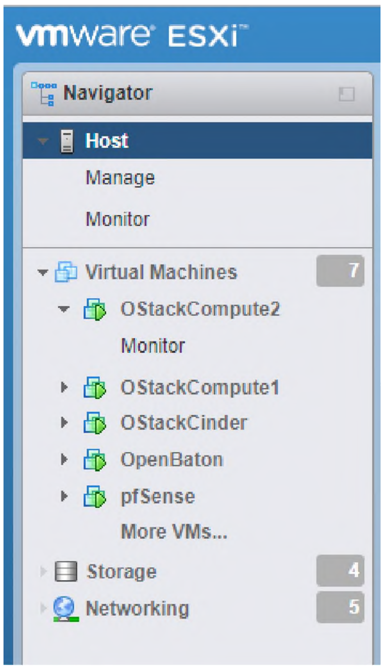
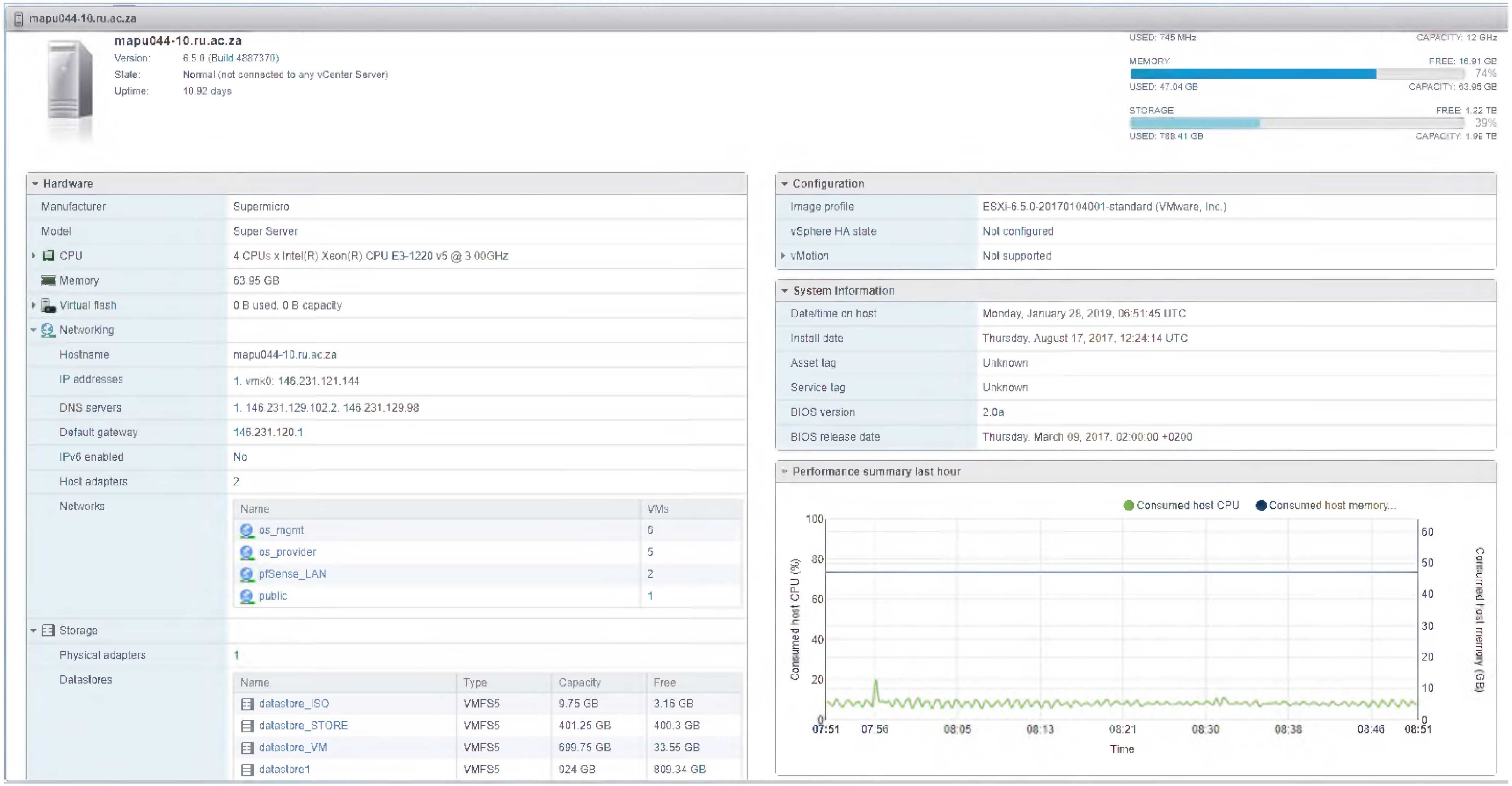


Figure 9.1: The VMware Navigator for Managing Resources.

Figure 9.3 shows the network settings that can be managed from the *Networking* option in the navigator and in this case shows the *Port groups* (subnets) that the VMs can be attached to. The *Storage* option is similarly simple and from it the datastores, adapters and storage devices can be managed.

The main reason for setting up the VMware hypervisor is to be able to create VMs that can provide infrastructure for the cloud. As such, the VMware environment has to be stable and reliable as the VMs created host the software that runs the cloud and if there are problems at the infrastructure, the cloud will also be unstable. The process of creating a VM is made up of 5 steps as shown in Figure 9.4. It's a very simple process that involves choosing the the operating system, selecting a datastore in which the configuration and disk files will be stored, customising virtual hardware settings and the last step is simply reviewing the configurations before finalising the deployment.

