



Individual Project

Evaluation of Juju as a VNFM for Container Virtualization

Submitted by:	Baharul Alam (1187786)
First examiner:	Professor Dr.-Ing. Ulrich Trick
Supervisor:	M.Eng. Gregor Frick
Date of start:	04.09.2018
Date of submission:	04.12.2018

Statement

I confirm that I have written this student project on my own. No other sources were used except those referenced. Content which is taken literally or analogously from published or unpublished sources is identified as such. The drawings or figures of this work have been created by myself or are provided with an appropriate reference. This work has not been submitted in the same or similar form or to any other examination board.

Date, signature of the student

Acknowledgment

Firstly, I would like to thank to Prof. Dr. -Ing Ulrich Trick of the Frankfurt am Main University of Applied Sciences for allowing me to do my student project in this telecommunication field which will help me to be solid foundation for my carrier in the future and being my examiner in this individual project.

Besides that, I would like also to say thank to Mr. Gregor Frick for organizing this topic and instructing me whenever facing difficulties. His guidelines and appreciations during the project help me to keep me on right track and to complete the project.

Moreover, it would be a big mistake here if I did not mention the support and encouragement of my family. The student project would be unsuccessful if I was not encouraged and supported mentally and practically by the members in TK-Lab in FH FFM. I finally would like to thank my friends and seniors for their suggestions and guidance during the project and writing of this project report.

Content

1	Introduction	6
2	Theoretical Background	7
2.1	Network Function Virtualization	7
2.1.1	Network Function Virtualization	7
2.1.2	NFV Framework	8
2.2	Virtualized Infrastructure Manager (VIM)	9
2.3	Network Functions Virtualization Orchestrator (NFVO)	9
2.4	Virtual Network Functions Manager (VNFM)	10
2.5	Cloud Computing	10
2.6	Container Virtualisation	11
2.6.1	Difference between Containers and Virtual Machines	12
2.6.2	Linux Container (LXC)	13
2.6.3	Linux container Daemon (LXD)	14
2.6.4	Juju	15
2.7	Kamailio	19
2.8	Session Initiation Protocol	19
2.9	VirtualBox	20
3	Requirements Analysis	22
3.1	General Objectives	22
3.2	Clarifying the Requirements	22
3.3	Time frames	22
3.4	Target State	23
3.5	Software Requirements	23
4	Realisation	25
4.1	Prepare the Environment in VirtualBox	25
4.2	Installation of LXC, LXD and Juju	32
4.3	Juju Command	33
4.3.1	Controller Management	33
4.3.2	Model Management	35
4.4	Kamailio Charm	36
4.4.1	Create Charm Template	36
4.4.2	Writing charm	37
5	Demonstration	42
5.1	Deploy MySQL	42
5.2	Deploy Kamailio	43

5.3	IP table setup	45
5.4	Call setup	45
6	Evaluation of Juju as VNFM	50
7	Summary and Perspectives	51
8	Abbreviations	53
9	References	55

1 Introduction

In modern age, technology is one of the important parts in human life. In one sentence life without technology is impossible now a days. Modern technology is increasing day by day to fulfil the user's choice. With the increasing of the technology, software is going to be complex. Micro services are increasing to fulfil the requirements for users. So, application is going to be complex. As a result, a company need to hire lots of people to manage those applications and sometimes it's not possible to train and hire people within a short time. For training and hiring people is expensive compare to application.

Cloud computing is one of the modern technologies in recent years. It's shared the same computer resources with multiple users. One of the main focuses on cloud computing is reduce the cost and accessibility from anywhere. Because of cloud computing, user don't need to invest on IT infrastructure cost and they can access on cloud application over internet from anywhere. To implement an application on cloud is like implement an application on local machine. But managing those application is complex as one application is dependent with others application or sometimes one application must fulfill lots of functionality to install. On the other hand, one application may be installed in different machine with the same configuration. So, it is tedious and increasing the application maintenance cost for application users.

Juju is one of the open source projects launched by Canonical to deploy, manage resources and scale software. At the same time Juju can be connected multiple services in the cloud platform. Juju can be expressed it service as automation in cloud. This automation technology is very new and promising in cloud computing. Juju will be reduced the managing software complexity and time also. It will make easier life for providing microservices.

The main purpose of this Student project is to implement Juju as a Virtual Network Function Manager (VNFM) for container virtualization and evaluation Juju as a VNFM. In the theoretical part of this student project will cover Linux container (LXC) technology, Linux container Daemon (LXD), Network functional Virtualization (NFV), Virtual Network Function Manager (VNFM), cloud computing, Juju and so on. The student project implementation part will cover the setup Linux operating system, install LXC, LXD and Juju. Writing a bash script for Kamailio server. Deploy MySQL and Kamailio application container from Juju and manage them as they can provide Voice Over Internet Protocol (VOIP) service.

The names of the main chapters of this student project documentation are in the below:

1. Introduction
2. Theoretical Background
3. Requirements Analysis
4. Realisation
5. Demonstration
6. Summary and Future Perspectives
7. Abbreviations
8. References

2 Theoretical Background

This chapter will discuss all the theoretical background which used in this student project. To understand the implementation of this student work it is essential to discuss the theoretical part from all perspective which are related to with the technology. It is very essential and useful to understand the theories, architecture, software which has been used in this project work. Firstly, introduction about virtualized Network Function Manager with Network Function Virtualization and container technology. Furthermore, the overview of Juju will be explained and some others theoretical parts which are including of this project also will be discussed.

In this chapter will discuss about:

- Network Function Virtualization
- Virtualized Network Function Manger
- Cloud computing
- Container technology
- Linux container
- LXD
- Juju
- MySQL
- Kmailio
- Bash script

In every section there have several sub sections which will cover the theoretical background and relation between the technology with this work.

2.1 Network Function Virtualization

In this chapter, will be discussed about Network Function Virtualization and architecture of NFV. In architectural part some components will be discussed and showed how they have connected each other.

2.1.1 Network Function Virtualization

Network function virtualization also known as NFV. NFV is an evolving transformation of global network architecture. Moving away from single-function hardware, NFV helps telecoms and enterprise move toward an elastic, cloud and software-based network that offers streamlined service provisioning, lowered Capital Expenses (CpaEx) and Operating Expenses (OpEx) and significant agility (OPNFV, 2016).

The concept of Network Function Virtualisation (NFV) originated from the requirement of telecommunication service providers worldwide, to accelerate deployment of new network services and to support their revenue and future growth objectives. Present day telecommunication networks are over populated with a large and increasing variety of proprietary hardware appliances. Some examples of typical telecom equipment are routers, switches, base stations, firewalls, voice gateways, and Internet Protocol (IP) Multimedia Subsystem (IMS) and Mobile Packet Core. These types of equipment are typically monolithic in design; that is, they consist of hardware, software, and associated management systems. This is further compounded by the increasing costs of energy, capital investment, requirement of huge technical manpower etc. The variety of skills necessary to design, integrate and operate increasingly complex hardware-based appliances, poses real challenge to both developer and the network operator (FN Division Telecommunication engineering centre) (Balamurali Thekkedath, 2016).

Network Function virtualization is a new way to design, deploy, and manage networking services by decoupling the physical network equipment from the function that run on them, which replaces hardware centric, dedicated

network devices with software running on general-purpose CPUs or virtual machines, operating on standard servers (FN Division Telecommunication engineering centre).

2.1.2 NFV Framework

The European Telecommunication Standards Institute (ETSI) is a recognized regional standards body made up of CSPs working together to define frameworks for solutions used in their networks. The ETSI NFV Industry Specification Group (ISG) has defined a high-level functional architectural framework for NFV. These standards and frameworks are used as a base by vendors when creating their offerings (Balamurali Thekkedath, 2016).

NFV framework is illustrated in Figure 2-1 is (Balamurali Thekkedath, 2016):

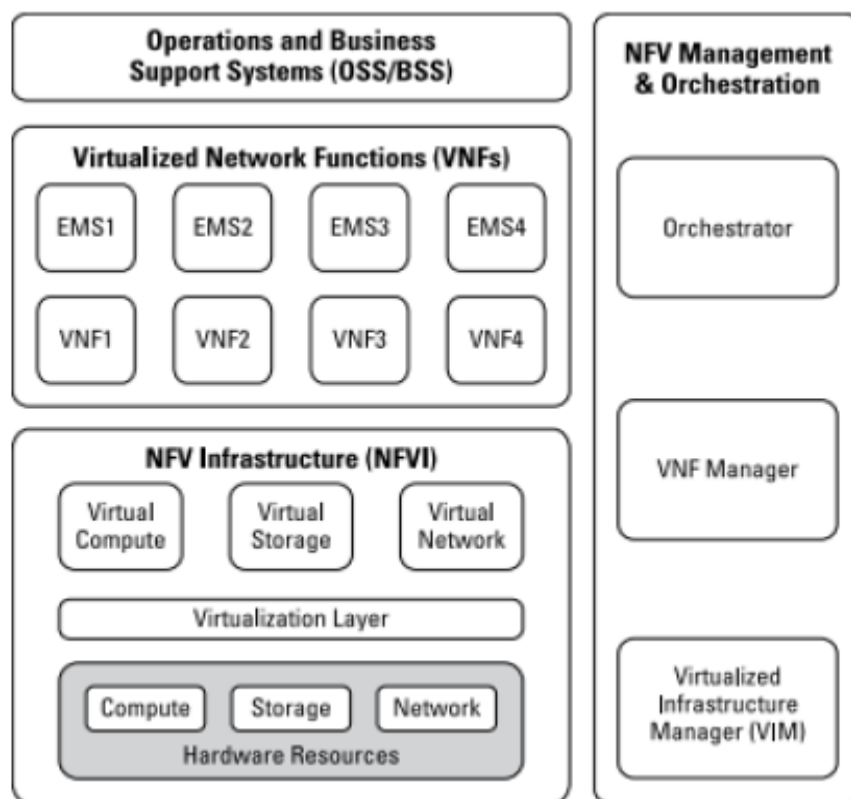


Figure 2-1 ETSI NFV Framework (Balamurali Thekkedath, 2016)

The ETSI NFV framework consists of three major components (Balamurali Thekkedath, 2016):

1. **Network Function Virtualization Infrastructure (NFVI):** A subsystem that consists of all the hardware (servers, storage, and networking) and software components on which Virtual Network Functions (VNFs) are deployed. This includes the compute, storage, and networking resources, and the associated virtualization layer (hypervisor).
2. **Management and Orchestration (MANO):** A subsystem that includes the Network Functions Virtualization Orchestrator (NFVO), the virtualized infrastructure manager (VIM), and the Virtual Network Functions Manager (VNFM).
3. **Virtual Network Functions (VNFs):** VNFs are the software implementation of network functions that are instantiated as one or more virtual machines (VMs) on the NFVI.

The figure 2-2 includes references for interfaces between the different architectural components (Balamurali Thekkedath, 2016).

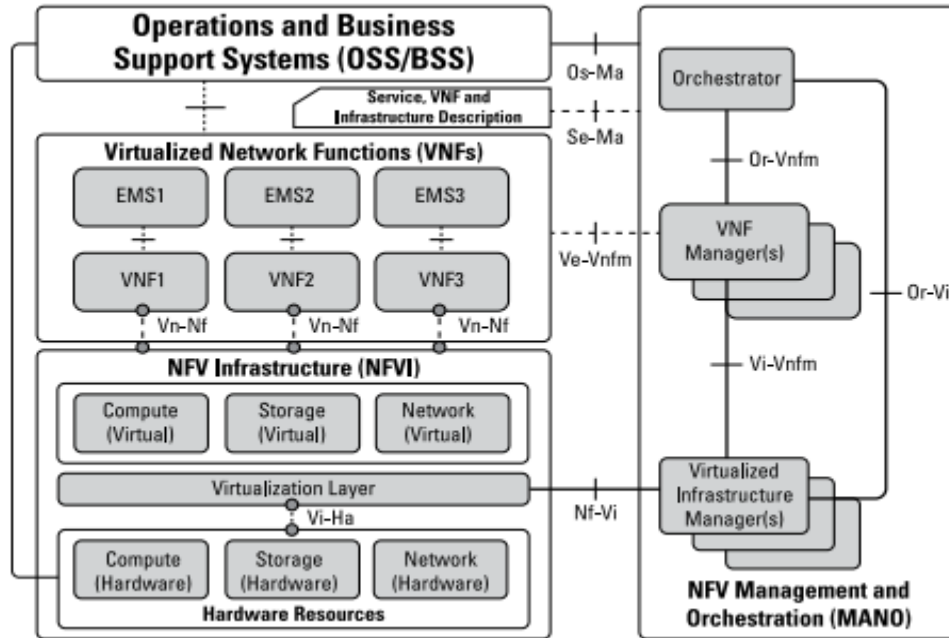


Figure 2-2 High Level NFV Architectural Framework with Interface (Balamurali Thekkedath, 2016)

2.2 Virtualized Infrastructure Manager (VIM)

The virtualized Infrastructure Manager (VIM) is responsible for controlling and managing the NFVI compute, storage and network resources, usually within one operator's Infrastructure Domain (e.g. all resources within an NFVI-PoP, resources across multiple NFVI-POPs, or a subset of resources within an NFVI-PoP). A virtualized Infrastructure Manager (VIM) may be specialized in handling a certain type of NFVI resource (e.g. compute-only, storage-only, networking-only), or may be capable of managing multiple types of NFVI resources (e.g. in NFVI-Nodes) (ETSI GS NFV-MAN 001, 2014-12).

2.3 Network Functions Virtualization Orchestrator (NFVO)

The NFVO functionality can be divided into two broad categories: (1) resource orchestration, and (2) service orchestration. The NFVO uses the Resource Orchestration functionality to provide services that support accessing NFVI resources in an abstracted manner independently of any VIMs, as well as governance of VNF instances sharing resources of the NFVI infrastructure. Service Orchestration functions to coordinate groups of VNF instances as Network Services that jointly realize a more complex function, including joint instantiation and configuration, configuring required connections between different VNFs, and managing dynamic changes of the configuration, e.g. for scaling the capacity of the Network Service. The Network Service Orchestration function uses the services exposed by the VNF Manager function and by the Resource Orchestration function (ETSI GS NFV-MAN 001, 2014-12).

2.4 Virtual Network Functions Manager (VNFM)

The Virtual Network Function Manager (VNFM) is the entity that manages the virtualized network functions (see Figure 2-2). Traditionally, the management component of a network function focuses on Fault Management, Configuration Management, Accounting Management, Performance Management, and Security Management (FCAPS). With the introduction of virtualization, additional aspects of managing the lifecycle of the VNF become a key function of the management component. The VNFM and the Element Management System (EMS) are closely aligned in providing overall management support for the VNF (Balamurali Thekkedath, 2016).

The VNF Manager is responsible for the lifecycle management of VNF instances. Each VNF instance is assumed to have an associated VNF Manager. A VNF manager may be assigned the management of a single VNF instance, or the management of multiple VNF instances of the same type or of different types (ETSI GS NFV-MAN 001, 2014-12).

Most of the VNF Manager functions are assumed to be generic common functions applicable to any type of VNF. However, the NFV-MANO architectural framework needs to also support cases where VNF instances need specific functionality for their lifecycle management, and such functionality may be specified in the VNF Package (ETSI GS NFV-MAN 001, 2014-12).

The following list expresses the non-exhaustive set of functions performed by the VNF Manager function. These functionalities may be exposed by means of interfaces and consumed by other NFV-MANO functional blocks or by authorised external entities (ETSI GS NFV-MAN 001, 2014-12):

- VNF instantiation, including VNF configuration if required by the VNF deployment template (e.g. VNF initial configuration with IP addresses before completion of the VNF instantiation operation).
- VNF instantiation feasibility checking, if required.
- VNF instance software update/upgrade.
- VNF instance modification.
- VNF instance scaling out/in and up/down.
- VNF instance-related collection of NFVI performance measurement results and faults/events information, and correlation to VNF instance-related events/faults.
- VNF instance assisted or automated healing.
- VNF instance termination.
- VNF lifecycle management change notification.
- Management of the integrity of the VNF instance through its lifecycle.
- Overall coordination and adaptation role for configuration and event reporting between the VIM and the EM.

2.5 Cloud Computing

Cloud computing is a modern computing paradigm that provides resources such as processing, storage, network and software that can be accessed over the Internet from any remote locations. Users can access cloud applications using web browsers, thin client machines or mobile devices, while all the data are processing.