VMware proved to be simple and stable software to use and all this can be done without needing to look deeply into how the hypervisor works. Proficiency in configuring a networking is the main skill required.


mapu044-10.ru.ac.za

mapu044-10.ru.ac.za

Version:	6.5.0 (Build 4887370)
State:	Normal (not connected to any vCenter Server)
Uptime:	10.92 days

USED: 745 MHz CAPACITY: 12 GHz

MEMORY FREE: 16.91 GB 74% USED: 47.04 GB CAPACITY: 63.95 GB

STORAGE FREE: 1.22 TB 39% USED: 788.41 GB CAPACITY: 1.98 TB

Hardware

Manufacturer	Supermicro
Model	Super Server
CPU	4 CPUs x Intel(R) Xeon(R) CPU E3-1220 v5 @ 3.00GHz
Memory	63.95 GB
Virtual flash	0 B used, 0 B capacity
Networking	
Hostname	mapu044-10.ru.ac.za
IP addresses	1. vmk0: 146.231.121.144
DNS servers	1. 146.231.129.102 2. 146.231.129.98
Default gateway	146.231.120.1
IPv6 enabled	No
Host adapters	2
Networks	

Name	VMs
os_mgmt	6
os_provider	5
pfSense_LAN	2
public	1

Storage	
Physical adapters	1
Datastores	

Name	Type	Capacity	Free
datastore_ISO	VMFS5	9.75 GB	3.16 GB
datastore_STORE	VMFS5	401.25 GB	400.3 GB
datastore_VM	VMFS5	699.75 GB	33.55 GB
datastore1	VMFS5	924 GB	809.34 GB

Configuration

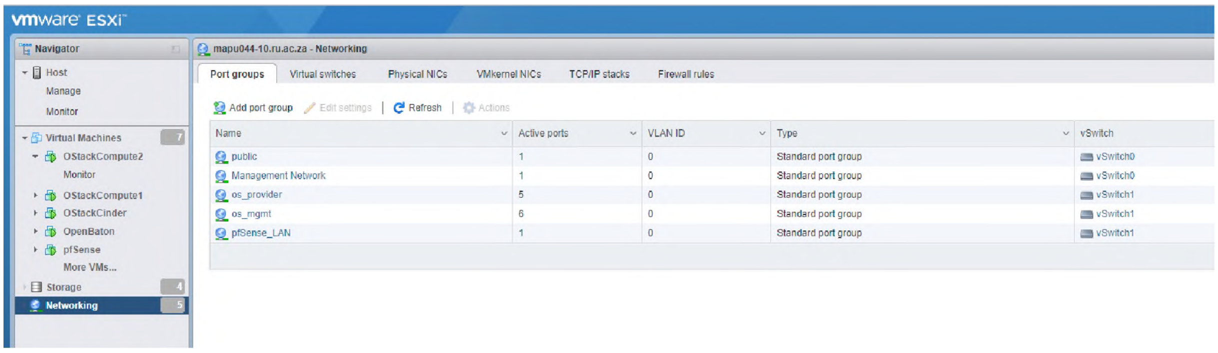
Image profile	ESXi-6.5.0-20170104001-standard (VMware, Inc.)
vSphere HA state	Not configured
vMotion	Not supported

System Information

Date/time on host	Monday, January 28, 2019, 08:51:45 UTC
Install date	Thursday, August 17, 2017, 12:24:14 UTC
Asset tag	Unknown
Service tag	Unknown
BIOS version	2.0a
BIOS release date	Thursday, March 09, 2017, 02:00:00 +0200

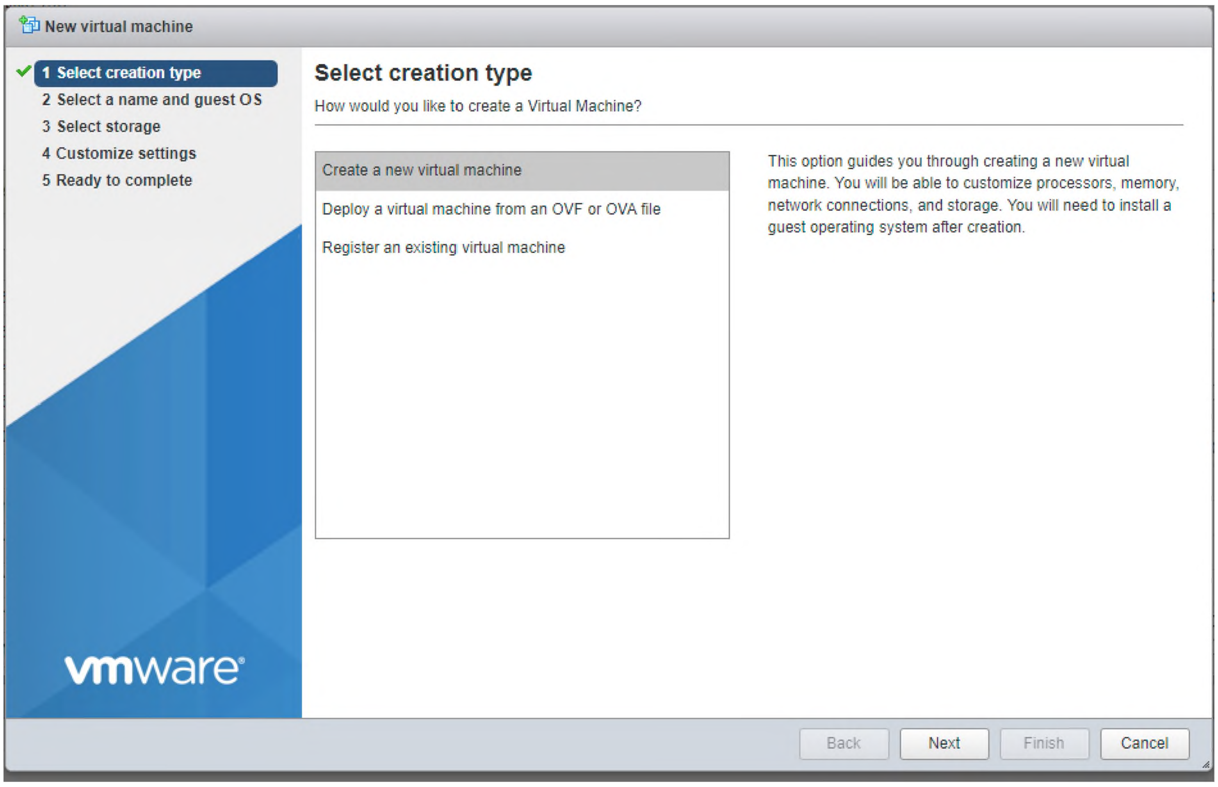
Performance summary last hour

Figure 9.2: Host information about the Physical Server.



Name	Active ports	VLAN ID	Type	vSwitch
public	1	0	Standard port group	vSwitch0
Management Network	1	0	Standard port group	vSwitch0
os_provider	5	0	Standard port group	vSwitch1
os_mgmt	6	0	Standard port group	vSwitch1
pfSense_LAN	1	0	Standard port group	vSwitch1

Figure 9.3: Network Settings in VMware.



Select creation type

How would you like to create a Virtual Machine?

Create a new virtual machine

Deploy a virtual machine from an OVF or OVA file

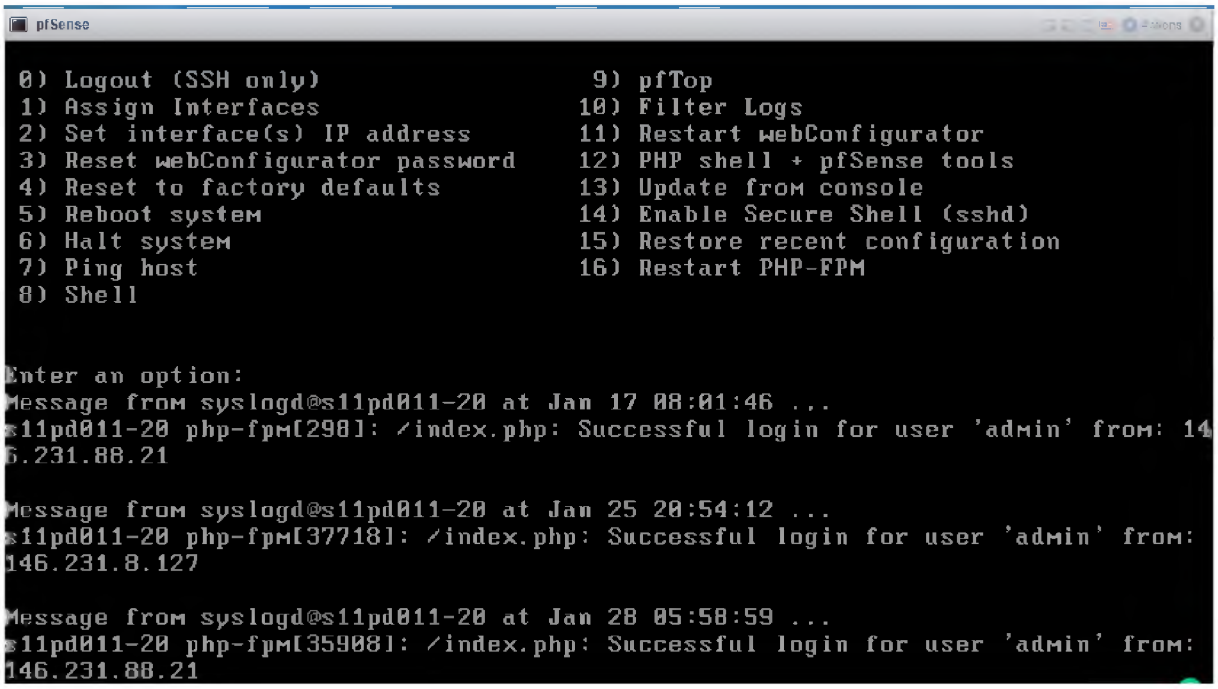
Register an existing virtual machine

This option guides you through creating a new virtual machine. You will be able to customize processors, memory, network connections, and storage. You will need to install a guest operating system after creation.

Figure 9.4: Creating a Virtual Machine in VMware.

9.2 pfSense

pfSense software was installed on a VM inside VMware. Once installed, several configuration settings could be done on the console interface inside VMware, shown in Figure 9.5. A public IP address was set, through which the web interface could be accessed. Various network settings for use by the OpenStack cloud infrastructure and the nodes on the OpenStack cloud were made using via the web interface. The dashboard for the pfSense router is shown in Figure 9.6. It gives an overview of the system information and the network interfaces currently configured on the router. Additional settings can be made by navigating through the different configuration options. For example, several firewall rules were set as shown in Chapter 5. It was easy to install, configure and get started with without needing to understand any of the source code.