In functions more efficiently since it offloads management and maintenance tasks, thereby making computing a service rather than a product. The sharing of hardware resources is expected to reduce idle time of machines and increase their productivity. Due to the shift from computing being a resource to a utility, cloud computing introduces billing models that are based on time and utilities. On demand availability, ease of provisioning, dynamic and virtually infinite scalability are some of the key attributes of cloud computing. Services within the cloud are typically provided under the following three categories (Azhagappan, D., 2015):

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)

- Software as a Service (SaaS)

Infrastructure as a Service (IaaS)

Infrastructure as a Service (IaaS) is a way of delivering Cloud Computing infrastructure – servers, storage, network and operating systems as an on-demand service. Rather than purchasing servers, software, datacentre space or network equipment, clients instead buy those resources as a fully outsourced service on demand. Its balancers and networking services are provided (Azhagappan, D., 2015).

The following are the characteristics of IaaS (Azhagappan, D., 2015):

- Resources are distributed as a service
- Allows for dynamic scaling
- Has a variable cost, utility pricing model.
- Generally, includes multiple users on a single piece of hardware

Platform as a Service (PaaS)

PaaS can be defined as a computing platform that allows the creation of web applications quickly and easily and without the complexity of buying and maintaining the software and infrastructure underneath it. In the PaaS model, a computing platform or solution stack including operating system, programming language execution environment, database, and web server are typically provided (Azhagappan, D., 2015).

The following are the characteristics of PaaS (Azhagappan, D., 2015):

- Services to develop, test, deploy, host and maintain applications in the same integrated development environment.
- Web based user interface creation tools help to create, modify, test and deploy different UI scenarios.
- Multi-tenant architecture where multiple concurrent users utilize the same development application.
- Built in scalability of deployed software including load balancing and failover.
- Integration with web services and databases via common standards.
- Support for development team collaboration.
- Tools to handle billing and subscription management.

Software as a service (SaaS)

With the SaaS model, cloud providers install and operate application software in the cloud, which user access using cloud clients on computers, mobile devices, browsers etc (Azhagappan, D., 2015).

The following are the characteristics of SaaS (Azhagappan, D., 2015):

- Web access to commercial software.
- Software is managed from a central location.
- Software delivered in a ‘one to many’ model.
- Users need not handle software upgrades and patches.
- Application Programming Interfaces (APIs) allow for integration between different pieces of software.

2.6 Container Virtualisation

Container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging (Docker, 2017).

Containers began life under the assumption that the operating system itself could be virtualized in such a way that, instead of starting with virtual hardware, one could start instead with virtualizing the operating system kernel API. Operating System virtualization means separating static resources (like memory or network interfaces)

into pools, and dynamic resources (like I/O bandwidth or CPU time) into shares that are allotted to the virtual system (Bottomley and Emelyanov, 2014).

The following Figure 2-3 will illustrate the general architecture of container:

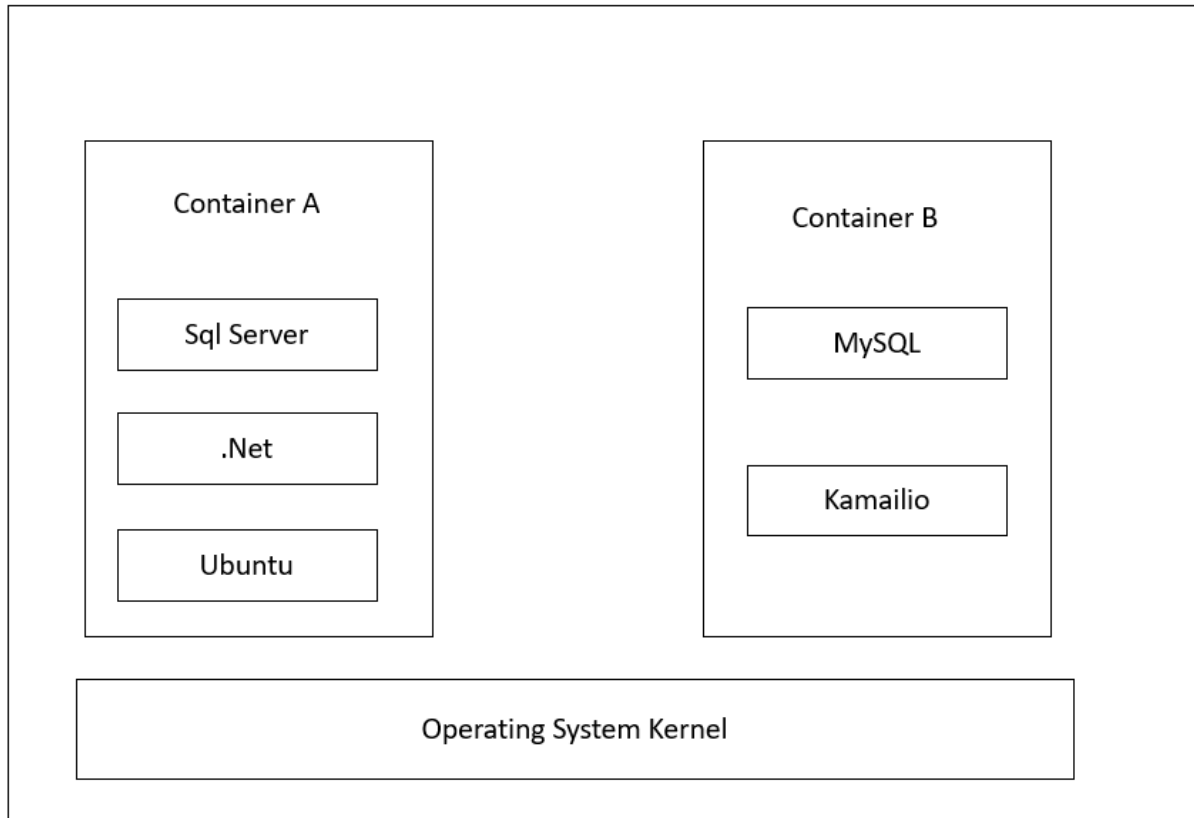


Figure 2-3 Container Architecture

From Figure 2-3 it is seen that in one single host is contain two containers running. Each container has the necessary environment itself.

So as a basic definition, containers could also be called operating system virtualization where the virtual layer runs as an application within one OS. When combined with a cloud platform, user don't have to worry about investing in multiple servers and other infrastructure to run user applications (Joyner, 2016).

2.6.1 Difference between Containers and Virtual Machines

Virtual machine (VM) is one of the eldest famous virtualization techniques. But now a days Container is one of the most popular virtualization technologies. But there are some basic differences between virtual machine and container. The concept of VM and container illustrated in below Figure 2-4:

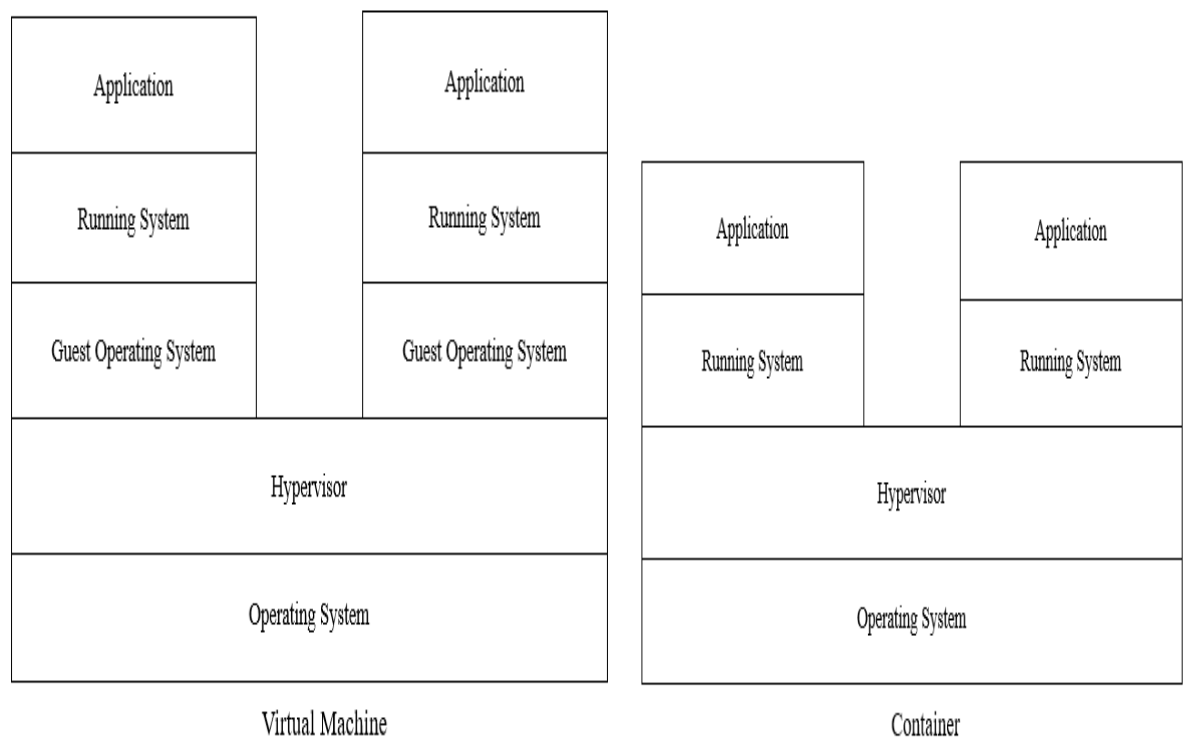


Figure 2-4 Virtual Machine and Container architecture Comparison

Virtual Machine (VM) can be described as a software program that emulates the functionality of a physical hardware or computing system. It runs on top of an emulating software called the hypervisor, which replicates the functionality of the underlying physical hardware resources with a software environment. These resources may be referred to as the Host Machine, while the VM that runs on the hypervisor is often called a Guest Machine. The virtual machine contains all necessary elements to run the apps, including the computing, storage, memory, networking and hardware functionality available as a virtualized system. It may also contain the necessary system binaries and libraries to run the apps, while the OS is managed and executed using the hypervisor (bmc blogs, 2018).

Containerization creates abstraction at an OS level that allows individual, modular and distinct functionality of the app to run independently. As a result, several isolated workloads can dynamically operate using the same physical resources. Containers can run on top of bare metal servers, hypervisors, or in the cloud infrastructure. They share all necessary capabilities with the VM to operate as an isolated OS environment for a modular app functionality with one key difference. Using a containerization engine such as the Docker Engine, containers create several isolated OS environments within the same host system kernel, which can be shared with other containers dedicated to run different functions of the app. Only bins, libraries and other runtime components are developed or executed separately for each container, which makes them more resource efficient as compared to VMs (bmc blogs, 2018).

In this project work, Linux container has been used with Linux container daemon. So, all the further discussion is based on Linux container with Linux container daemon.

2.6.2 Linux Container (LXC)

A Linux container is a set of one or more processes that are isolated from the rest of the system. All the files necessary to run them are provided from a distinct image, meaning that Linux containers are portable and con-

sistent as they move from development, to testing, and finally to production. This makes them much quicker than development pipelines that rely on replicating traditional testing environments (Containers, 2018).

Linux containers can be applied to problems in many different ways where ultimate portability, configurability, and isolation is needed. The point of Linux containers is to develop faster and meet business needs as they arise. No matter the infrastructure—on-premise, in the cloud, or a hybrid of the two—containers meet the demand. Of course, choosing the right container platform is just as important as the containers themselves (Containers, 2018).

Linux Container or LXC is a user space interface for the Linux kernel containment features. Through a powerful API and simple tools, it lets Linux users easily create and manage system or application containers. LXC containers are often considered as something in the middle between a chroot and a full-fledged virtual machine. The goal of LXC is to create an environment as close as possible to a standard Linux installation but without the need for a separate kernel (Linux Containers).

Current LXC uses the following kernel features to contain processes:

- Kernel namespaces (Inter process communication (ipc), mount, Process ID (pid), network and user)
- Apparmor and SELinux profiles
- Seccomp policies
- Chroots (using pivot_root)
- Kernel capabilities
- CGroups (control groups)

2.6.3 Linux container Daemon (LXD)

LXD is a next generation system container manager know as Linux Container Daemon. It offers a user experience similar to virtual machines but using Linux containers instead. The core of LXD is a privileged daemon which exposes a REST API over a local Unix socket as well as over the network (if enabled).

Clients, such as the command line tool provided with LXD itself then do everything through that Representational State Transfer application programming interface (REST API). It means that whether local host or a remote server, everything works the same way.

LXD isn't a rewrite of LXC, in fact it's building on top of LXC to provide a new, better user experience. Under the hood, LXD uses LXC through liblxc and its Go binding to create and manage the containers (Linux Containers).

Some of the biggest features of LXD are (Linux Containers):

- Secure by design (unprivileged containers, resource restrictions and much more)
- Scalable (from containers on laptop to thousands of compute nodes)
- Intuitive (simple, clear API and crisp command line experience)
- Image based (with a wide variety of Linux distributions published daily)
- Support for Cross-host container and image transfer (including live migration with CRIU)
- Advanced resource control (CPU, memory, network I/O, block I/O, disk usage and kernel resources)
- Device passthrough (USB, GPU, Unix character and block devices, NICs, disks and paths)
- Network management (bridge creation and configuration, cross-host tunnels, ...)
- Storage management (support for multiple storage backends, storage pools and storage volumes)

In the Figure 2-5 has been illustrated the concept of Linux container daemon with Linux container:

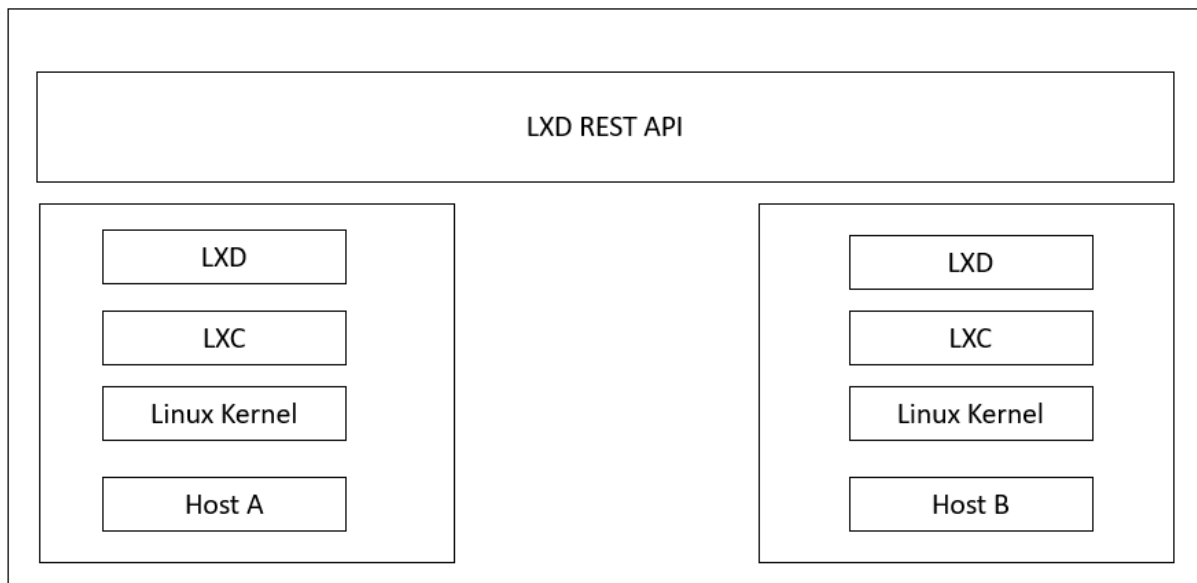


Figure 2-5 Linux container with LXD

LXD is working with Linux container. So, LXD is dependent on LXC. One user can use LXC without LXD but not LXD without LXC. If user wants to virtualize an entire operating system (OS) then LXD is the best way to proceed. Multiple host can be connected with LXD REST API.

2.6.4 Juju

Juju is a state-of-the-art, open source modelling tool for operating software in the cloud. Juju allows to deploy, configure, manage, maintain, and scale cloud applications quickly and efficiently on public clouds, as well as on physical servers, OpenStack, and containers. Juju can be used from the command line or through its beautiful GUI (Juju).

Juju is a generic Virtual Network Function Manager (VNFM) in the ETSI NFV architecture. it is not a specific Network Function Virtualization Orchestrator (NFVO). Juju is a universal service modelling system (Artur Tyloch, 2015).

Services are first-class concepts in the Juju model, in contrast with traditional configuration management systems which focus on machines. This service-orientation makes Juju particularly well suited to the role of VNFM, enabling higher-level orchestrators to make business decisions and articulate those decisions very clearly and simply through the underlying model provided by Juju (Artur Tyloch, 2015).

In Juju, a service is provided by a group of units. The scale of the service is determined by the number of units providing that service, and availability (“HA”) of the service is often determined by the number of units combined with their relationship to heartbeat or monitoring systems. Juju models both traditional scale-up software, and modern scale-out software, equally well. Placement of service units on machines can be done by Juju (Artur Tyloch, 2015).