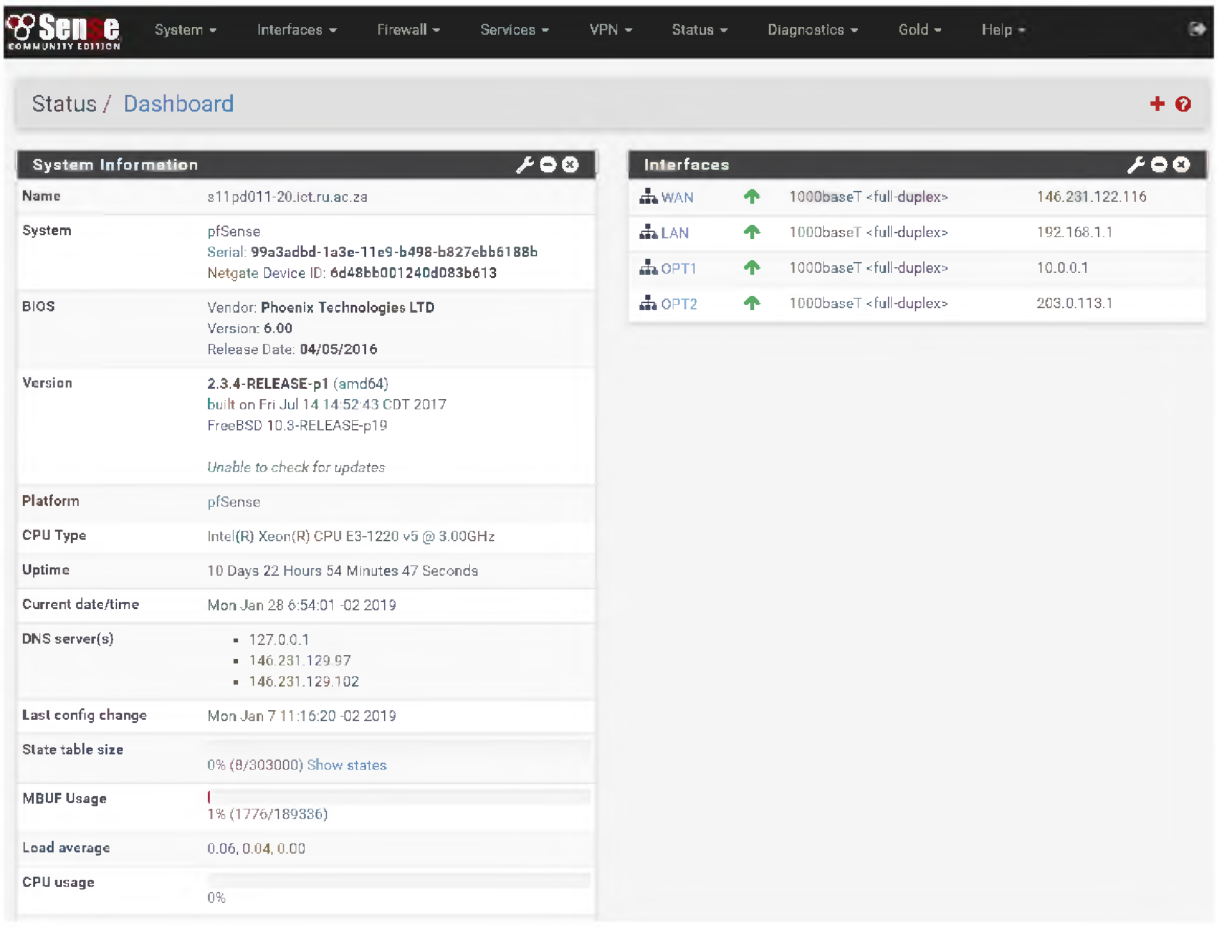


```
 0) Logout (SSH only)                    9) pfTop
 1) Assign Interfaces                   10) Filter Logs
 2) Set interface(s) IP address         11) Restart webConfigurator
 3) Reset webConfigurator password      12) PHP shell + pfSense tools
 4) Reset to factory defaults           13) Update from console
 5) Reboot system                       14) Enable Secure Shell (sshd)
 6) Halt system                         15) Restore recent configuration
 7) Ping host                           16) Restart PHP-FPM
 8) Shell


Enter an option:
Message from syslogd@s11pd011-20 at Jan 17 08:01:46 ...
s11pd011-20 php-fpm[298]: /index.php: Successful login for user 'admin' from: 14
6.231.88.21

Message from syslogd@s11pd011-20 at Jan 25 20:54:12 ...
s11pd011-20 php-fpm[37718]: /index.php: Successful login for user 'admin' from:
146.231.8.127

Message from syslogd@s11pd011-20 at Jan 28 05:58:59 ...
s11pd011-20 php-fpm[35908]: /index.php: Successful login for user 'admin' from:
146.231.88.21
```

Figure 9.5: pfSense CLI accessed via the VMware Console.

9.3 OpenStack

OpenStack is a widely used open source cloud operating system and is therefore fast developing. When this research commenced, OpenStack Ocata was the latest version as of February 2017 and when it concluded, OpenStack Rocky was the latest version as of



Status / Dashboard

System Information

Name	s11pd011-20.ict.ru.ac.za
System	pfSense Serial: 99a3adbd-1a3e-11e9-b498-b827ebb6188b Netgate Device ID: 6d48bb001240d083b613
BIOS	Vendor: Phoenix Technologies LTD Version: 6.00 Release Date: 04/05/2016
Version	2.3.4-RELEASE-p1 (amd64) built on Fri Jul 14 14:52:43 CDT 2017 FreeBSD 10.3-RELEASE-p19 *Unable to check for updates*
Platform	pfSense
CPU Type	Intel(R) Xeon(R) CPU E3-1220 v5 @ 3.00GHz
Uptime	10 Days 22 Hours 54 Minutes 47 Seconds
Current date/time	Mon Jan 28 6:54:01 -02 2019
DNS server(s)	• 127.0.0.1 • 146.231.129.97 • 146.231.129.102
Last config change	Mon Jan 7 11:16:20 -02 2019
State table size	0% (8/303000) Show states
MBUF Usage	1% (1776/189336)
Load average	0.06, 0.04, 0.00
CPU usage	0%

Interfaces

WAN	↑	1000baseT <full-duplex>	146.231.122.116
LAN	↑	1000baseT <full-duplex>	192.168.1.1
OPT1	↑	1000baseT <full-duplex>	10.0.0.1
OPT2	↑	1000baseT <full-duplex>	203.0.113.1

Figure 9.6: pfSense Web Interface.

the end of August 2018. Even with its fast growth, an OpenStack cloud is fairly simple to build due to its rigorous documentation which tries to minimise ambiguity.

OpenStack is a complex system and the installation requires several different elements to be installed and configured. Chapter 4 discussed OpenStack in greater detail and gave an idea of the different elements that make up an OpenStack installation. At the infrastructure level, OpenStack requires several nodes that perform different tasks in making the cloud functional. VMs were created inside VMware to be the controller, compute and block storage nodes that OpenStack requires. The features provided by the OpenStack cloud are available through services installed on these nodes. This means that there is a lot of dependency between the nodes, as parts of the services can be configured on different nodes. Additionally, some services rely on the other services running correctly. Despite OpenStack being a complex operating system, the installation process was fairly simple due to the documentation being thorough and clear. As mentioned earlier, OpenStack is constantly going through development and there is a lot of innovation going towards the operating system. This is fueled by it having a fast growing open source community as it is a popular choice for deploying private and public clouds. Currently it has over 96368 community members, 666 supporting companies, is deployed in 187 countries and has over 20 million lines of code [9]. As a result, new versions of OpenStack are frequently being released. This could easily start causing problems in terms of compatibility between services, also, other software that has been developed to work with OpenStack might require older versions of OpenStack. This problem is eliminated because the full installations are documented based on the version and the documentation covers how to install and configure all the services for that particular version.

9.4 Orchestrating IMS

There are several orchestrators available that already have templates available that can be used to deploy either the Project Clearwater or Open IMS Core network on an OpenStack cloud.

9.4.1 Heat

Heat was the first orchestrator experimented with. It is the native orchestrator for OpenStack that can installed as an additional service for the cloud. Heat, like most orchestrators, uses templates (HOT - Heat Orchestration Templates) to create networks and other virtual appliances. It is easy to use and some examples of how to create templates are provided in the documentation making it easier to create new templates.

Some HOT templates for launching the deployment of the Project Clearwater IMS network were downloaded from [75] and experimented with. The templates were easy to use and all that had to be done was to make sure that the parameters in the main template correspond to the names of resources in the OpenStack cloud. The network in this case was more complex as it involved multiple instances and each instance belongs to both the signaling and management network which was a way to separate service traffic from management traffic. Despite the network being more complex in topology as well as having to install software for each network function, Heat is able to get the network deployed in a short time.

Heat proved to be an easy to install, easy to use, reliable orchestrator for launching new instances, networks and even able to deploy network services using templates. However, this research wanted to create a telco cloud and therefore needs to follow NFV standards. The NFV MANO specification developed by ETSI is a framework that is being used to standardise how NFV is used in telecommunication. The specification gives a structure that the management and orchestration of VNFs and NSs must follow. Even though Heat is able to deploy the network service, it is however not compliant with the ETSI NFV MANO specification.

9.4.2 Cloudify

Cloudify, which was introduced in Chapter 2, mainly makes use of a Cloudify Manager to run the management service. The management service is responsible for performing the management and orchestration tasks such as automating the deployment, configuration and scaling of the NSs.

The challenge however was understanding the documentation and how it integrates with OpenStack to manage the resources inside OpenStack. The documentation in [76] provides templates and instruction for deploying Clearwater IMS using Cloudify. It says that the Cloudify manager could run as an instance inside OpenStack using a cloudify-manager image. This however did not make sense architecturally as the Cloudify manager is supposed to orchestrate instances inside the OpenStack cloud when it is an instance in OpenStack itself. This does not align with the ETSI MANO specification. The architecture of how Cloudify works was not very clear based on the documentation and attempting the installation did not prove very successful.

9.4.3 Open Baton

Open Baton was the preferred option and it is an ETSI compliant orchestrator that also offers a generic VNFM, therefore completing the three functions required in the NFV MANO architecture. Open Baton has a Market Place on which templates for deploying the Open IMS Core and Project Clearwater networks are made available. The Open Baton NFVO and generic VNFM were installed without much difficulty. Open Baton has a modular approach to make the platform easy to extend by adding services. Currently available is the autoscaling and fault management services. Both provide very important functions to a telco cloud environment and enable the system to provide quality services and minimise downtime.Both these services require that the VNFs be monitored in order for the system to know whether everything is running at an acceptable level. Open Baton has a Zabbix plugin to interoperate with the Zabbix monitoring system. The plugin was installed as per the documentation and configured to work with the NFVO.

To test if the orchestrator was working, some simpler network services were deployed namely the Iperf and SIPp NSs. These were quick and easy to launch and run on the OpenStack cloud. All this was done via the Open Baton web interface which gives a useful view of the elements available as shown previously in Figure 6.2. The sequence of events of how the VNFs and the resulting NS are created is provided in the documentation making it easier to understand the workflow between the components. Open Baton defines states that a VNFR can be in and these correspond to the ETSI NFV states as defined in [68]. The Open Baton NFVO and generic VNFM proved to be mature and able to orchestrate the deployment of network services as well as able to tear down the network if the command is given to delete the NSR.

9.5 Zabbix

Zabbix is the monitoring system that Open Baton currently supports, therefore, it was the default monitoring system chosen for the deployment. Zabbix uses a client-server model and therefore required that a Zabbix server be installed. The documentation states that the server can run on the same node as Open Baton and for simplicity, the Zabbix server was installed on the same node as Open Baton. Open Baton provides instructions for installing and configuring the Zabbix server [71]. When VNFs are created, a Zabbix agent is automatically installed on the VNF and this Zabbix agent makes up the client with which the Zabbix server interacts with. Open Baton offers a helper script that is run on the Zabbix server to automatically register all VNFs deployed using Open Baton.

Experiments done in Chapter 7 showed that Zabbix is able to monitor the VNFs successfully using metrics which can be set by the administrator. Zabbix also provides a web interface that gives the system administrator a visual overview of the system as well as graphical representations of the performance of the VNF instances. The zabbix monitor is able to send notifications when a threshold has been crossed for one of the metrics and get into a PROBLEM state but once the system is working as expected, the state is changed back to OK. The Zabbix web interface and documentation [69] made it very easy to use the monitoring system and therefore proved to be a mature and reliable network monitor.

9.6 Autoscaling Engine and Fault Management System

One of the advantage of using a cloud environment and particularly a telco cloud, is that it supports services such as autoscaling and fault management. Open Baton allows users to extend their platform by adding such services via the Open Baton SDK. There are some services already available on Open Baton, including an Autoscaling Engine and a Fault Management System.

According to the Open Baton documentation [11], services can be enabled by the administrator via the dashboard. Once a service has been saved on the dashboard, a service token should be generated, however this gave an error message. There is often more than one way of installing the services and so, the Autoscaling Engine was successfully installed using the source code installation. Based on the Open Baton documentation and experiments done using Zabbix, the Autoscaling Policies that go in the VNFD files were fairly simple to compile. Attempts to monitor the VNFs were unsuccessful due the engine not receiving the correct data format it was expecting. Not much is said in the documentation to explain the format in which the data should be, furthermore it was not simple to debug how the data is sent from the Zabbix server to the Zabbix plugin and finally to the Autoscaling engine.

The Fault Management System could also be installed in multiple ways, including a source code installation. The source code installation worked for the Autoscaling Engine but was unsuccessful for the Fault Management System. The expectation would have been that the two services can be installed in the same way for consistency. Failure to install the Fault Management System meant that no experiments could be done to test the service.

Despite not being able to perform both the autoscaling and fault management functions, it was evident from the documentation that there are two ways of getting notifications from the Zabbix server. One way uses a createThreshold method inside the source code and a script file that sends information received from the Zabbix server to the Zabbix plugin. The second way uses a MonitoringPluginCaller configured in the source code files. The documentation does say that the Fault Management System uses the createThreshold method, the same was initially assumed for the Autoscaling Engine for consistency. However, a scan of the source code showed that it uses the MonitoringPluginCaller.

The work done trying to use these services as a system administrator shows that this level of the implementation is still very immature. More work is yet to be done on the documentation and implementation of the services.