Juju is particularly good at service composition – the combination of multiple services into a single functional system. It provides two mechanisms for such composition – integration of services across the network through standard interfaces, and the ability to co-locate services on a common machine. This is valuable because it decouples service definitions from implementation-specific preferences such as monitoring, enabling easier reuse of service definitions in different environments (production, test) and different organisations (Artur Tyloch, 2015).

The following figure 2-6 is illustrating the juju as VNFM (Artur Tyloch, 2015):

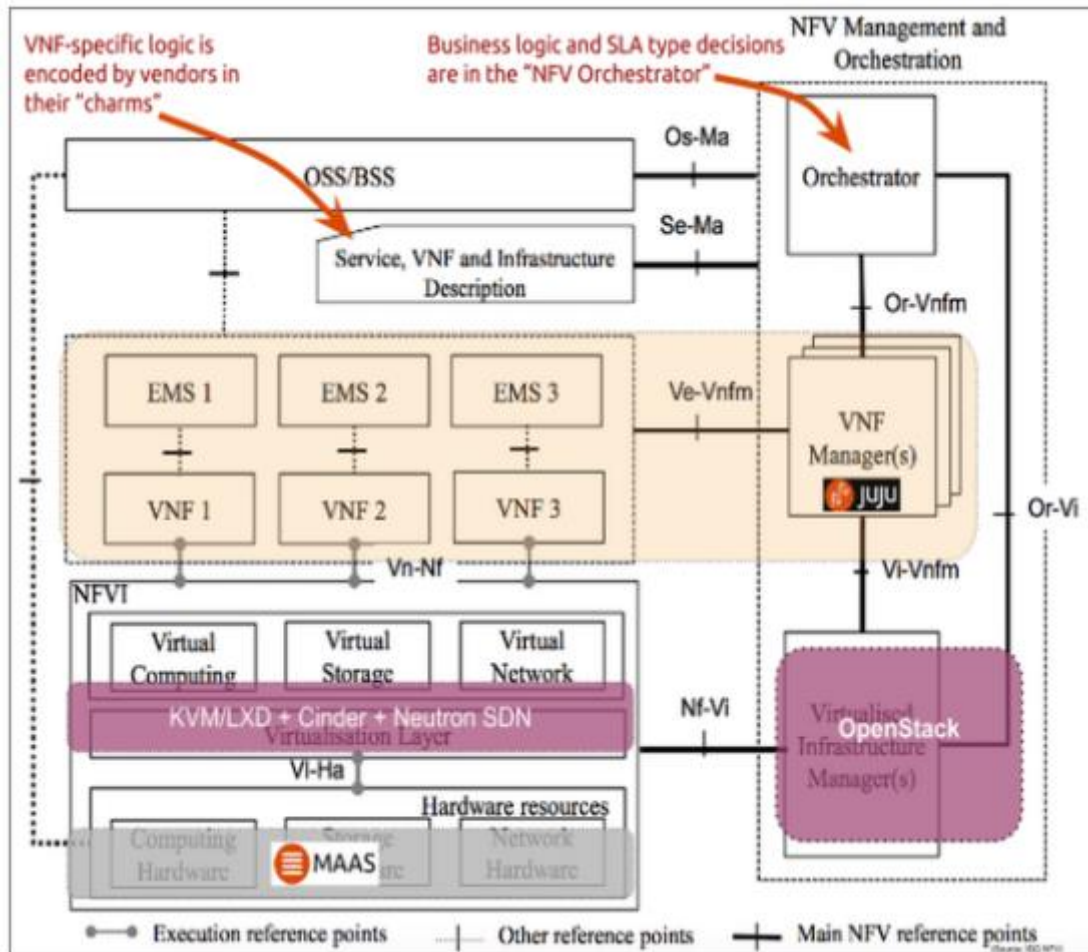


Figure 2-6 Juju as Virtual Network Function Manager (Artur Tyloch, 2015)

Juju has different important terms to control application in cloud:

- Controller
- Model
- Charm
- Bundle
- Machine
- Hooks

Controller

Controller is the management node of a Juju cloud environment. It's called the heart of Juju. It is the initial cloud instance which is created for Juju to gain access to a cloud. It is created by having the Juju client contact the cloud's API. Controller take care of all operations requested by Juju client (Juju).

Model

A model is associated with a single controller and is the space within which application units are deployed. A controller can have an indefinite number of models and each model can have an indefinite number of machines (and thus applications). Models themselves can be shared amongst Juju users (Artur Tyloch, 2015).

After creating a Controller, it has two models, the 'controller' model and the 'default' model. The primary purpose of the 'controller' model is to run and manage the Juju API server and the underlying database. The purpose of 'default' model or others model are to deploy applications (Juju).

Charm

Juju Charm contains all the instructions necessary for deploying and configuring application units. Charms are publicly available. Charms make it easy to reliably and repeatedly deploy applications, then scale up as desired with minimal effort (Juju).

Services in Juju are defined by Charms, which are similar to packages in that they are self-contained, self-describing encapsulations of operational code (that handles configuration and service lifecycle) as well as the software itself. Charms can include all the resources needed for the full lifecycle of the service (Artur Tyloch, 2015).

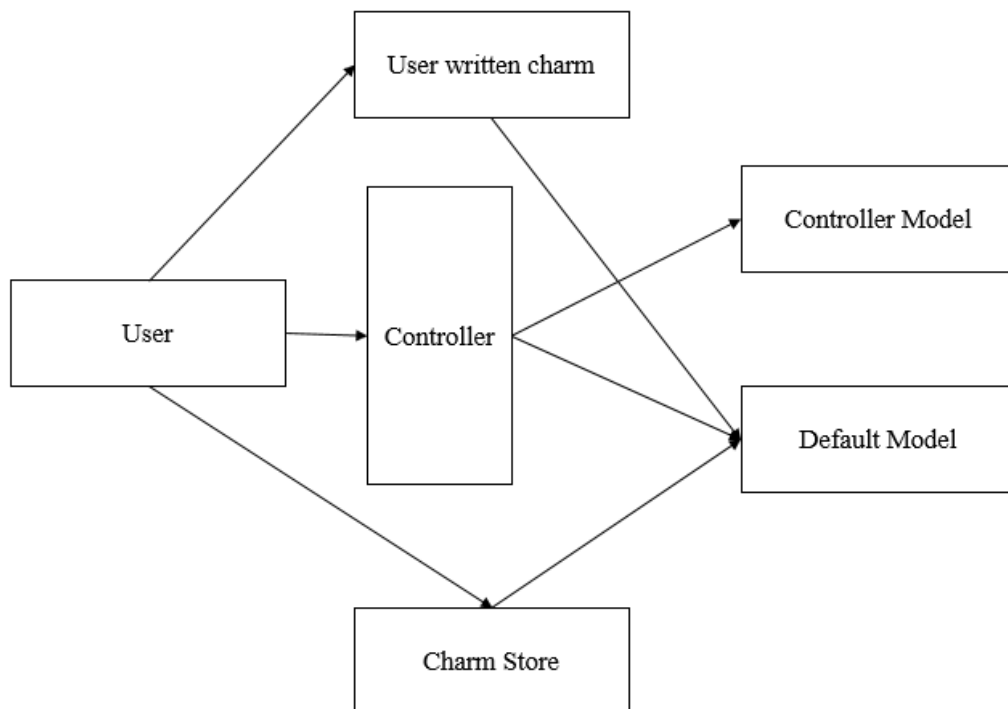


Figure 2-7 Charm deployment from user

From Figure 2-7: a user can create new written charm or directly use from charm store. When user deploy a charm, it will be deployed a machine on default model or any new created model. User can control those charms through the controller.

Bundle

A Juju bundle is a collection of charms which have been carefully combined and configured in order to automate a multi-charm solution. For example, a WordPress bundle may include the 'wordpress' charm, the 'mysql' charm, and the relation between them. The operations are transparent to Juju and so the deployment can continue to be managed by Juju as if everything was performed manually (Juju).

Machine

A Juju machine is the term used to describe a cloud instance that was requested by Juju. Machines will usually house a single unit of a deployed application, but this is not always the case (Juju).

Unit and application

A Juju unit (or application unit) is deployed software. Simple applications may be deployed with a single application unit, but it is possible for an individual application to have multiple units running in different machines. All units for a given application will share the same charm, the same relations, and the same user-provided configuration (Juju).

Endpoint

An endpoint (or application endpoint) is used to connect to another application's endpoint in order to form a relation. An endpoint is defined in a charm's `metadata.yaml` (Juju).

Interface

An interface is the communication protocol used over a relation between applications.

Relation

Charms contain the intelligence necessary for connecting different applications together. These inter-application connections are called relations, and they are formed by connecting the applications' endpoints. Endpoints can only be connected if they support the same interface and are of a compatible role (Juju).

The overview of Juju architecture is explained in below Figure 2-8:

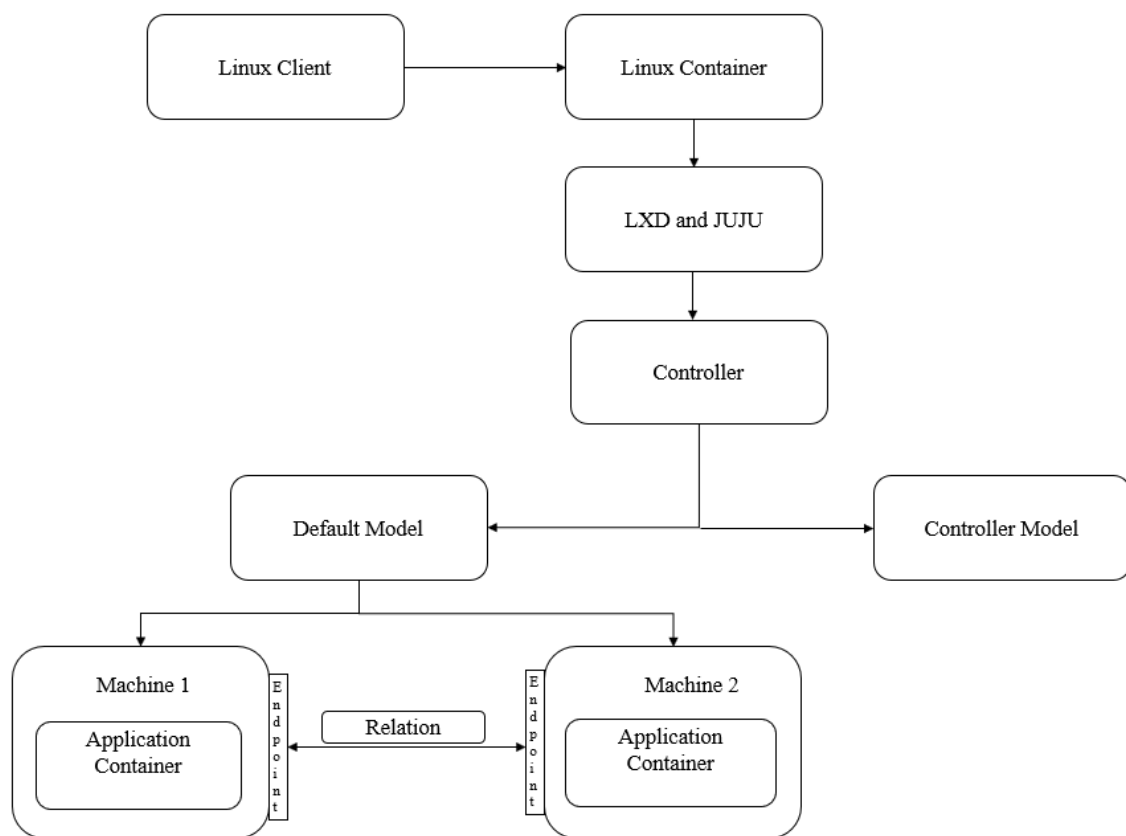


Figure 2-8 Overview of Juju architecture (Juju)

Hooks

Juju manages the service lifecycle with hooks implemented inside charms, in any language. At present there are 5 “unit-hooks” with predefined names that can be implemented by any charm:

- install
- config-changed
- start
- upgrade-charm
- stop

In practice, hooks are scripts (written in any language) called during the lifecycle of a service. They are triggered either by the commands run (like ‘deploy’) or events (like a relation to another service being added). Charm hooks represent the focal points where one defines what the charm actually needs to do. (Artur Tyloch, 2015)

For every interface declared by a charm, additional 4 “relation hooks” can be implemented, named after the interface: (Artur Tyloch, 2015)

- [name]-relation-joined
- [name]-relation-changed
- [name]-relation-departed
- [name]-relation-broken

2.7 Kamailio

Kamailio is an open source Session Initiation Protocol (SIP) Server released under General Public License (GPL), able to handle thousands of call setups per second, Kamailio can be used to build large platforms for VoIP and real-time communications – presence, web browsers and mobile applications with real-time communication (WebRTC), Instant messaging and other applications. Moreover, it can be easily used for scaling up SIP-to-PSTN gateways, Private Branch Exchange (PBX) systems or media servers. It can be configured to act as a SIP register, proxy or redirect server. (Kamailio)

2.8 Session Initiation Protocol

The Session Initiation Protocol (SIP) is an application-layer control (signaling) protocol for creating, modifying and terminating sessions with one or more participants. It can be used for voice, video, instant messaging, gaming and so on (MIT, 2005).

SIP invitations used to create sessions carry session descriptions that allow participants to agree on a set of compatible media types. SIP makes use of elements called proxy servers to help route requests to the user's current location, authenticate and authorize users for services, implement provider call-routing policies, and provide features to users. SIP also provides a registration function that allows users to upload their current locations for use by proxy servers. SIP runs on top of several different transport protocols (RFC 3261, 2002).

The following Figure 2-9 is showing the basic call flow for Session Initiation Protocol (MIT, 2005):

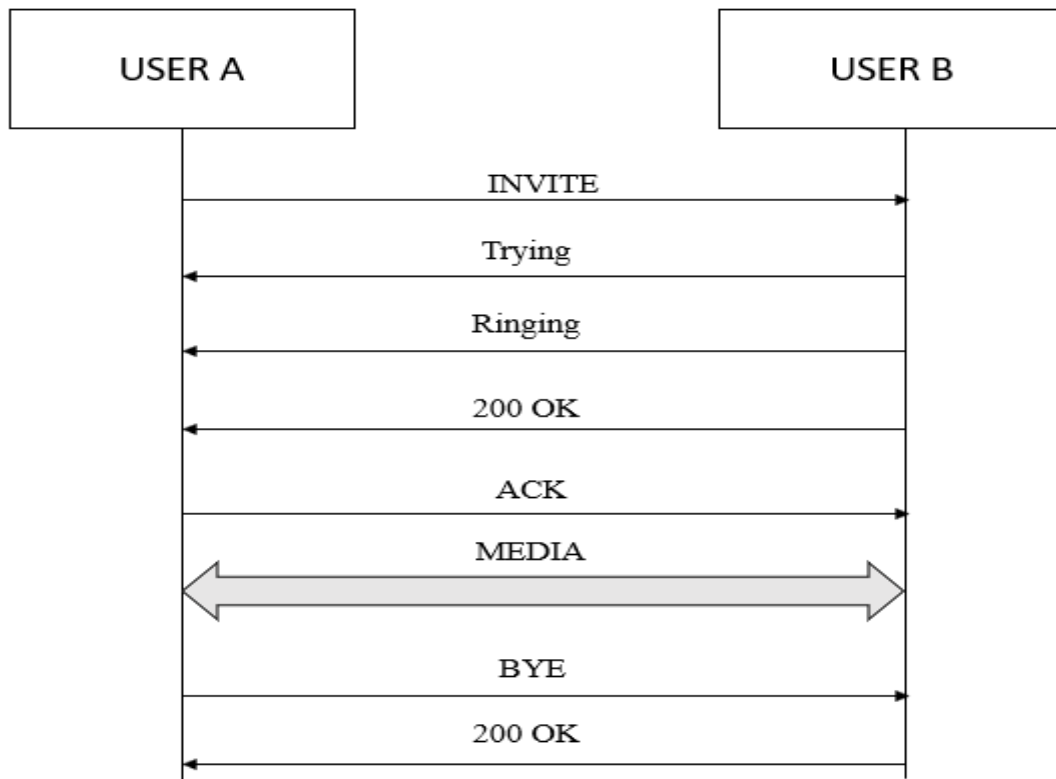


Figure 2-9 SIP call flow diagram (MIT, 2005)

2.9 VirtualBox

VirtualBox is a powerful virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for enterprise customers, it is also the only professional solution that is freely available as Open Source Software under the terms of the GNU General Public License (GPL) (VirtualBox, 2018).

It installs on existing Intel or AMD-based computers, whether they are running Windows, Mac, Linux or Solaris operating systems. Secondly, it extends the capabilities of existing computer so that it can run multiple operating systems (inside multiple virtual machines) at the same time. So, for example, run Windows and Linux on Mac, run Windows Server 2008 on Linux server, run Linux on Windows PC, and so on, all alongside existing applications. Anyone can install and run as many virtual machines as like – the only practical limits are disk space and memory. VirtualBox is deceptively simple yet also very powerful. It can run everywhere from small embedded systems or desktop class machines all the way up to datacenter deployments and even Cloud environments. (Oracle VM VirtualBox, 2017)

The techniques and features that VirtualBox provides are useful for several scenarios:

- **Running multiple operating systems simultaneously:** VirtualBox allows to run more than one operating system at a time. This way software written for one operating system on another (for example, Windows software on Linux or a Mac) without having to reboot to use it. Since can configure what kinds of “virtual” hardware should be presented to each such operating system, install an old operating system such as DOS or OS/2 even if real computer’s hardware is no longer supported by that operating system. (Oracle VM VirtualBox, 2017)

- **Easier software installations:** Software vendors can use virtual machines to ship entire software configurations. For example, installing a complete mail server solution on a real machine can be a tedious task. With VirtualBox, such a complex setup (then often called an “appliance”) can be packed into a virtual machine. Installing and running a mail server becomes as easy as importing such an appliance into VirtualBox. (Oracle VM VirtualBox, 2017)

- **Testing and disaster recovery:** Once installed, a virtual machine and its virtual hard disks can be considered a “container” that can be arbitrarily frozen, woken up, copied, backed up, and transported between hosts. On top of that, with the use of another VirtualBox feature called “snapshots”, one can save a particular state of a virtual machine and revert back to that state, if necessary. This way, one can freely experiment with a computing environment. If something goes wrong (e.g. after installing misbehaving software or infecting the guest with a virus), one can easily switch back to a previous snapshot and avoid the need of frequent backups and restores. Any number of snapshots can be created, allowing to travel back and forward in virtual machine time. Also, possible to delete snapshots while a VM is running to reclaim disk space. (Oracle VM VirtualBox, 2017)

- **Infrastructure consolidation:** Virtualization can significantly reduce hardware and electricity costs. Most of the time, computers today only use a fraction of their potential power and run with low average system loads. A lot of hardware resources as well as electricity is thereby wasted. So, instead of running many such physical computers that are only partially used, one can pack many virtual machines onto a few powerful hosts and balance the loads between them. (Oracle VM VirtualBox, 2017)

3 Requirements Analysis

In this project, all the requirements not only analyzed in theoretical aspects but also in practical ones. The main objective of this project is developing a charm for Kamailio server and managing Voice Over Internet Protocol (VoIP) service using MySQL charm and Kamailio charm from Juju. Beside the main objective, the functionality of Virtualized Network Function Manager (VNFM) will be evaluated by Juju.

3.1 General Objectives

In this project, a charm will be written for Kamailio server which will be developed in bash script. All packages for Kamailio should be implemented through this charm, such as installing Kamailio, changing Kamailio config file, creating Kamailio database and so on. Everything will be configured automatically when Kamailio will be deployed from charm in an application container.

In this work, a developed MySQL charm also will be deployed from charm store in an application container. A relation will be established between Kamailio application container and MySQL application container to fulfill the requirements for Kamailio server. At the end, communication will be built up between Kamailio subscribers. Last but not least, how Juju is fulfilling the functionality of VNFM will be described with the practical work

3.2 Clarifying the Requirements

In this student project, Juju will take part with LXD. Combination of Juju and LXD is an efficient way to deploy a software. Juju is working as a service modeling system, which also making relationship between different services and scale between them. Services are deploying in Juju model, which is focusing on machines for configuration management system. Controller will manage all the model associated with its node.

Charms are providing the services for application. To deploy and configure an application juju charm contains the all necessary instructions. Packages are included as operational code in charms which is implemented with installation process.

Juju manages the relation between multiple applications. Applications will be created a relation between them through interfaces and provided the expected services.

3.3 Time frames

In the time frames section, the timetable including the start and end time and the milestones of work are presented.

Milestones:

05.09.2018	Starting date
05.09.2018 – 19.09.2018	Analysis of the Requirements
20.09.2018	Submission of Requirements analysis
20.09.2018 – 30.09.2018	Literature review and learning theoretical background
01.10.2018 – 10.10.2018	VM installation, LXC, LXD and Juju implementation

11.10.2018 – 20.10.2018	Analyzing Juju performance as a VNFM and Documentation writing
21.10.2018 – 07.11.2018	Writing charm for Kamailio
08.11.2018 – 09.11.2018	Deploy MySQL and Kamailio application and add relation between them
09.11.2018 – 12.11.2018	Call set up between two users
02.11.2018 – 14.11.2018	Evaluate performance of Juju as a VoIP Service
15.11.2018 – 24.11.2018	Documentation writing
25.11.2018 - 03.12.2018	Editing report as suggested by supervisor
04.12.2018	Final report Submission to the examination office

3.4 Target State

The target state consists of total set up is shown in the Figure 3-1. Oracle VirtualBox is used for virtualization platform. TK-Lab network will be used for core network, as machine connected with this.

Using Juju Kamailio server and MySQL database applications will be deployed on the local LXD environment in two containers. Juju will make a relation between them. Two subscribers will be registered in the Kamailio database. Established a call between two users via Kamailio server.

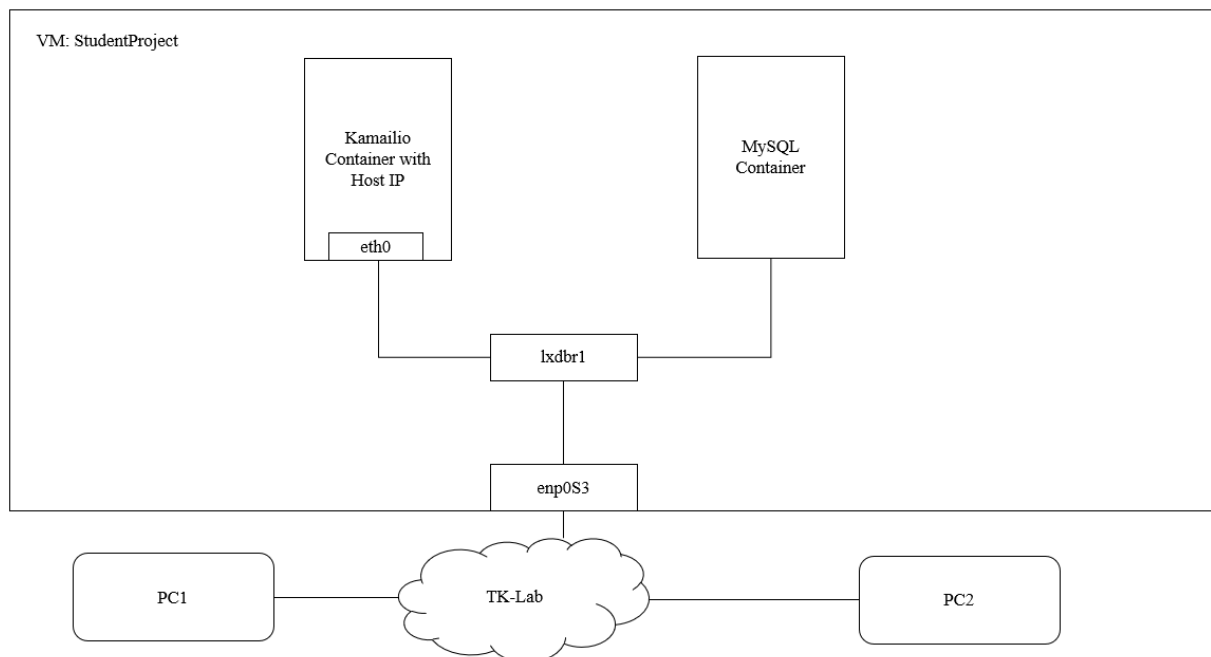


Figure 3-1 Juju implementation environment

3.5 Software Requirements

- Virtual Box
- Ubuntu Operating System
- LXC

-
- LXD
 - Juju
 - YAML
 - Kmailio
 - Mysql
 - PhonerLite
 - Wireshark

4 Realisation

In this chapter will be presented the implementation part of this project work. Following are the important steps of the implementation of this student project.

1. Prepare the Environment in VirtualBox
2. Install LXC, LXD and Juju
3. Deploy application from Juju
4. Making Charm for Kamilio server
5. Deploy kamilio server from Juju

4.1 Prepare the Environment in VirtualBox

For this project, a virtual machine is created using VirtualBox software on Windows host system. Based on the operating system, correct version of the VirtualBox has to be downloaded from the official website of Oracle. After the successful download, VirtualBox windows installer is started and the software is installed with the instructions from the installer. [Oracle VM VirtualBox, 2017]



Figure 4-1 VirtualBox Installation Prompt

For creating a virtual machine in VirtualBox following steps must followed-

Start creating a virtual machine in VirtualBox

1. After the VirtualBox executed file is started below window appears. In the window, to create a virtual machine must be clicked on New button.

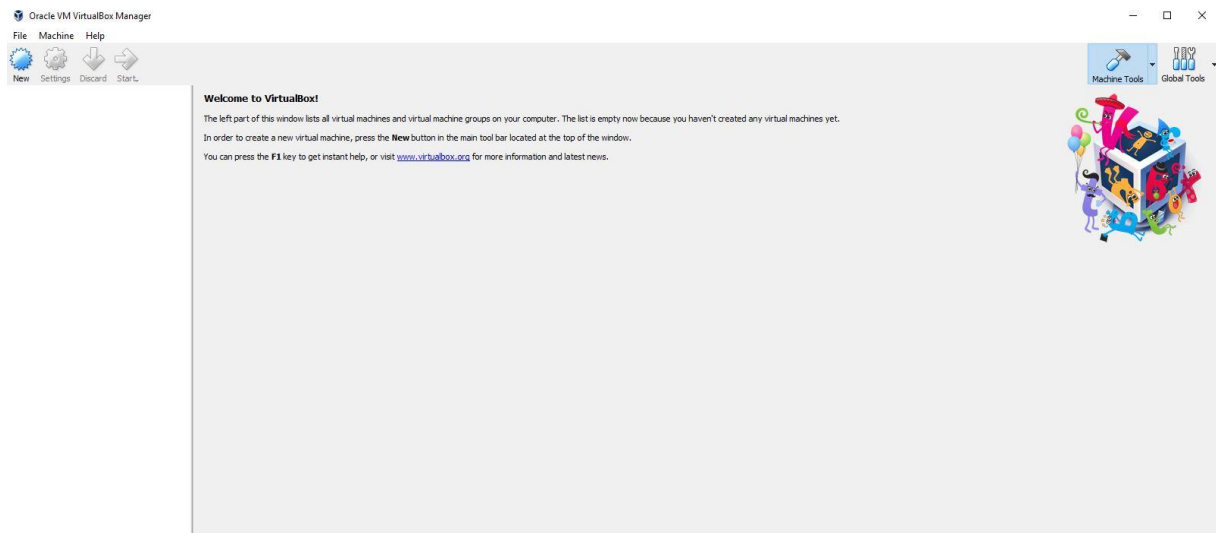


Figure 4-2 Creation a virtual machine

2. The below window shows the name of the virtual machine. Type and Version can be selected based on the requirement. For this student project, Linux is selected as the Type and version is Ubuntu. The Figure 4-3 explained precisely:

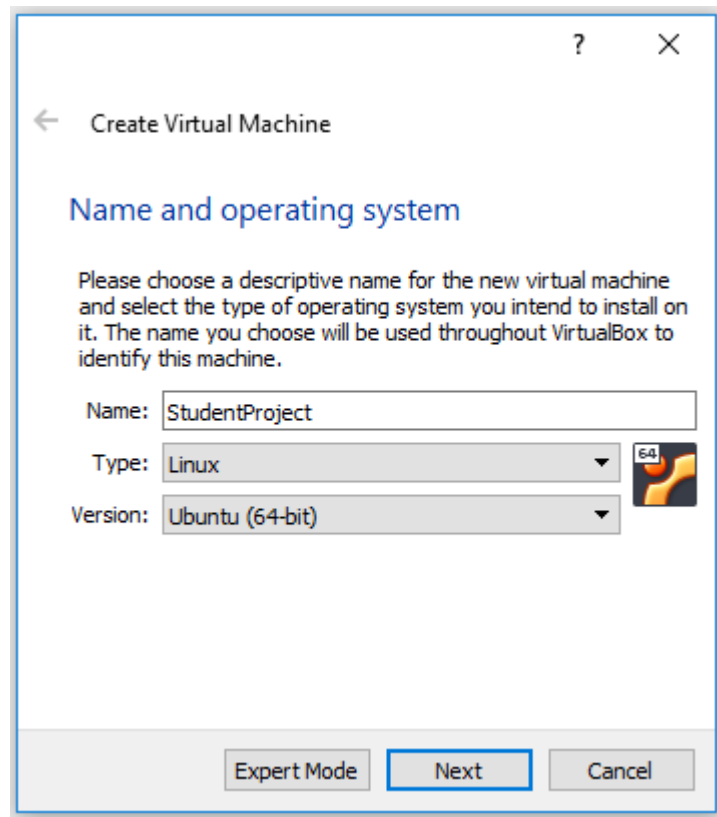


Figure 4-3 Name of the project with Operating System

3. Memory for the virtual machine has to be selected. It is an important part for virtual machine. For this project, used 4 GB memory. Figure 4-4 is showing the total memory:

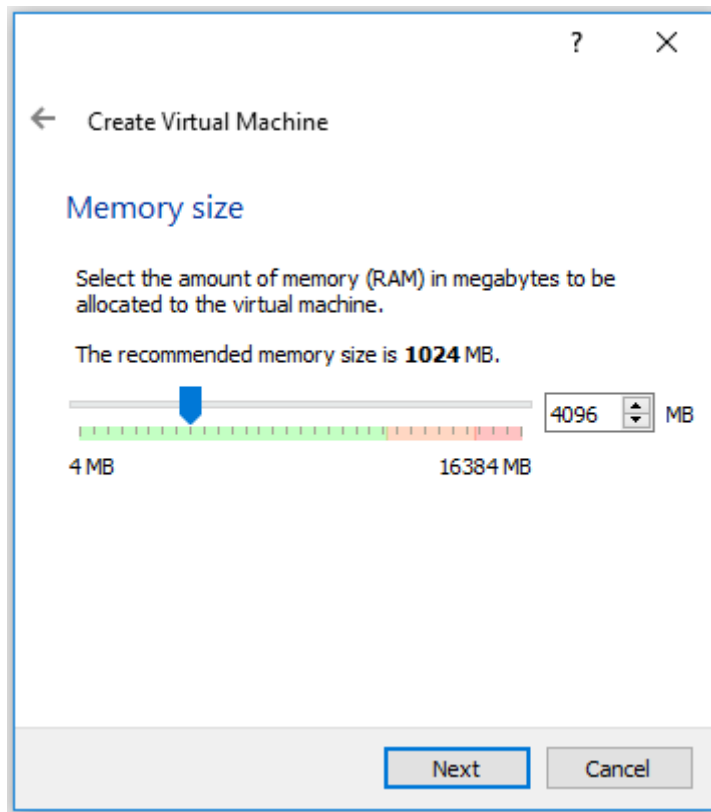


Figure 4-4 Memory Allocation for the virtual machine

- 4 Virtual hard disk space for the virtual machine must be determined based on the requirement of the project.

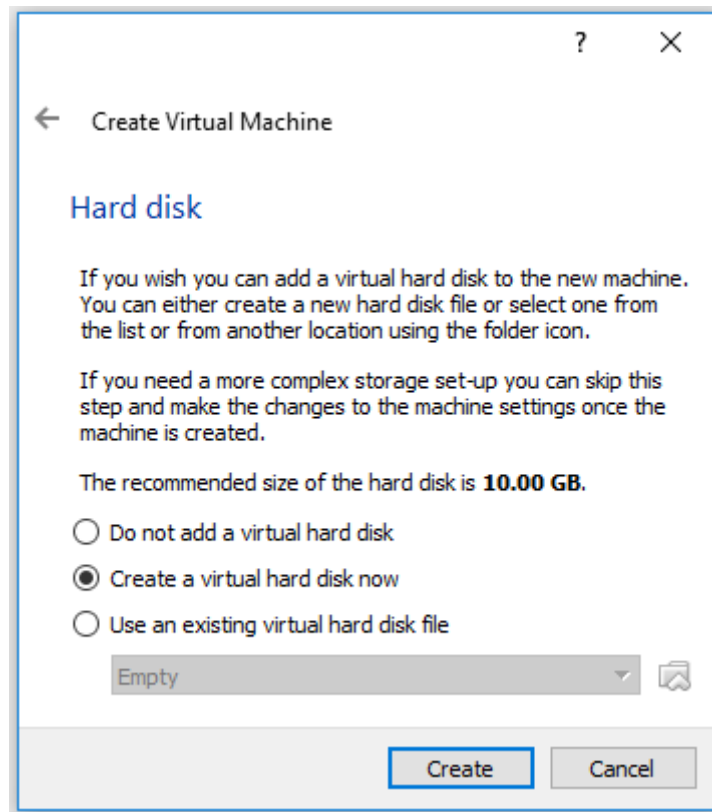


Figure 4-5 Selecting Hard Disk option

5. In this step, it will ask for the type of virtual disk need to be sued. For this project, VirtualBox Disk Image (VDI) has been used.