9.7 Summary

This chapter reviewed and discussed the various open source software used in this work to build an experimental telco cloud. The software used in building the basic cloud platform was VMware, pfSense and OpenStack. The aim was to get insight on the readiness of the open source software to build a telco cloud from the perspective of a system administrator. All three software tools were easy to use and well documented making the task of building the cloud doable without much difficulty. The ETSI compliant orchestrator added to the system is the Open Baton NFVO as well as the Generic VNFM. In order to be able to monitor VNFs, a network monitor is needed and since Open Baton already has a plugin for the Zabbix monitor, Zabbix was the natural choice of monitor. The Open Baton NFVO, Generic VNFM and Zabbix monitor were all installed on the same server and with ease, also proving to be mature software. When the cloud and orchestration was in place, features such as autoscaling and fault management were to be added. These are the services that add resilience to the platform. Autoscaling and fault management were the services chosen for this deployment as they are already available as part the Open Baton services. Unfortunately, the software for these services proved to be immature and not very well documented.

Chapter 10

Conclusion

This research sought to investigate the readiness of open source tools to build a cloud environment that is tailored for telecommunication services. This was achieved through studying literature and some of the readilily available tools and using the insight aquired to experiment with implementing such a cloud.

This chapter summarises the work presented in this thesis focusing mainly on the four phases under which the telco cloud implementation was done as discussed in chapters 5, 6, 7 and 8. An analysis of the work done will be used to draw a conclusion on the main findings of the research and lastly, some suggestions will be given for future work.

10.1 Achieved Objectives

The primary outcome of this research was to give insight into the readiness of open source tools to build a telco cloud. In this section, the objectives outlined in section 1.5 are revisited by analysing the different phases of the implementation.

10.1.1 Building the Infrastructure As A Service

The infrastructure for the deployment was made up of a single physical server and as such, the starting point was installing a hypervisor in the form of VMware. Several nodes were created inside VMware to host the software that was to run the cloud. One of these nodes ran pfSense software and performed the functions of a router and firewall system.

The other nodes ran OpenStack software as OpenStack is the cloud operating system that was chosen as the IaaS solution for the platform.

VMware, pfSense and OpenStack are all mature, open source software. From the view of a system administrator installing and configuring a cloud platform using the available documentation as a guide, it can be concluded that there are mature, well documented open source software for building IaaS.

10.1.2 Management and Orchestration with Open Baton

The second phase involved adding the next two components required to complete an ETSI NFV MANO compliant system. Open Baton proved to be a good choice that offers both the orchestrator and manager functions. The orchestrator and manager were both installed easily on the same node, following the Open Baton documentation and experiments done with deploying basic network services proved that the orchestrator is a able to successfully launch the network services. The NFVO and VNFM functions of Open Baton worked without difficulty and can therefore be regarded as mature implementations of those components.

10.1.3 Monitoring

The third phase was adding the system to monitor the VNF instances. Since Open Baton already integrates with the Zabbix monitoring system, it became the default choice of network monitor. A Zabbix server was installed and made accessible via a web interface where various configurations could be made following the Open Baton and Zabbix documentation. Zabbix also proved to be a mature monitor that is well documented, easy to install and easy to use.

10.1.4 IMS Virtualisation, Autoscaling and Fault Management

There are several ways available of installing the IMS network but using templates via an orchestrastor proved to be the most efficient. The Open IMS core network was deployed using templates via Open Baton. However, attempts to perform autoscaling and fault management on the network services were unsuccessful. The autoscaling engine was installed but was unable to receive correct monitoring data from the Zabbix plugin on

Open Baton. Some debugging was done to try and pin point where the problem was. This proved to not be an easy task as the autoscaling engine is not very well documented. There is no clear documentation detailing how the autoscaling engine receives data and what format the data is expected in. The implementation was done from the perspective of a system administrator and therefore the source code was not debugged as thoroughly as a developer would.

Installing the fault management system was unsuccessful too. There were different ways to install the fault management system but none was successful.

From this, it can be concluded that the top-level services that complete the requirements for a telco cloud such as autoscaling and fault management are not yet mature. There is on going work to develop and improve these services as the demand for the increases, however they are yet to be available as mature and well documented services.

10.2 Future Work

The nature of this work could be described as taking a snapshot of a moving target. Looking a year or two ahead, it is almost certain that an investigation into the maturity of open source software for building a telco cloud would produce a different conclusion. For example, as more work and research is done for autoscaling, fault management and other such services, more mature and well documented implementations will become available and their integration and use should be studied.

This work was guided by the ETSI NFV MANO framework and looked specifically at Open Baton. There are other open source ETSI compliant orchestrators available that can be used in place of Open Baton to carry out a similar investigation.

The experimental system presented in this work is fully capable of hosting other network services besides the ones discussed in this work. It would be interesting to deploy and study such services.

10.3 Summary

This thesis has detailed the process undertaken to investigate the readiness of open source software to build a telco cloud on which VNFs can be implemented. The investigation focused on a subset of the existing open source software. The software was used to carry out an experiment to test the maturity of the software tools by using the to build a telco cloud, assuming the role of a system administrator. The thesis concludes that open source tools for building and orchestrating the cloud are generally mature and well documented but the top level services, such as autoscaling and fault management, are yet to reach maturity and be easily deployable.

References

[1] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. DeTurck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[2] G. Camarillo and M.-A. Garca-Martn, *The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds*, 3rd ed. John Wiley & Sons, 2011, 2011.

[3] L. J. Vora, "Evolution of mobile generation technology: 1g to 5g and review of upcoming wireless technology 5g," *International Journal of Modern Trends in Engineering and Research*, vol. 2, no. 10, pp. 281–290, 2015.

[4] J. Yu, "From 3g to 4g: technology evolution and path dynamics in china's mobile telecommunication sector," *Technology Analysis & Strategic Management*, vol. 23, no. 10, pp. 1079–1093, 2011.

[5] D. Bonderud, "Internet speed test: 3g, 4g, lte, and wifi who wins?" https://www.bandwidthplace.com/internet-speed-test-3g-4g-lte-and-wifi-who-wins-article/, Feb 2014, (Accessed on 01/31/2019).