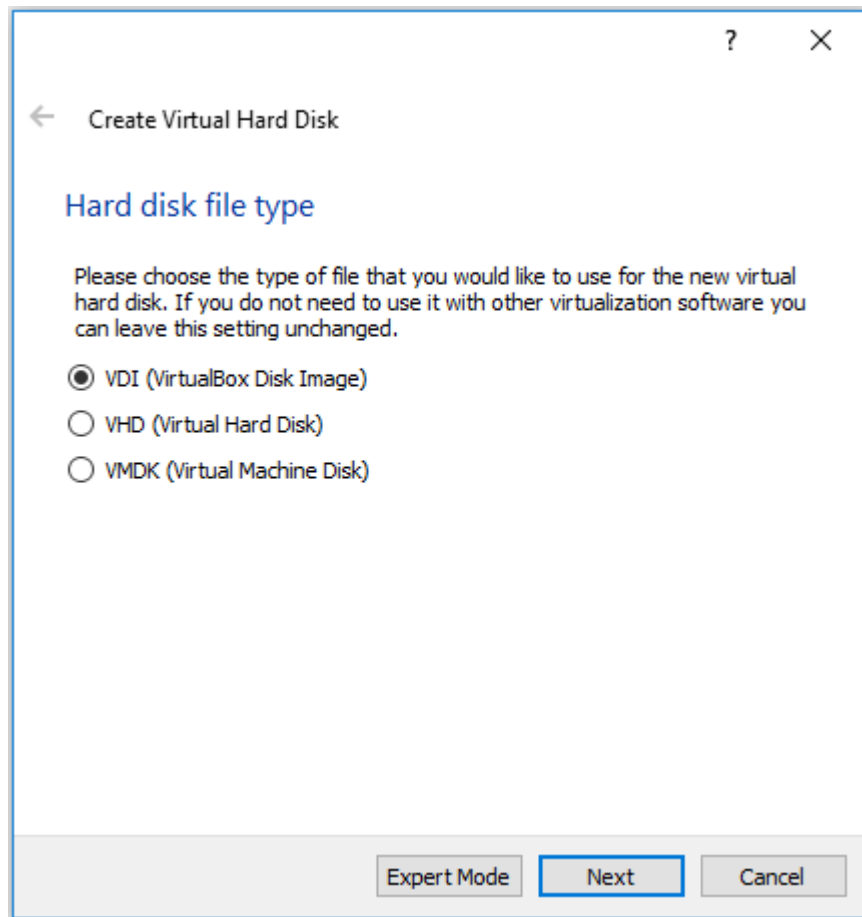


Figure 4-6 Defining Hard Disk File type

6. Then it will ask for the new virtual disk how it will expand memory when it is used. For this student project 'Dynamically allocated' option is used.

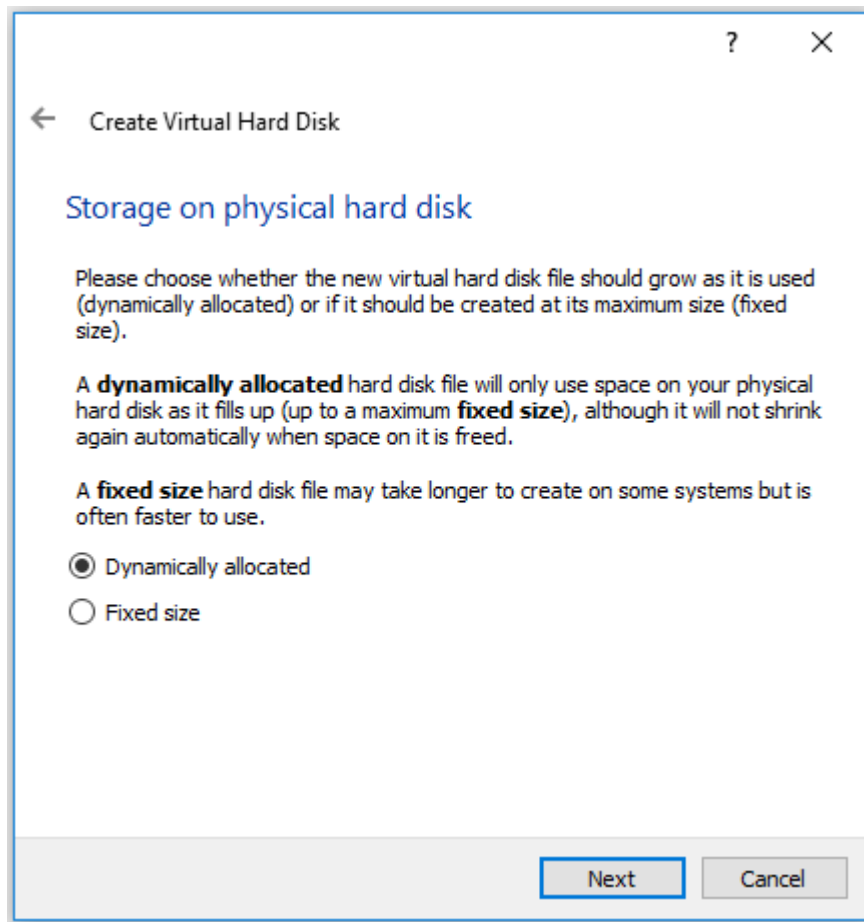


Figure 4-7 Selecting physical hard disk type for virtual machine

7. After the creation of virtual machine, setup ubuntu for this project in virtual machine. For this project used Ubuntu 18.04.1 LTS. Below Figure 4-8 shows the final stage for ubuntu setup.

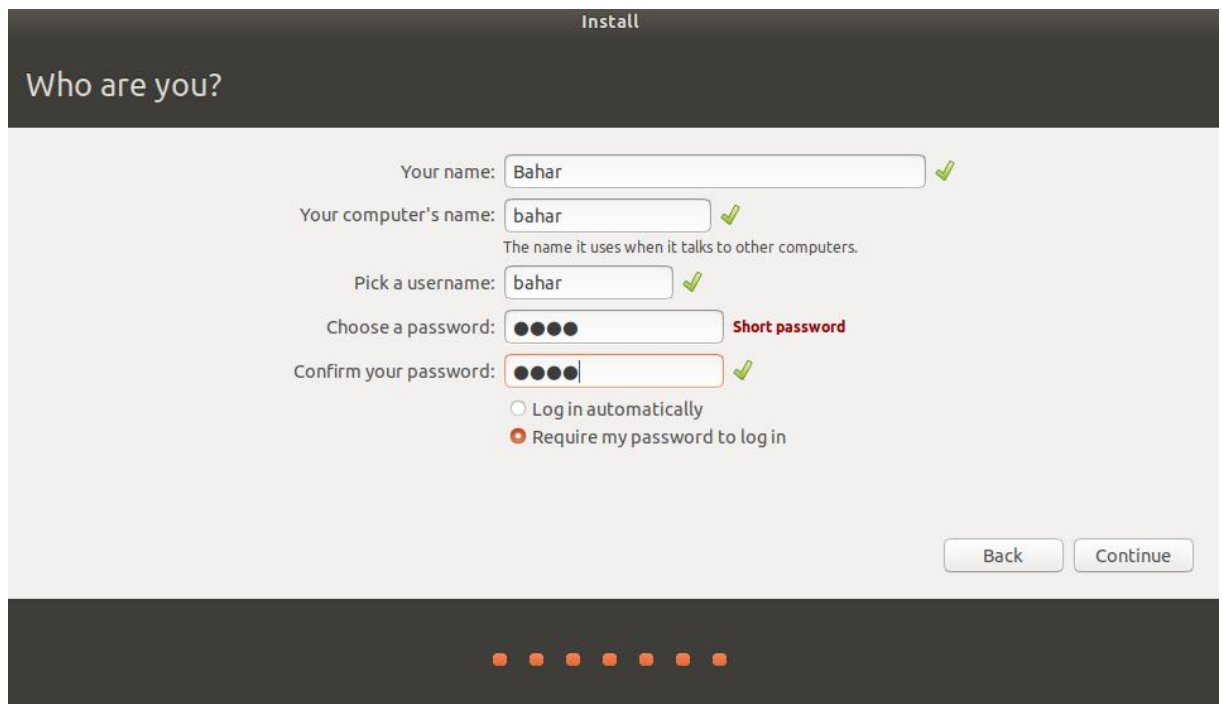


Figure 4-8 Virtual machine Operating System setup

4.2 Installation of LXC, LXD and Juju

Before installing LXC, ubuntu will be updated and upgraded by the below command:

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

Every Linux distribution has recent version of LXC is available. On such Ubuntu system, LXC is as simple as:

```
$ sudo apt-get install lxc
```

After this command, ubuntu has all lxc commands available.

For installing the LXD following command should be used

```
$ sudo apt install lxd
```

To install juju following command should be used

```
$ sudo apt install juju
```

Command for checking the version of Juju:

```
$ sudo juju version
```

The following output should appear in Figure 4-9:

```
bahar@bahar:~$ sudo juju version
2.4.6-bionic-amd64
```

Figure 4-9 Juju version

4.3 Juju Command

In this chapter, all the commands which is used for Juju will be described.

4.3.1 Controller Management

For deploying any applications, a controller is needed. Controller is the heart of Juju The controller will manage the state and events for model that host the applications.

To create a controller following command has been used:

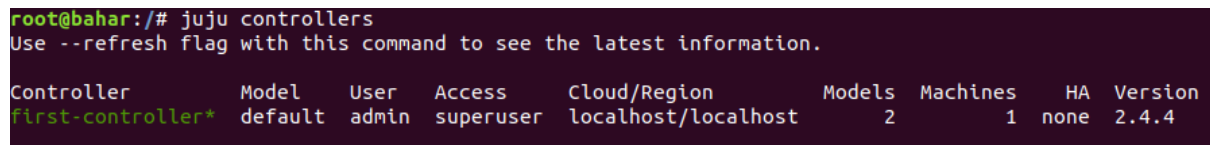
```
$ sudo juju bootstrap localhost controller-name
```

Here, controller-name is the controller name which is created with the bootstrap command. The built-in LXD cloud is known as 'localhost'.

To check the controller list following command, need to run:

```
$ sudo juju controllers
```

It will return a list of the controllers known to juju. Below figure 4-10 shows the controller list with name:



```
root@bahar:/# juju controllers
Use --refresh flag with this command to see the latest information.

Controller      Model  User  Access  Cloud/Region  Models  Machines  HA  Version
first-controller* default admin superuser localhost/localhost 2      1    none 2.4.4
```

Figure 4-10 Juju controller list

To check the details about a controller, execute the following command:

```
$ sudo juju show-controller localhost-localhost
```

The following output should be seen Figure 4-11. where uuid is representing the universal identifier with octet. api-endpoints shows the IP address with port number. controller-machines showing the machine number and instance id. For this controller two models are available and figure showing which model has how many machines by machine-count.

```

localhost-localhost:
details:
  uuid: c778ab01-d5da-4159-875b-486d67fba898
  api-endpoints: ['10.93.63.81:17070']
  ca-cert: |
    -----BEGIN CERTIFICATE-----
    MIIDrDCCApSgAwIBAgIUMCjnLT449ZF2jWYHTNzPg+ghHnYwDQYJKoZIhvcNAQEL
    BQAwbjENMA5GA1UEChMEanVqdTEuMCwGA1UEAwlanVqdS1nZW5lcmF0ZWQgQ0Eg
    Zm9yIG1vZGVsICJqdWp1LWNhIjEtMCsGA1UEBRMkYmJiODI1NjUtNWM3NC00MDlk
    LTg2NWEtZmFiZTM0MTc1YTY1MB4XDTE4MTAyMjE4MDcyOVVoXDTI4MTAyOTE4MDcy
    OVowbjENMA5GA1UEChMEanVqdTEuMCwGA1UEAwlanVqdS1nZW5lcmF0ZWQgQ0Eg
    Zm9yIG1vZGVsICJqdWp1LWNhIjEtMCsGA1UEBRMkYmJiODI1NjUtNWM3NC00MDlk
    LTg2NWEtZmFiZTM0MTc1YTY1MIIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKC
    AQEAwTqVTPQy0FGkqxb2fakx88KraqdZhEYVkjneCk0NPpfDy5g70PA7cdZUvsB
    ih4fzKKyZzyJe+FNhd5ytTKLFzG4dvspoC4HVqGqRFSaITjkiZSkHPv5HSShmZA
    PkEgEQqyRIr+eXfIKs5r7cdmEf7zpQLtUkKBFXigXIwfGt/XQrzlwTFGgdbvZQF0
    pa/Ii/RURwjTQ3I+cdSqRbJj0SDRHYfYhRA8z3k5KwWs2cfr2IqpltdfsIZ5WqQC
    8VeUZkpHWuT/NR4j1kDPpijr7YnMR6+NuGukQLFMX8ACsMQX96QMjUe8UZ4704Qp
    OmCWS3cxGLXWCBTNkvLUSUVQdwIDAQABo0IwQDAOBgNVHQ8BAf8EBAMCAQwDwYD
    VR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQU/axavlrLRpZEw9VN3i6+a6erd10wDQYJ
    KoZIhvcNAQELBQADggEBAD6BfrPR55tU6d0hiUTHlavNKXsIl0AvfV0vMI5eoc0w
    MmauSuF8NNGQ6D/DTMQEJu1nDMSXXRp53nSKfU823F4mw07CQcppV+0oL5AkCuOK
    E9aLZnbjz3wwXz0nYisvs3WwGLB5mmF87sUfycsNZjJSuKscG8oYGL4F1PRyPhKv
    ZByrsHwvaPyL9h3iOpyDpNoToR3EkEWVme9JjRYKbSooWERzLgoFS3I/jE+p18ig
    FBQUa4m1esd7k0ibh07PqzPXpd3IknF812bF5iRccLUh5pxHbrrt2hbz0Pzzfppi
    TOKp6wn+5jx9ax1GqDj3a689bY2EnIvpYy6zPrPEaSE=
    -----END CERTIFICATE-----
  cloud: localhost
  region: localhost
  agent-version: 2.4.4
  controller-machines:
    "0":
      instance-id: juju-c94f97-0
  models:
    controller:
      uuid: 31764082-7fc3-42ca-8d99-3258efc94f97
      machine-count: 1
    default:
      uuid: 5419cf4e-2347-45fd-8d98-fcc1590349f2
      machine-count: 2
  current-model: admin/default
  account:
    user: admin
    access: superuser

```

Figure 4-11 Controller details

To destroy the controller following command will be run:

```
$ sudo juju destroy-controller localhost-localhost
```

Sometimes controller is not accessible from command line because of any error in the controller. That situation destroy-controller command will be not worked to destroy or delete a controller. If any controller will be not accessible from CLI, then following will be executed to delete:

```
$ sudo juju kill-controller localhost-localhost
```

To switch one controller to another controller with the following command:

```
$ sudo switch
```

4.3.2 Model Management

To add a model to a controller, use the following command:

```
$ sudo juju add-model model-name
```

Here, model-name is the name of the model given by user.

Check the model list for a selected controller with the following command:

```
$ sudo juju models
```

The following output should be seen in Figure 4-12:

```
root@bahar:~# juju models
Controller: localhost-localhost

Model      Cloud/Region      Status      Machines  Access  Last connection
controller localhost/localhost available    1  admin  just now
default*   localhost/localhost available    2  admin  3 hours ago
```

Figure 4-12 Juju model list

To select one model to another model, execute the following command:

```
$ sudo switch localhost-localhost:admin/controller
```

```
root@bahar:~# juju models
Controller: localhost-localhost

Model      Cloud/Region      Status      Machines  Access  Last connection
controller* localhost/localhost available    1  admin  just now
default     localhost/localhost available    2  admin  3 hours ago
```

Figure 4-13 switch model

From Figure 4-13: Controller model is showing green after the `switch` command executed. Now the active model is controller model. If any command is run through the localhost-localhost controller, controller model will be affected.

Delete a model with the following command:

```
$ sudo juju destroy-model default
```

The following command shows the currently active controller, model and user

```
$ sudo juju whoami
```

The following output should be visible in Figure 4-14:

```
root@bahar:~# juju whoami
Controller: localhost-localhost
Model:      default
User:       admin
```

Figure 4-14 Juju whoami

4.4 Kamailio Charm

In this chapter Kamailio charm will be presented and explained. Multiple scripts have been created for writing script. The bash scripts are installing the required software and implementing the required bash command that will be described in detail. Each script has different purpose and will be executed when their functionality is needed.

4.4.1 Create Charm Template

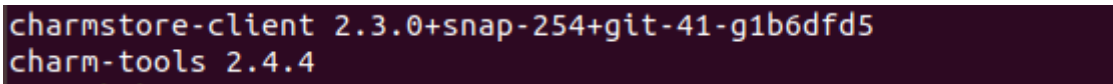
The charm Tools package is containing all the commands to maintain charm. This tools also known as charm management tools. The author can create a charm by using this charm. To install the charm tools, run the following command:

```
$ sudo snap install charm
```

Test the correct installation by the following command:

```
$ sudo charm version
```

The following output should appear:



```
charmstore-client 2.3.0+snap-254+git-41-g1b6dfd5
charm-tools 2.4.4
```

Figure 4-15 Version of Charm Tools

Create new folder and navigated into the created directory with the following command:

```
$ mkdir charms/precise
```

```
$ cd charms/precise
```

Using charm Tools now it's very easy to create a charm which providing the recommended structure for charm and includes all the right pieces with right place. Charm template can be created with the author specific choices language. To create a template for charm, run the following command:

```
$ sudo charm create -t bash kamailio-new
```

Here, `-t bash` option is providing the information about the charm will be written in bash script.

Navigated the kamailio-new directory and run the following command:

```
$ tree
```

The following output should be seen:

```

root@bahar:~/charms/precise/kamailio-new# tree
.
├── config.yaml
├── hooks
│   ├── config-changed
│   ├── install
│   ├── relation-name-relation-broken
│   ├── relation-name-relation-changed
│   ├── relation-name-relation-departed
│   ├── relation-name-relation-joined
│   ├── start
│   ├── stop
│   └── upgrade-charm
├── icon.svg
├── metadata.yaml
├── README.ex
└── revision

1 directory, 14 files

```

Figure 4-16 Kamailio charm template

From the figure, `config.yaml` file will be set, if need any change in configuration in the charm. If there have any change in `config.yaml`, it can be get from `config-changed` hooks. `README` file is important for making the charm public. It is a plaintext file. This fill will be described about implemented charm and how it will relate to others charm.

`metadata.yaml` is an important file for charm. From this file Juju can read the name of charm, understand what type of charm it is and define which relation is required or provided by this application.

`install` hooks need to install all the software for the charm and provide the configuration for any other application.

`relation-name` hook will be described about the relation with others charm. This name will be changed with the name which is given in `metadata.yaml` file.

4.4.2 Writing charm

This section will be described about the script written for Kamailio charm. For Kamailio charm following files will be created in YAML:

- `metadata.yaml`
- `config.yaml`

and following hooks will be written in bash script:

- `config-changed`
- `install`
- `relation-name-relation-changed`
- `relation-name-relation-departed`

Metadata File:

Set the name for charm by using the name parameter. After deploying the application, the given name is used to create the name for application. The following name pattern is used for Kamailio:

```
name: kmailio
```

In the following steps, relation name “database” is given with relation type “requires” which will be used to rename the relation hooks and checking for MySQL charm which is provided through interface. Interface `mysql-root` is used for root user of MySQL. Indentation is important as it is providing the level for every step.

```
requires:
```

```
    database:
```

```
        interface: mysql-root
```

Config File:

This file will be provided the specific configuration options to support an application when it is deploying. If charm need any change in application that will be provided through this file. Kmailio application will be deployed with host-address. The following pattern will be used with indentation:

```
options:
```

```
    host-addresses:
```

```
        type: string
```

```
    default:
```

```
        description: "A short description of the configuration option"
```

Install Hook:

Install is one of the main hooks for Kmailio application. All the software which will be used for this application will be introduced with this hook. This hook will be used for installing and fetching the Kmailio code. The following code will be used for this hook:

```
#!/bin/bash

set -e

juju-log "KAM-NEW: install"

apt update

apt install grep

apt install -y kmailio kmailio-mysql-modules

juju-log "KAM-NEW: kmailio installed"

status-set maintenance "installing kmailio"

status-set blocked "waiting for database"
```

In the first line used a hash-bang line which is indicating that it's a bash file. `set -e` a command used for if any command is wrong then script will be stopped and raise an error. This line is important because of smooth work and from this command it's very easy to get the information about the error in application.

`-y` option is used with installation command because it will provide the yes answer of any question when they are installing the packages. So, when Kmailio package will be installed automatically it will give the answer for installation question with yes. Kmailio and MySQL packages will be installed from this hook.

`juju-log` command used for checking the message in juju log which is useful for testing and debugging. After every successful step this command will be provided the message.

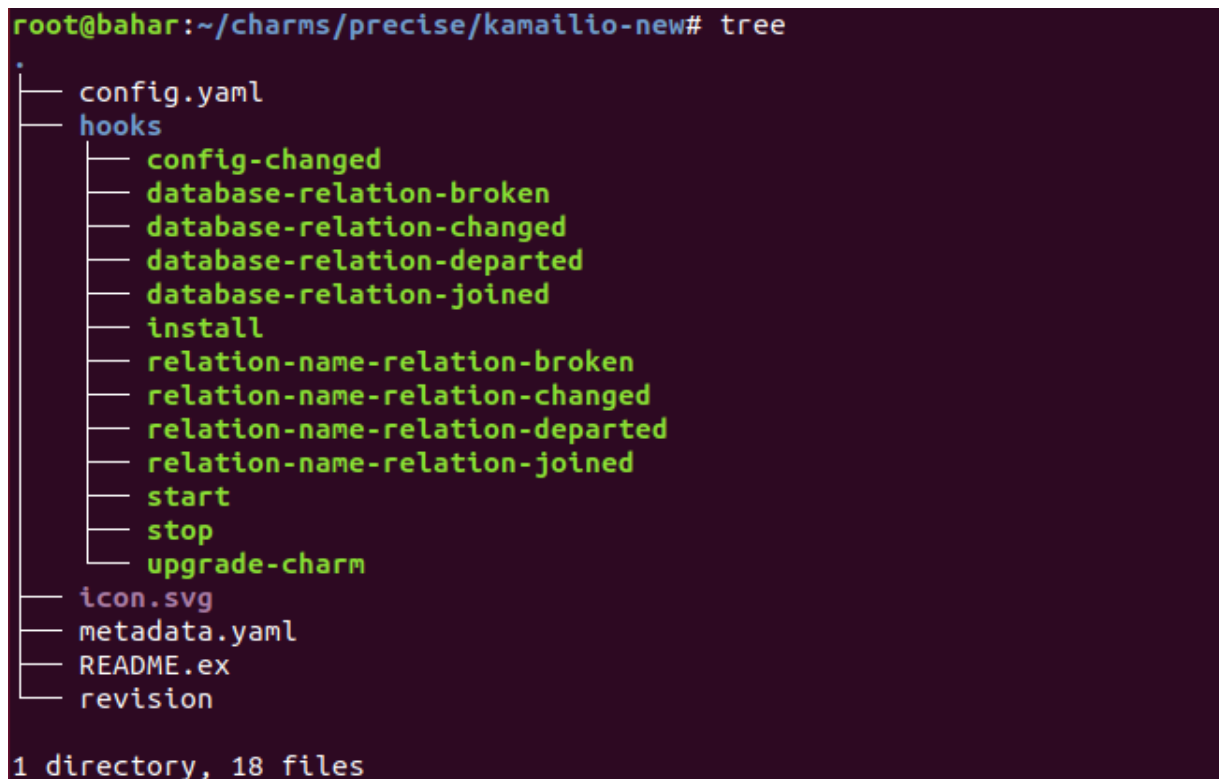
`status-set` command is used to update the status and message displayed by `juju status` command. It will provide the information what going on inside the environment when deploying an application or after deployed an application. `maintenance` status means this unit is not providing any services and `juju` agent will set the `maintenance` status when it is installing the packages. `blocked` status is providing the information about the application need a database connection. This status command is given by Juju and every status has a define color.

Rename Relation Hook:

From the discussion of the `metadata.yaml`, In the `metadata.yaml` file set a connection called 'database', so relation-name is database in hook. To change the name of some relation hooks file, following commands will be executed:

```
$ sudo cp relation-name-relation-changed database-relation-changed
$ sudo cp relation-name-relation-departed database-relation-departed
$ sudo cp relation-name-relation-joined database-relation-joined
$ sudo cp relation-name-relation-broken database-relation-broken
```

Following output should appear with `tree` command:



```
root@bahar:~/charms/precise/kamailio-new# tree
.
├── config.yaml
├── hooks
│   ├── config-changed
│   ├── database-relation-broken
│   ├── database-relation-changed
│   ├── database-relation-departed
│   ├── database-relation-joined
│   ├── install
│   ├── relation-name-relation-broken
│   ├── relation-name-relation-changed
│   ├── relation-name-relation-departed
│   ├── relation-name-relation-joined
│   ├── start
│   ├── stop
│   └── upgrade-charm
├── icon.svg
├── metadata.yaml
├── README.ex
└── revision

1 directory, 18 files
```

Figure 4-17 New charm template

Database-relation-changed Hook:

`database-relation-changed` hook fetches the named values from the corresponding hook on the charm are connecting to. This hook will use for all kind of information need to change in `kamailio.cfg` and `kamctlrc` file. This hook will be worked when a relation will be established between Kamailio and MySQL applications. To get the information about user and database by using `relation-get` command. To get the information for database and Kamailio host following command will be executed:

```
db_user=$(relation-get user)
db_db=$(relation-get database)
db_pass=$(relation-get password)
db_host=$(relation-get private-address)
```

Kamailio container IP will be get by the following command:

```
kam_host=$(hostname --ip-address)
```

If valid database is not connected with Kamailio charm or name is not correct, then it will be exit from this hook and give a message. The following code is used for this purpose:

```
if [-z "$db_db"]; then
    juju-log "No database information sent "
    exit 0
fi
```

In install hook `grep` command is also used to install `grep` packages. The main function of this command to search a text from given file. Following `if-else` statement is used for inserting data in `kamailio.cfg` file. If the command is available it will exit from `if` statement and go for next line otherwise it will insert the value in 2nd, 3rd and 4th line.

```
if grep -q '#!define WITH_MYSQL' /etc/kamailio/kamailio.cfg
then
    exit 0
else
    sed -i '2i#!define WITH_MYSQL' /etc/kamailio/kamailio.cfg
    sed -i '3i#!define WITH_AUTH' /etc/kamailio/kamailio.cfg
    sed -i '4i#!define WITH_USRLOCDB' /etc/kamailio/kamailio.cfg
fi
```

Kamailio database will be installed with the following parameter:

```
sed -i "s/#\s*INSTALL_EXTRA.*\s*INSTALL_EXTRA_TABLES=yes/g"
/etc/kamailio/kamctlrc

sed -i "s/#\s*INSTALL_PRESENC.*\s*INSTALL_PRESENCE_TABLES=yes/g"
/etc/kamailio/kamctlrc

sed -i "s/#\s*INSTALL_DBUID.*\s*INSTALL_DBUID_TABLES=no/g"
/etc/kamailio/kamctlrc
```

To change the access permission for two files following command will be executed:

```
chmod -R 777 /etc/kamailio/kamailio.cfg
chmod -R 777 /etc/kamailio/kamctlrc
```

Kamailio database will be created by the following command:

```
kamdbctl create
```


After all the completed all the configuration the Kamilio application will be reboot by the following command:

```
reboot
```

Database-relation-departed Hook:

This hook will be deleted all the relations between Kamilio and MySQL when Kamilio application will be removed the relation with MySQL. Following command will be run with this bash script file:

```
kamdbctl drop
```

This command will delete the kamilio database from MySQL database.

Config-changed Hook:

This bash file is used for updating configuration value if there have any configuration change. This file is related with `config.yaml` file. Execute the following command:

```
host_ip=$(config-get host-addresses)
```

`config-get` will provide the change of configuration in configuration file.

add the entry to the `kamilio.cfg` file:

```
sed -i "5iadvertised_address=${host_ip}" /etc/kamilio/kamilio.cfg
```

```
sed -i "6ialias=${host_ip}" /etc/kamilio/kamilio.cfg
```

5 Demonstration

In this chapter the demonstration part will be presented and explained for this project. The demonstration part consists of deploying MySQL and Kamailio charm in different container and add relation between them to get the VoIP services. Call has been established between two users and traffic will be analyzed on the access network by using SIP.

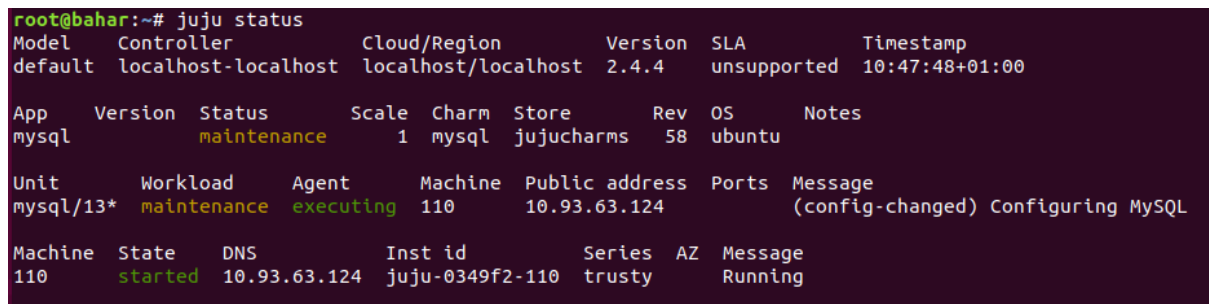
5.1 Deploy MySQL

To deploy MySQL charm from charm store following command will be executed:

```
$ sudo juju deploy cs:trusty mysql
```

When Juju is deploying MySQL charm, the status will be showed by the following command:

```
$ sudo juju status
```



```

root@bahar:~# juju status
Model      Controller      Cloud/Region      Version      SLA          Timestamp
default    localhost-localhost    localhost/localhost    2.4.4        unsupported    10:47:48+01:00

App      Version      Status      Scale  Charm  Store      Rev  OS    Notes
mysql    5.5.62      maintenance  1      mysql  jujucharms  58   ubuntu

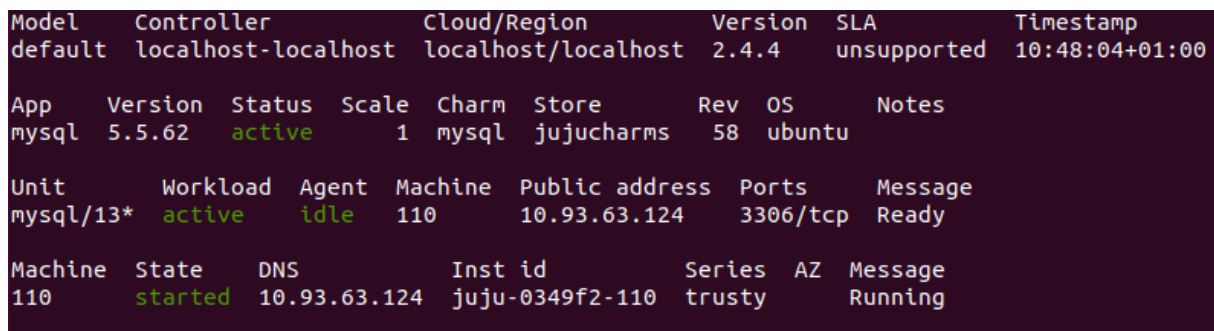
Unit      Workload      Agent      Machine  Public address  Ports  Message
mysql/13*  maintenance  executing  110      10.93.63.124    3306/tcp  (config-changed) Configuring MySQL

Machine   State      DNS          Inst id      Series  AZ  Message
110       started    10.93.63.124  juju-0349f2-110  trusty   Running

```

Figure 5-1 MySQL status during the deployment process

Figure 5-1 is showing the status of MySQL during the deploying process. Where it is showing the maintenance status during the deployment.



```

Model      Controller      Cloud/Region      Version      SLA          Timestamp
default    localhost-localhost    localhost/localhost    2.4.4        unsupported    10:48:04+01:00

App      Version      Status      Scale  Charm  Store      Rev  OS    Notes
mysql    5.5.62      active      1      mysql  jujucharms  58   ubuntu

Unit      Workload      Agent      Machine  Public address  Ports  Message
mysql/13*  active        idle       110      10.93.63.124    3306/tcp  Ready

Machine   State      DNS          Inst id      Series  AZ  Message
110       started    10.93.63.124  juju-0349f2-110  trusty   Running

```

Figure 5-2 MySQL status after deployment

Figure 5-2 is showing the status of MySQL after successful deployment from charm store. The main difference between two pictures is one is showing maintenance during the deployment another one is showing active after deployment. Active status providing the successful deployment of MySQL charm information.