[6] M. I. Baba, N. Nafees, I. Manzoor, K. A. Naik, and S. Ahmed, "Evolution of mobile wireless communication systems from 1g to 5g : A comparative analysis," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, vol. 4, no. 1, pp. 1–8, 2018.

[7] M. Fizza and M. A. Shah, "5g technology: An overview of applications, prospects, challenges and beyond," in *Proceedings of the IOARP International Conference on Communication and Networks (ICCN 2015)*, 2015.

[8] ETSI, "Nfv-man 001 (v1.1.1) network functions virtualisation (nfv); management and orchestration," Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG), Technical Report, 12 2014.

[9] OpenStack. Open source software for creating private and public clouds. OpenStack. [Online]. Available: https://www.openstack.org/

[10] OpenStack Telco Working Group. Mission statement and scope. [Online]. Available: https://wiki.openstack.org/wiki/TelcoWorkingGroup

[11] Fraunhofer FOKUS. Open baton: An extensible and customizable nfv mano-compliant framework. [Online]. Available: https://openbaton.github.io/

[12] J. Soares, C. Gonalves, B. Parreira, P. Tavares, J. Carapinha, J. P. Barraca, R. L. Aguiar, and S. Sargento, "Toward a telco cloud environment for service functions," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 98–106, Feb 2015.

[13] Cobham Wireless and Stratus Technologies, "Ensuring high-availability and resiliency for nfv," Whitepaper, 2016.

[14] S. N. Abbot, J. A. Goodwin, M. B. Muttur, and T. H. Smith, "Predictively managing failover in high availability systems," U.S. Patent US9 229 843B2, Jan. 5, 2016. [Online]. Available: https://patentimages.storage.googleapis.com/35/33/69/6c041c7f491336/US9229843.pdf

[15] R. Jhawar and V. Piuri, "Chapter 9 - fault tolerance and resilience in cloud computing environments," in *Computer and Information Security Handbook (Third Edition)*, third edition ed., J. R. Vacca, Ed. Boston: Morgan Kaufmann, 2017, pp. 165 – 181. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128038437000090

[16] S. Lightner and F. Weckesser, "Fault tolerant architecture for distributed computing systems," U.S. Patent US9 201 744B2, Dec. 1, 2015. [Online]. Available: https://patentimages.storage.googleapis.com/5a/9d/11/5d4f1fc3e76f8a/US9201744.pdf

[17] M. Garca-Valls, T. Cucinotta, and C. Lu, "Challenges in real-time virtualization and predictable cloud computing," *Journal of Systems Architecture*, vol. 60, no. 9, pp. 726–740, Oct 2014.

[18] T. Erl, Z. Mahmood, and R. Puttini, *Cloud Computing: Concepts, Technology and Architecture*. Prentice Hall, 2013.

[19] Microsoft. Microsoft azure: Your vision. your cloud. Microsoft. [Online]. Available: https://azure.microsoft.com/en-us/

[20] VMware Inc. Vmware. [Online]. Available: https://www.vmware.com/

[21] Amazon Web Services Inc. aws. Amazon Web Services. [Online]. Available: https://aws.amazon.com/

[22] Google. Google cloud: Make your move here. Google. [Online]. Available: https://cloud.google.com/

[23] Oracle. Oracle cloud: Complete, integrated cloud. Oracle. [Online]. Available: https://cloud.oracle.com/

[24] G. Carella, M. Corici, P. Crosta, P. Comi, T. M. Bohnert, A. A. Corici, D. Vingarzan, and T. Magedanz, "Cloudified ip multimedia subsystem (ims) for network function virtualization (nfv)-based architectures," *Computers and Communication (ISCC), 2014 IEEE Symposium*, pp. 1–6, 2014.

[25] T. D. Quoc, H. Perkuhn, D. Catrein, U. Naumann, and T. Anwar, "Optimization and evaluation of a multimedia streaming service on hybrid telco cloud," *International Journal on Cloud Computing: Services and Architecture (IJCCSA)*, vol. 1, no. 2, p. 20 S., Aug 2011. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1109/1109.1583.pdf

[26] ETSI NFV Industry Specification Group, "Network function virtualisation (nfv) - network operator perspectives on nfv priorities for 5g," Technical Report, Feb. 2017. [Online]. Available: https://portal.etsi.org/NFV/NFV_White_Paper_5G.pdf

[27] A. Gupta and R. Jha, "A survey of 5g network: Architecture and emerging technologies," *Access, IEEE*, vol. 3, pp. 1206–1232, 08 2015.

[28] D. Kreutz, F. M. V. Ramos, P. E. Verssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[29] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, July 2013.

[30] ETSI NFV Industry Specification Group, "Network functions virtualisation - introductory white paper," Technical Report, Oct. 2012, published October 22-24, 2012 at the "SDN and OpenFlow World Congress", Darmstadt-Germany. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[31] ETSI NFV Industry Specification Group, “Network functions virtualisation - network operator perspectives on industry progress - white paper 3,” Technical Report, Oct. 2014, published October 14-17, 2014 at the ”SDN and OpenFlow World Congress”, Dusseldorf-Germany. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf

[32] SDxCentral. Nfv. [Online]. Available: https://www.sdxcentral.com/nfv/?c_action=num_ball

[33] B. Yi, X. Wang, K. Li, S. Das, and M. Huang, “A comprehensive survey of network function virtualization,” *Computer Networks*, vol. 133, 03 2018.

[34] ETSI ISG NFV. Nfv. ETSI. [Online]. Available: https://www.etsi.org/technologies-clusters/technologies/nfv

[35] X. P. Costa, A. Festag, H.-J. Kolbe, J. Quittek, S. Schmid, M. Stiemerling, J. Swetina, and H. van der Veen, “Latest trends in telecommunication standards,” *Computer Communication Review*, vol. 43, pp. 64–71, 2013.

[36] M. Cilloni, “Design and implementation of an etsi network function virtualization-compliant container orchestrator,” M. Eng. thesis, University of Bologna, Bologna, Italy, Mar. 2017. [Online]. Available: http://amslaurea.unibo.it/13373/

[37] Microsoft Azure. What is a virtual machine? [Online]. Available: https://azure.microsoft.com/en-us/overview/what-is-a-virtual-machine/

[38] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, “Survey on network virtualization hypervisors for software defined networking,” *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 655–685, 2016.