5.2 Deploy Kmailio

To deploy the Kmailio charm from bash script navigate in to the following directory by the following command:

```
cd /charms/precise
```

Execute the following command for deploying Kmailio:

```
$ sudo juju deploy ./kmailio-new --config host-addresses="192.168.50.79"
```

Kmailio will be deployed with host IP address. After executing the command following output should be seen:

```
root@bahar:~/charms/precise# juju deploy ./kmailio-new --config host-addresses="192.168.50.79"
Deploying charm "local:bionic/kmailio-90".
```

Figure 5-3 Deploy Kmailio

During the deploying process run the following command for Kmailio application status:

```
$ sudo juju status
```

Model	Controller		Cloud/Region		Version	SLA	Timestamp	
default	localhost-localhost		localhost/localhost		2.4.4	unsupported	10:56:19+01:00	
App	Version	Status	Scale	Charm	Store	Rev	OS	Notes
kamailio		maintenance	1	kamailio	local	90	ubuntu	
mysql	5.5.62	active	1	mysql	jujucharms	58	ubuntu	
Unit	Workload	Agent	Machine	Public address		Ports	Message	
kamailio/70*	maintenance	executing	111	10.93.63.8			(install) installing charm software	
mysql/13*	active	idle	110	10.93.63.124		3306/tcp	Ready	
Machine	State	DNS	Inst id	Series	AZ	Message		
110	started	10.93.63.124	juju-0349f2-110	trusty		Running		
111	started	10.93.63.8	juju-0349f2-111	bionic		Running		

Figure 5-4 Kmailio status during deployment process

Figure 5-4 is showing the status during the installing process of Kmailio charm.

After installing all the software from install hook, Kmailio status will be given bellow picture:

Model	Controller		Cloud/Region		Version	SLA	Timestamp	
default	localhost-localhost		localhost/localhost		2.4.4	unsupported	10:56:33+01:00	
App	Version	Status	Scale	Charm	Store	Rev	OS	Notes
kamailio		blocked	1	kamailio	local	90	ubuntu	
mysql	5.5.62	active	1	mysql	jujucharms	58	ubuntu	
Unit	Workload	Agent	Machine	Public address	Ports	Message		
kamailio/70*	blocked	idle	111	10.93.63.8		waiting for database		
mysql/13*	active	idle	110	10.93.63.124	3306/tcp	Ready		
Machine	State	DNS	Inst id	Series	AZ	Message		
110	started	10.93.63.124	juju-0349f2-110	trusty		Running		
111	started	10.93.63.8	juju-0349f2-111	bionic		Running		

Figure 5-5 Kmailio status after deployment

Form the Figure 5-5: Kmailio status is showing blocked because it's waiting for database. Which will be activated after making a relation with MySQL charm.

To check the installation log for Kmailio application with the following command:

```
$ sudo juju debug-log
```

```
unit-kamailio-70: 10:56:19 INFO unit.kamailio/70.juju-log KAM-NEW: kamailio installed
unit-kamailio-70: 10:56:19 INFO juju.worker.uniter.operation ran "install" hook
unit-kamailio-70: 10:56:19 INFO juju.worker.uniter found queued "leader-elected" hook
unit-kamailio-70: 10:56:19 INFO juju.worker.uniter.operation skipped "leader-elected" hook (missing)
unit-kamailio-70: 10:56:19 INFO unit.kamailio/70.juju-log KAM-NEW: config-changed!
unit-kamailio-70: 10:56:19 INFO unit.kamailio/70.juju-log Received config: ip-addresses = 192.168.50.79
unit-kamailio-70: 10:56:19 DEBUG unit.kamailio/70.config-changed Listening on
unit-kamailio-70: 10:56:19 DEBUG unit.kamailio/70.config-changed          udp: 127.0.0.1:5060
unit-kamailio-70: 10:56:19 DEBUG unit.kamailio/70.config-changed          udp: 10.93.63.8:5060
unit-kamailio-70: 10:56:19 DEBUG unit.kamailio/70.config-changed          tcp: 127.0.0.1:5060
unit-kamailio-70: 10:56:19 DEBUG unit.kamailio/70.config-changed          tcp: 10.93.63.8:5060
unit-kamailio-70: 10:56:19 DEBUG unit.kamailio/70.config-changed Aliases:
unit-kamailio-70: 10:56:19 DEBUG unit.kamailio/70.config-changed          tcp: juju-0349f2-111.lxd:5060
unit-kamailio-70: 10:56:19 DEBUG unit.kamailio/70.config-changed          tcp: localhost:5060
unit-kamailio-70: 10:56:19 DEBUG unit.kamailio/70.config-changed          udp: juju-0349f2-111.lxd:5060
unit-kamailio-70: 10:56:19 DEBUG unit.kamailio/70.config-changed          udp: localhost:5060
unit-kamailio-70: 10:56:19 DEBUG unit.kamailio/70.config-changed          *: 192.168.50.79:*
unit-kamailio-70: 10:56:19 DEBUG unit.kamailio/70.config-changed
unit-kamailio-70: 10:56:19 INFO juju.worker.uniter.operation ran "config-changed" hook
```

Figure 5-6 Kamailio log

After installation, the log provides the information about the successful installation of Kamailio server. Also it is providing the information about the config-changed hook is affected.

After installation MySQL and Kamailio relation will be added by the following command:

```
$ sudo juju add-relation kamailio mysql
```

The following output should be seen from juju debug log:

```
unit-kamailio-70: 11:37:51 INFO juju.worker.uniter.relation joining relation "kamailio:database mysql:db-admin"
unit-mysql-13: 11:37:51 INFO juju.worker.uniter.relation joining relation "kamailio:database mysql:db-admin"
unit-mysql-13: 11:37:51 INFO juju.worker.uniter.relation joined relation "kamailio:database mysql:db-admin"
unit-kamailio-70: 11:37:51 INFO juju.worker.uniter.relation joined relation "kamailio:database mysql:db-admin"
unit-kamailio-70: 11:37:51 INFO unit.kamailio/70.juju-log database:138: mysql/13 joined
unit-kamailio-70: 11:37:51 INFO unit.kamailio/70.juju-log database:138: KAM-NEW: database-relation-joined!
unit-kamailio-70: 11:37:51 INFO juju.worker.uniter.operation ran "database-relation-joined" hook
unit-kamailio-70: 11:37:52 INFO unit.kamailio/70.juju-log database:138: mysql/13 modified its settings
unit-kamailio-70: 11:37:52 INFO unit.kamailio/70.juju-log database:138: KAM-NEW: database-relation-changed!
unit-kamailio-70: 11:37:52 INFO unit.kamailio/70.juju-log database:138: mysql/13 modified its settings
unit-kamailio-70: 11:37:52 INFO unit.kamailio/70.juju-log database:138: KAM-NEW: database-relation-changed!
```

Figure 5-7 Log for Kamailio and MySQL relation joined

Figure 5-7 is showing the log for MySQL and Kamailio relation joined.

```
unit-kamailio-70: 11:38:15 INFO juju.worker.uniter awaiting error resolution for "relation-changed" hook
unit-kamailio-70: 11:38:15 INFO unit.kamailio/70.juju-log database:138: mysql/13 modified its settings
unit-kamailio-70: 11:38:15 INFO unit.kamailio/70.juju-log database:138: KAM-NEW: database-relation-changed!
unit-kamailio-70: 11:38:15 INFO unit.kamailio/70.juju-log database:138: mysql/13 modified its settings
unit-kamailio-70: 11:38:15 INFO unit.kamailio/70.juju-log database:138: KAM-NEW: database-relation-changed!
unit-kamailio-70: 11:38:15 INFO unit.kamailio/70.juju-log database:138: KAM-NEW: kam_host = 10.93.63.8
unit-kamailio-70: 11:38:15 INFO unit.kamailio/70.juju-log database:138: KAM-NEW: db_user = meobeinaijeubei
unit-kamailio-70: 11:38:15 INFO unit.kamailio/70.juju-log database:138: KAM-NEW: db_pass = geedaexohbaugoh
unit-kamailio-70: 11:38:15 INFO unit.kamailio/70.juju-log database:138: KAM-NEW: db_host = 10.93.63.124
unit-kamailio-70: 11:38:15 INFO unit.kamailio/70.juju-log database:138: KAM-NEW: db_db = kamailio
unit-kamailio-70: 11:38:15 INFO juju.worker.uniter.operation ran "database-relation-changed" hook
unit-kamailio-70: 11:38:15 INFO unit.kamailio/70.juju-log KAM-NEW: config-changed!
unit-kamailio-70: 11:38:15 INFO unit.kamailio/70.juju-log Received config: ip-addresses = 192.168.50.79
unit-kamailio-70: 11:38:15 INFO juju.worker.uniter.operation ran "config-changed" hook
```

Figure 5-8 Details log for Kamailio and MySQL

Figure 5-8 is providing the information for Kamailio application container host address and database user name, database password, database host address from MySQL application container. Kamailio is the database name.

To check the status of both containers, following figure will be seen with `juju status` command:

Model	Controller	Cloud/Region		Version	SLA	Timestamp		
default	localhost-localhost	localhost/localhost		2.4.4	unsupported	13:03:31+01:00		
App	Version	Status	Scale	Charm	Store	Rev	OS	Notes
kamailio		active	1	kamailio	local	90	ubuntu	
mysql	5.5.62	active	1	mysql	jujucharms	58	ubuntu	
Unit	Workload	Agent	Machine	Public address		Ports	Message	
kamailio/70*	active	idle	111	10.93.63.8				
mysql/13*	active	idle	110	10.93.63.124		3306/tcp	Ready	
Machine	State	DNS	Inst id		Series	AZ	Message	
110	started	10.93.63.124	juju-0349f2-110		trusty		Running	
111	started	10.93.63.8	juju-0349f2-111		bionic		Running	

Figure 5-9 Juju status for Kamailio and MySQL container

After a successful relation set up between Kamailio and MySQL application containers, Kamailio status has been changed from blocked to active.

5.3 Iptables setup

Iptables is required for this project for accessing the service provided by a container externally. Therefore, traffic incoming on the host/VM get processed and forwarded to the container. In the following step, to create the iptables rule which is used for adding the table rule when the source and destination belongs to the same subnet. The rule is attached to the PREROUTING chain of the table and is defined as follows:

```
$ sudo iptables -t nat -A PREROUTING -I enp0s3 -p udp --dport 5060 -j DNAT --to 10.93.63.8:5060
```

To check the iptables rule by the following command:

```
sudo iptables -t nat -L
```

The following figure 5-10 shows the iptables rule:

```
Chain PREROUTING (policy ACCEPT)
target     prot opt source                destination              udp dpt:sip to:10.93.63.8:5060
```

Figure 5-10 Iptables setup

To delete a rule, run the following command:

```
$ sudo iptables -t nat -D PREROUTING 1
```

where 1 is indicating the serial number of the rule from iptables.

5.4 Call setup

To setup a call between two users, PhonerLite softphone will be used and following configuration will be configured:

Server Benutzer Netzwerk Codecs Zertifikate Sound

Proxy/Registrar
 192.168.50.79
☒ Registrierung ☒ MWI
 900 Sekunden

STUN Server
 Domain/Realm

Figure 5-11 PhonerLite setup

Here, 192.168.50.79 is the host IP.

Following Figure 5-12 is showing the registration status before adding a relation between MySQL and Kamailio:

Server Benutzer Netzwerk Codecs Zertifikate Sound

Benutzername: test ☒
 Angezeigter Name:
 Mailbox-Nummer:
 Kennwort:
 Authentifizierungsname:
 Telefonnummer:

Settings sip:test@192.168.50.79 registriert

Figure 5-12 SIP status setup before Kamailio and MySQL relation

After adding a relation between MySQL and Kamailio, following figure 5-13: is showing the status of registration without adding a user:

Server Benutzer Netzwerk Codecs Zertifikate Sound

Benutzername: test ☒
 Angezeigter Name:
 Mailbox-Nummer:
 Kennwort:
 Authentifizierungsname:
 Telefonnummer:

Settings sip:test@192.168.50.79 nicht registriert <Unauthorized>

Figure 5-13 SIP status setup after Kamailio and MySQL relation without add subscriber

From Figure 5-12 and Figure 5-13, the main difference is authentication process. Without add a relation between Kamailio application container and MySQL application container don't need an authentication criterion but after the relation built up it's not possible without authentication.

To add a user to kamailio database below command will be used:

```
$ sudo juju run "kamctl add test@192.168.50.79 1234" --unit kamailio/70
```

After executing the command following figure will be seen:

```
root@bahar:~# juju run "kamctl add test@192.168.50.79 1234" --unit kamailio/70
new user 'test@192.168.50.79' added
mysql: [Warning] Using a password on the command line interface can be insecure.
mysql: [Warning] Using a password on the command line interface can be insecure.
```

Figure 5-14 Add a user

To log in Kamailio container following command will be execute:

```
$ sudo juju ssh kamailio
```

To check the subscriber list in the Kamailio container, following command will be used:

```
kamctl db show subscriber
```

After executing the command following figure will be seen:

```
ubuntu@juju-0349f2-111:~$ kamctl db show subscriber
mysql: [Warning] Using a password on the command line interface can be insecure.
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | username | domain | password | ha1 | ha1b | email_address | rpid |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | test | 192.168.50.79 | 1234 | aa0c25d20d7ba33941d60bdf5ca8bba7 | 7d6d126a4a204aaa5469629b34c317b2 | NULL | NULL |
| 2 | test1 | 192.168.50.79 | 12345 | 49fd459a740d179129be2373a279d629 | 1e9f9fd63388b8b0d052e029ddc1b7c7 | NULL | NULL |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

Figure 5-15 Kamailio user list

Following figure 5-16 is showing the status of valid user which is added in Kamailio database:

The screenshot shows the 'Benutzer' (User) tab in the Kamailio web interface. The 'Benutzername' (Username) field contains 'test' and is checked. The 'Kennwort' (Password) field is masked with dots. The 'Angezeigter Name' (Displayed Name) and 'Mailbox-Nummer' (Mailbox Number) fields are empty. The 'Authentifizierungsname' (Authentication Name) and 'Telefonnummer' (Phone Number) fields are also empty. At the bottom, a status bar shows 'sip:test@192.168.50.79 registriert' (registered) with a green status icon.

Figure 5-16 SIP status setup after Kamailio and MySQL relation with a valid subscriber

A successful call has been established between test and test1 user. The following Figure 5-17 is illustrating the call establishment from test1 to test user:

The screenshot shows the same 'Benutzer' tab as Figure 5-16. A small window titled 'PhonerLite: test' is overlaid on the right side of the interface. This window shows a call log with the following details: 'Anrufer: test1' (Caller: test1) and 'PhonerLite' (Device: PhonerLite). The call status is indicated by a green phone icon and a red hang-up icon.

Figure 5-17 Successful Call establishment

The following capture shows the message flow via the proxy authentication required functionality of SIP:

No.	Time	Source	Destination	Protocol	Length	Info
39	5.243313981	192.168.50.28	192.168.50.79	SIP/SDP	1281	Request: INVITE sip:test1@192.168.50.79
40	5.243344772	192.168.50.28	10.93.63.8	SIP/SDP	1281	Request: INVITE sip:test1@192.168.50.79
41	5.243349626	192.168.50.28	10.93.63.8	SIP/SDP	1281	Request: INVITE sip:test1@192.168.50.79
42	5.243977925	10.93.63.8	192.168.50.28	SIP	558	Status: 407 Proxy Authentication Required
43	5.243977925	10.93.63.8	192.168.50.28	SIP	558	Status: 407 Proxy Authentication Required
44	5.243988537	192.168.50.79	192.168.50.28	SIP	558	Status: 407 Proxy Authentication Required
45	5.244167067	192.168.50.28	192.168.50.79	SIP	388	Request: ACK sip:test1@192.168.50.79
46	5.244175781	192.168.50.28	10.93.63.8	SIP	388	Request: ACK sip:test1@192.168.50.79
47	5.244178049	192.168.50.28	10.93.63.8	SIP	388	Request: ACK sip:test1@192.168.50.79
48	5.244252032	192.168.50.28	192.168.50.79	SIP/SDP	1482	Request: INVITE sip:test1@192.168.50.79
49	5.244256844	192.168.50.28	10.93.63.8	SIP/SDP	1482	Request: INVITE sip:test1@192.168.50.79
50	5.244258106	192.168.50.28	10.93.63.8	SIP/SDP	1482	Request: INVITE sip:test1@192.168.50.79
57	5.245393249	10.93.63.8	192.168.50.28	SIP	433	Status: 100 trying -- your call is important to us
58	5.245393249	10.93.63.8	192.168.50.28	SIP	433	Status: 100 trying -- your call is important to us
59	5.245400726	192.168.50.79	192.168.50.28	SIP	433	Status: 100 trying -- your call is important to us
60	5.245690728	10.93.63.8	192.168.50.43	SIP/SDP	1443	Request: INVITE sip:test1@192.168.50.43:5060
61	5.245690728	10.93.63.8	192.168.50.43	SIP/SDP	1443	Request: INVITE sip:test1@192.168.50.43:5060
62	5.245697614	192.168.50.79	192.168.50.43	SIP/SDP	1443	Request: INVITE sip:test1@192.168.50.43:5060
63	5.248254695	192.168.50.43	192.168.50.79	SIP	607	Status: 100 Trying
64	5.248262275	192.168.50.43	10.93.63.8	SIP	607	Status: 100 Trying
65	5.248264004	192.168.50.43	10.93.63.8	SIP	607	Status: 100 Trying
66	5.295582700	192.168.50.43	192.168.50.79	SIP	774	Status: 180 Ringing
67	5.295603792	192.168.50.43	10.93.63.8	SIP	774	Status: 180 Ringing
68	5.295606495	192.168.50.43	10.93.63.8	SIP	180	Status: 180 Ringing
69	5.296120786	10.93.63.8	192.168.50.28	SIP	683	Status: 180 Ringing
70	5.296120786	10.93.63.8	192.168.50.28	SIP	683	Status: 180 Ringing
71	5.296129745	192.168.50.79	192.168.50.28	SIP	683	Status: 180 Ringing
94	9.218190240	192.168.50.43	192.168.50.79	SIP/SDP	1397	Status: 200 OK
95	9.218227597	192.168.50.43	10.93.63.8	SIP/SDP	1397	Status: 200 OK
96	9.218236715	192.168.50.43	10.93.63.8	SIP/SDP	1397	Status: 200 OK
97	9.219539077	10.93.63.8	192.168.50.28	SIP/SDP	1306	Status: 200 OK
98	9.219539077	10.93.63.8	192.168.50.28	SIP/SDP	1306	Status: 200 OK
99	9.219561437	192.168.50.79	192.168.50.28	SIP/SDP	1306	Status: 200 OK
100	9.221451757	192.168.50.28	192.168.50.79	SIP	803	Request: ACK sip:test1@192.168.50.43:5060;gr=00AC6F6C-F4F2-E811-8816-0926EA72C025
101	9.221470621	192.168.50.28	10.93.63.8	SIP	803	Request: ACK sip:test1@192.168.50.43:5060;gr=00AC6F6C-F4F2-E811-8816-0926EA72C025
102	9.221475270	192.168.50.28	10.93.63.8	SIP	803	Request: ACK sip:test1@192.168.50.43:5060;gr=00AC6F6C-F4F2-E811-8816-0926EA72C025
103	9.222773015	10.93.63.8	192.168.50.43	SIP	891	Request: ACK sip:test1@192.168.50.43:5060;gr=00AC6F6C-F4F2-E811-8816-0926EA72C025
104	9.222773015	10.93.63.8	192.168.50.43	SIP	891	Request: ACK sip:test1@192.168.50.43:5060;gr=00AC6F6C-F4F2-E811-8816-0926EA72C025
105	9.222793088	192.168.50.79	192.168.50.43	SIP	891	Request: ACK sip:test1@192.168.50.43:5060;gr=00AC6F6C-F4F2-E811-8816-0926EA72C025
138	12.728068386	192.168.50.28	192.168.50.79	SIP	838	Request: BYE sip:test1@192.168.50.43:5060;gr=00AC6F6C-F4F2-E811-8816-0926EA72C025
139	12.728831477	192.168.50.28	10.93.63.8	SIP	838	Request: BYE sip:test1@192.168.50.43:5060;gr=00AC6F6C-F4F2-E811-8816-0926EA72C025
140	12.728837198	192.168.50.28	10.93.63.8	SIP	838	Request: BYE sip:test1@192.168.50.43:5060;gr=00AC6F6C-F4F2-E811-8816-0926EA72C025
141	12.729638446	10.93.63.8	192.168.50.43	SIP	926	Request: BYE sip:test1@192.168.50.43:5060;gr=00AC6F6C-F4F2-E811-8816-0926EA72C025
142	12.729638446	10.93.63.8	192.168.50.43	SIP	926	Request: BYE sip:test1@192.168.50.43:5060;gr=00AC6F6C-F4F2-E811-8816-0926EA72C025
143	12.729647829	192.168.50.79	192.168.50.43	SIP	926	Request: BYE sip:test1@192.168.50.43:5060;gr=00AC6F6C-F4F2-E811-8816-0926EA72C025
144	12.730991072	192.168.50.43	192.168.50.79	SIP	593	Status: 200 OK
145	12.731000900	192.168.50.43	10.93.63.8	SIP	593	Status: 200 OK
146	12.731003579	192.168.50.43	10.93.63.8	SIP	593	Status: 200 OK
147	12.731138368	10.93.63.8	192.168.50.28	SIP	502	Status: 200 OK
148	12.731138368	10.93.63.8	192.168.50.28	SIP	502	Status: 200 OK
149	12.731158657	192.168.50.79	192.168.50.28	SIP	502	Status: 200 OK

Figure 5-18 SIP capture between two users

From the Figure 5-18, a call has been established between two users, maintaining the all procedures. According to the previous discussion 2.8 where a SIP call flow chart has been shown, an invitation has been sent from test1 user to test user. After trying and ringing a successful call has been established with 200 OK. Call has been cancelled with BYE and 200 OK message. Every request and response are happened three times or multiple of three times because of the architecture of network. From the section 3.4, where architecture is provided for this student project in the target state. Due to the capturing on all interfaces of the virtual machine, a packet is being captured on each involved interface. In case of an incoming or outgoing SIP message, the corresponding packet is captured on the enp0s3 interface of the Linux-host, on the LXD-Bridge lxdbr1 and on the eth0 interface of the container providing the Kamailio server.

In the Figure 5-19, a 200 OK message header is been shown. Where vis is providing the proxy server IP addresses. From and To are providing the calling party and called party address respectively. Contact part addressing the OK message sender details.


```
▼ Session Initiation Protocol (200)
  > Status-Line: SIP/2.0 200 OK
  ▼ Message Header
    > Via: SIP/2.0/UDP 192.168.50.79:5060;branch=z9hG4bK2dad.e8fdb140b87bb4a0a68b22f7d54c5137.0
    > Via: SIP/2.0/UDP 192.168.50.28:5060;received=192.168.50.28;branch=z9hG4bK00b241a6f6f2e811a56cacbae5ab2439;rport=5060
    > Record-Route: <sip:192.168.50.79;lr>
    > From: <sip:test@192.168.50.79>;tag=3440202114
    > To: <sip:test1@192.168.50.79>;tag=00b241a6f6f2e811881c0926ea72c025
    Call-ID: 00B241A6-F6F2-E811-A56A-ACBAE5AB2439@192.168.50.28
    > CSeq: 212 INVITE
    > Contact: <sip:test1@192.168.50.43:5060;gr=00AC6F6C-F4F2-E811-8816-0926EA72C025>
    Content-Type: application/sdp
    Allow: INVITE, ACK, BYE, CANCEL, INFO, MESSAGE, NOTIFY, OPTIONS, REFER, UPDATE, PRACK
    Session-Expires: 1800;refresher=uac
    Supported: 100rel, replaces, from-change, gruu, timer
    Server: SIPPER for PhonerLite
    Content-Length: 549
```

Figure 5-19 Message Header details for 200 OK Message

6 Evaluation of Juju as VNFM

Juju can instantiate and configure application from a charm or bundle of charm by `deploy` command. Charm is providing the all required software and configuration procedure during the deployment. Juju can deploy the same application one or multiple unit at the same time. Scaling is one of the main functionalities of cloud computing. Juju can scale up and down for a service without any complexity. To scale up, Juju is using `add-unit` command as well as to scale down it is using `remove-unit`. To scale up or down, Juju is getting this power from charm. Charms is providing the all types of information about an application.

Juju is managing the lifecycle of an application. It's also controlling the application dependency like which application integrate to another application through the charm. It maintains the relation between applications, so users don't need to concern about elasticity of his environment. Juju is using `add-relation` command to create dependency one application to another. To remove the relation between two applications it's using `remove-relation` command. Juju can terminate lifecycle of an application or multiple applications by using `remove-application` command.

Juju is not only providing the application configurations but also collecting the data with in time series to support and manage an application. Juju metric is providing the opportunity to analyze the data to effectively operate and manage an application. Juju executes the `collect-metrics` hooks to get the information of a unit. Using the `add-metric` command in `collect-metrics` hook is providing the measurement records which is forwarding to the Juju controller.

Modification is one of the main features in the cloud computing. Juju is able to make any kind of change in application which is already deployed or during the deploy process. For modification, Juju is using `config` command to modify any application.

Upgrading an application is very important in cloud. Most of the application has a new version after a certain time. Juju is supporting to upgrade an application by using `upgrade-charm`. With this command an application is switching out the current charm and replacing with new charm.

From the discussion in the section 2.4 and this chapter, it can be summaries that Juju is able to perform like virtual network function manager (VNFM). It can manage, configure, upgrade and provide all kind of support during the deployment and after the deployment of the application. This student project work also providing the practical experience of Juju as a VNFM with some functionality of Juju. The usage of Juju as a VNFM, charms are required which have to be developed in case they are not existing, and this will require a certain development effort for each required service.

7 Summary and Perspectives

In this student project, Juju is the key word which is used to automate the cloud infrastructure. Juju is providing all kind of assistance to automate a service through the charm. Charm is encapsulating application configurations, manage and maintain the application. Last but not least, it's also providing the scale efficiency for an application. Charms will provide the information about an application will be connected to which application. In one sentence it can be said that, charm will do all the work for user but if there have no charm in charm store regarding user application then user must be developed a charm for that application. If user write a charm for an application, then using that charm user can deploy an application without any limitation. Juju can add relation between applications as well as remove that relation too.

In this student project a charm has been developed in bash script for Kamailio server. This charm has all kind of software for deploying Kamailio server. The following files and hooks are written to develop the Kamailio charm in YAML and bash scripts respectively:

Metadata file: This file is providing the information about the charm name and others charm which relates to this charm via interface name. This file must be present in a charm. It's also providing the summary and description of the charm.

Config file: Config file is furnishing the application if there have any change. This file is working during the deployment but it's also possible to modify after the application is deployed. This config file is supporting the modification when application is on operation.

Install hook: Installing and fetching the software for Kamailio charm are providing by this hook. All the software is installed with the given information through this hook. This hook is run only one time in complete lifecycle of the charm. The installed software never changes with configuration changes.

Database-relation-changed hook: This hook is working when Kamailio makes a relation with MySQL and providing all kind of changes in Kamailio application.

Database-relation-departed hook: This hook works when applications are removing relation between them. When Kamailio is removing the relation with MySQL, this hook will drop the all kind of relation between them.

Config changed hook: This hook is updating the configuration if there have any change in the config file. This hook runs in different situations like after install or if charm is updated or if there any change in configuration.

After developed the charm for Kamailio, Deployed MySQL application from charm store and Kamailio application from developed charm in local machine in different container and add a relation between them using Juju. Two users have been created for Kamailio and established a successful call between them using PhonerLite softphone. Finally, captured has been analyzed from Wireshark capture. And at the end, the main goal for this project is evaluation Juju as a Virtualized Network Function Manager (VNFM) has been described.

The main benefit of Juju is providing the automatic deploying behavior for any application as well as reuse the same code and model without any change. Juju is performing to reduce the complexity of micro-services. Providing services are the main concept for the Juju. Juju can maintain not only the lifecycle of the instance but also how an instance is configured for an application. An application will be deployed with all required configuration for the instance. It can maintain multiple instance or application through command line interface or graphical user interface. Juju can modify any application and update the software for that application. Juju provided the relation between two application as well as termination. Any error can be detected through the Juju debugging system and if there have any change for the application also.

For the future extension of this project, again add a relation after removing an existence relation between two applications is still under development which is containing multiple errors and missing parts. Charm store has

some policy to upload a charm. In the next step, this charm will be updated as it can fulfill all kind of requirements to upload in charm store. Juju is supporting only IPv4 address not Ipv6. Removing an application with error is different and difficult compare to successful application. If any error is happening on the installation process need to deploy full application again and it's time consuming. In the next step this Kamilio server will deploy with MySQL database as a bundle and to improve the monitoring of Kamilio server metric part will be used.

8 Abbreviations

C

CpaEx Capital Expense

G

ETSI European Telecommunication Standard Institute

G

GSM Global System for Mobile communications

GUI Graphical User Interface

GPL General Public License

H

HTML Hypertext Mark-up Language

HTTP Hypertext Transfer Protocol

I

IP Internet Protocol

ISG Industry Specification Group

L

LXC Linux Container

LXD Linux Container Daemon

M

MANO Management and Orchestration

N

NFV Network Function Virtualization

NFV	Network Function Virtualization Infrastructure
NFVO	Network Functions Virtualization Orchestrator

O

OpEx	Operational Expense
------	---------------------

S

SIP	Session Initiation Protocol
-----	-----------------------------

V

VIM	Virtualized Infrastructure Manager
VM	Virtual Machine
VNFM	Virtual Network Function Manager
VOIP	Voice Over Internet Protocol

9 References

1. OPNFV (2016): *Accelerating Business with OpenStack and OPNFV*, <https://www.openstack.org/assets/marketing/OPNFVandOpenStack-Digital.pdf> [accessed 8 October 2018].
2. FN Division Telecommunication Engineering Centre: *Network Function Virtualization (NFV) & Its impact on Future Telecom Networks*, [online] http://tec.gov.in/pdf/Studypaper/Network_Function_Virtualization%20.pdf [accessed 8 October 2018].
3. Balamurali Thekkedath (2016): *Network Functions virtualization FOR DUMMIES*, John Wiley & Sons, Inc. [accessed 8 October 2018].
4. ETSI GS NFV-MAN 001 (2014-12): *Network functions virtualization (NFV); management and orchestration*, https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFVMAN001v010101p.pdf [accessed 9th October 2018]
5. James Bottomley and Pavel Emelyanov (2014): *Containers* https://www.usenix.org/system/files/login/articles/login_1410_02-bottomley.pdf [accessed 9th October 2018]
6. Docker (2018): *What is a container*, <https://www.docker.com/resources/what-container> [accessed 9th October 2018]
7. Norman Joyner (2016): *A History Of Container Technology*, <https://blog.containership.io/a-history-of-container-technology> [accessed 12th October 2018]
8. bmc blogs (2018): *Containers vs Virtual Machines: What's the Difference?*, <https://www.bmc.com/blogs/containers-vs-virtual-machines> [accessed 12th October 2018]
9. Containers (2018): *What's a Linux Container?* <https://www.redhat.com/en/topics/containers/whats-a-linux-container> [accessed 13th October 2018]
10. Linux Containers: *LXC*, [online] <https://linuxcontainers.org/lxc/> [accessed 15th October 2018]
11. Linux Containers: *LXD*, [online] <https://linuxcontainers.org/lxd/> [accessed 15th October 2018]
12. Azhagappan, D. (2015): *Deploy and Test Open Platform for Network Function Virtualization (OPNFV) within the laboratory environment*, Frankfurt: Frankfurt University of Applied Sciences.
13. Juju: *Juju Documentation*, [online] <https://docs.jujucharms.com/2.4/en/> [accessed 15th October 2018]
14. Kamilio: <https://www.kamilio.org/w/> [online] [accessed 9th October 2018]
15. MIT (2005): *SIP Basics*, [online] <http://web.mit.edu/sip/presentations/np119.pdf> [accessed 9th October 2018]
16. RFC 3261 (2002): *SIP: Session Initiation Protocol*, [online] <https://tools.ietf.org/html/rfc3261> [accessed 15th October 2018]
17. VirtualBox (2018): *VirtualBox downloads*, <https://www.virtualbox.org/> [accessed 15 October 2018].

-
18. Oracle VM VirtualBox (2017): *user manual*,
<https://download.virtualbox.org/virtualbox/5.1.22/UserManual.pdf> [accessed 15th October 2018]
 19. Artur Tyloch (2015): *Juju for Telcos and Service Providers*, [online]
<https://blog.ubuntu.com/2015/07/01/juju-for-telcos-and-service-providers-pt-1-2> [accessed 27th October 2018]

Attached CD/DVD content

The attached contains the following items:

- Documentation in Word.
- Documentation in PDF.
- Reference books used in documentation.
- Wireshark capture.