[39] K. Katsalis, N. Nikaein, and A. Edmonds, “Multi-domain orchestration for nfv: Challenges and research directions,” in *2016 15th International Conference on Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security (IUCC-CSS)*, Dec 2016, pp. 189–195.

[40] Open Baton. Open baton zabbix plugin. [Online]. Available: https://github.com/openbaton/zabbix-plugin

[41] Cloudify. Cloudify: Cloud native transformation at the speed of business. Cloudify. [Online]. Available: https://cloudify.co

[42] T. Nozoe, M. Noguchi, M. Sakuma, and M. Isawa, "Live migration of virtualized carrier grade sip server," *International Journal of Communication Networks and Information Security (IJCNIS)*, vol. 8, no. 2, pp. 57–63, Aug 2016.

[43] M. Tsietsi, A. Terzoli, and G. Wells, "Mobicents as s service creation and deployment environment for the open ims core," *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, 2009.

[44] TelecomABC. Ims. [Online]. Available: http://www.telecomabc.com/i/ims.html

[45] P. Janert, C. Kelley, and T. Williams, *Gnuplot in Action - Understanding data with graphs*, 1st ed., N. Miller, T. Cirtin, and B. Berg, Eds. Manning Publications Co., 2010.

[46] Core Network Dynamics. Opensourceimscore by cnd. [Online]. Available: http://www.openimscore.com/

[47] M. Poikselk and G. Mayer, *The IMS: IP Multimedia Concepts and Services*, 3rd ed. John Wiley & Sons, 2013, 2013.

[48] Kamailio. Kamailio - the open source sip server. [Online]. Available: https://www.kamailio.org/w/

[49] Clearwater. Welcome to project clearwater - ims in the cloud. [Online]. Available: http://www.projectclearwater.org

[50] D. Thißen, J. Miguel, E. Carlín, and R. Herpertz, "Evaluating the performance of an ims/ngn deployment," in *GI Jahrestagung*, 2009.

[51] E. M. Nahum, J. Tracey, and C. P.Wright, "Evaluating sip server performance," *SIGMETRICS'07*, pp. 349–350, Jun. 2007.

[52] D. Verbeiren, P. Lecluse, and X. Simonart. (2014, 04) Ims bench sipp. [Online]. Available: http://sipp.sourceforge.net

[53] opensource.com. What is openstack. [Online]. Available: https://opensource.com/resources/what-is-openstack

[54] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: Toward an open-source solution for cloud computing," *International Journal of Computer Applications (0975 - 8887)*, 2012.

[55] OpenStack. Heat. [Online]. Available: https://wiki.openstack.org/wiki/Heat

[56] OpenStack. Telemetry. [Online]. Available: https://wiki.openstack.org/wiki/Telemetry

[57] OpenStack. (2017, Nov.) Ceilometer's documentation. OpenStack. [Online]. Available: https://docs.openstack.org/ceilometer/latest/

[58] The Gnocchi Developers. (2017) Gnocchi - metric as a service. [Online]. Available: http://gnocchi.xyz/

[59] SDxCentral. ebrief: The pros and cons of openstack. (Accessed on 02/10/2019). [Online]. Available: https://www.sdxcentral.com/resources/sponsored/ebriefs/pros-cons-openstack-2018-02/

[60] A. Vogel, D. Griebler, C. A. F. Maron, C. Schepke, and L. G. Fernandes, "Private iaas clouds: A comparative analysis of opennebula, cloudstack and openstack," *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, 02 2016.

[61] Trilio Content Team. Why openstack is taking off in the enterprise. (Accessed on 02/10/2019). [Online]. Available: https://www.trilio.io/resources/why-openstack/

[62] OpenStack. The world runsonopenstack. (Accessed on 02/10/2019). [Online]. Available: https://www.openstack.org/user-stories/

[63] Mark Shiozaki. The top 3 openstack benefits and challenges. (Accessed on 02/10/2019). [Online]. Available: https://www.stratoscale.com/blog/openstack/the-top-3-openstack-benefits-and-challenges/

[64] L. Nobach, O. Hohlfeld, and D. Hausheer, "New kid on the block: network functions visualization: from big boxes to carrier clouds," in *CCRV*, 2018.

[65] A. Desai, R. Oza, P. Sharma, and B. Patel, "Hypervisor: A survey on concepts and taxonomy," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 2, no. 3, pp. 222–225, Feb. 2013.

[66] E. Siebert. Understanding hosted and bare-metal virtualization hypervisor types. [Online]. Available: https://searchservervirtualization.techtarget.com/tip/Understanding-hosted-and-bare-metal-virtualization-hypervisor-types

[67] pfSense. pfsense: Open source security. [Online]. Available: https://www.pfsense.org/

[68] ETSI, “Etsi gs nfv-swa 001 v1.1.1 (2014-12) network functions virtualisation (nfv); virtual network functions architecture,” Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG), Technical Report, 12 2014.

[69] Zabbix. Zabbix doucumentation. [Online]. Available: https://www.zabbix.com/documentation/3.0/start

[70] Open Baton. How to install and configure zabbix to securely monitor remote servers on ubuntu 16.04. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-zabbix-to-securely-monitor-remote-servers-on-ubuntu-16-04

[71] OpenBaton. Zabbix server 3.0 installation and configuration. [Online]. Available: https://openbaton-docs.readthedocs.io/en/develop/zabbix-server-configuration-3.0/

[72] Voip-info.org. Sipp. [Online]. Available: https://www.voip-info.org/sipp/

[73] SIPp community. Performance testing with sipp sipp 3.5 documentation. (Accessed on 12/11/2018). [Online]. Available: https://sipp-wip.readthedocs.io/en/latest/perftest.html

[74] Chef. Open source chef - ensure configurations are applied consistently in every environment, at any scale, with infrastructure automation. [Online]. Available: https://www.chef.io/chef/

[75] Metaswitch. Openstack heat templates for clearwater deployments. [Online]. Available: https://github.com/Metaswitch/clearwater-heat

[76] Orange-OpenSource. Cloudify clearwater get starting installation. [Online]. Available: https://github.com/Orange-OpenSource/opnfv-cloudify-clearwater