CLOUD COMPUTING ENHANCEMENTS AND PRIVATE CLOUD MANAGEMENT

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Antonio Alonso Gil

In partial fulfilment

of the requirements for the degree in

AUDIOVISUAL SYSTEMS ENGINEERING

Advisors: Paulo da Fonseca Pinto

Pedro Amaral

Jordi Domingo-Pascual

Barcelona, May 2018

Abstract

This is a project developed in the Departamento de Engenharia Electrotécnica (DEE) in the Faculdade de Ciências e Tecnologia (FCT) from Universidade NOVA de Lisboa (UNL).

The objective of this project is to implement a private cloud in a small datacenter network using the MAAS server provisioning tool and the Openstack software platform for cloud computing, leaving it ready to be interconnected it with an experimental SDN Network.

The private cloud and Network will serve the telecommunications group undergraduate and post-graduate labs and it will be used both as a production Network and as a test bed for new research with the cloud being used to integrate several available computing resources in order to maximize the available computation power for research tasks.

Resum

Aquest és un projecte desenvolupat al Departament d'Enginyeria Electrotècnica (DEE) a la Facultat de Ciències i Tecnologia (FCT) de la Universitat NOVA de Lisboa (UNL).

L'objectiu d'aquest projecte és implementar un núvol privat en una petita xarxa de centre de dades fent servir la eina d'aprovisionament de servidors MAAS el projecte de computació en el núvol Openstack per a que posteriorment aquesta sigui interconectada amb una xarxa SDN experimental.

El núvol privat i la xarxa serviràn als laboratoris de pregrau i postgrau del grup de telecomunicacions de la Universitat i s'utilitzarà tant com una xarxa de producció com un banc de proves per a noves investigacions, fent servir el núvol per integrar diversos recursos informàtics disponibles per maximitzar la computació disponible per a tasques d'investigació.

Resumen

Este es un proyecto desarrollado en el Departamento de Ingeniería Electrotécnica (DEE) en la Facultad de Ciencias y Tecnología (FCT) de la Universidad NOVA de Lisboa (UNL).

El objetivo de este proyecto es implementar una nube privada en una pequeña red de centro de datos usando la herramienta provisionadora de servidores MAAS y el proyecto de computación en la nuve Openstack para que posteriormente esta sea interconectada con una red SDN experimental.

La nube privada y la red servirán a los laboratorios de pregrado y posgrado del grupo de telecomunicaciones de la Universidad y se utilizará tanto como una red de producción como un banco de pruebas para nuevas investigaciones, usando la nuve para integrar vários recursos informáticos disponibles para maximizar la computación disponible para las tareas de investigación.

Dedication

To my family and friends,

for supporting me in the toughest moments.

Acknowledgements

I wish to express my sincere thanks to Paulo da Fonseca Pinto and Pedro Amaral, Professors of the Department of Electronic Engineering at the Faculty of Science and Technology in Universidade NOVA de Lisboa, for giving me the opportunity to work with them and for the facilities of adaptation they have offered me as a foreigner, making my work at the laboratory much more easy.

Also I would like to thank Professor Jordi Pascual-Domingo, who agreed to be my supervisor in Barcelona. I really appreciate his help and his kind attention throughout my thesis.

Finally for my friends, who give me their strength in the toughest moments and of course for Antonio Alonso, my father, and Concepción Gil, my mother because without their bravery and his effort in my education this project would not have been possible.

Revision history and approval record

Revision	Date	Purpose
0	23/04/2018	Document creation
1	10/05/2018	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Toni Alonso	tony.ag@hotmail.com
Paulo da Fonseca Pinto	pfp@fct.unl.pt
Pedro Amaral	pfa@fct.unl.pt
Jordi Domingo-Pascual	jordi.domingo@ac.upc.edu

Written by:		Reviewed and approved by:	
Date	23/04/2018	Date	10/05/2018
Name	Toni Alonso Gil	Name	Paulo da Fonseca Pinto Pedro Amaral
Position	Project author	Position	Project director Manager/Advisor

Table of contents

List of Figures

List of Tables

1. Introduction

The purpose of this section is to provide an executive summary of the project. A detailed documentation of the technologies studied and implemented are found in this report.

1.1. Statement of purpose (objectives)

During the course of the project I have discovered the high complexity of the platform in which I have been working, since it has a modular architecture that supports the most advanced technologies in its field.

The deployment of MAAS and OpenStack in the personal computer was a process quite extensive in time, since new technologies were learned from the scratch and multiple configuration problems were solved in the meantime.

Based on this experience, the following objectives were defined:

- Formation of a broad theoretical and practical knowledge of MAAS and OpenStack.
- Through OpenStack, an open source tool, make the network and computing resources of the data center scalable, low cost, flexible, easy to manage and with the option of enabling multiple client services.
- Make a theoretical and practical documentation for the support of new subjects or projects related to Cloud Computing, as well as for the future investigation lines for the readers of the project.

1.2. Requirements and specifications

Project requirements:

- Private Cloud based VM provisioning for both end users and network management software.
- Separation between the Network Control VMs – Openflow Controllers – Network Sitches traffic and the User VMs – End Hosts traffic.

Project specifications:

- Use of MAAS for building the private cloud to provide virtual machines.
- Use of Openstack for setting up the private cloud and leave it ready to interconnect it with the existing SDN network.

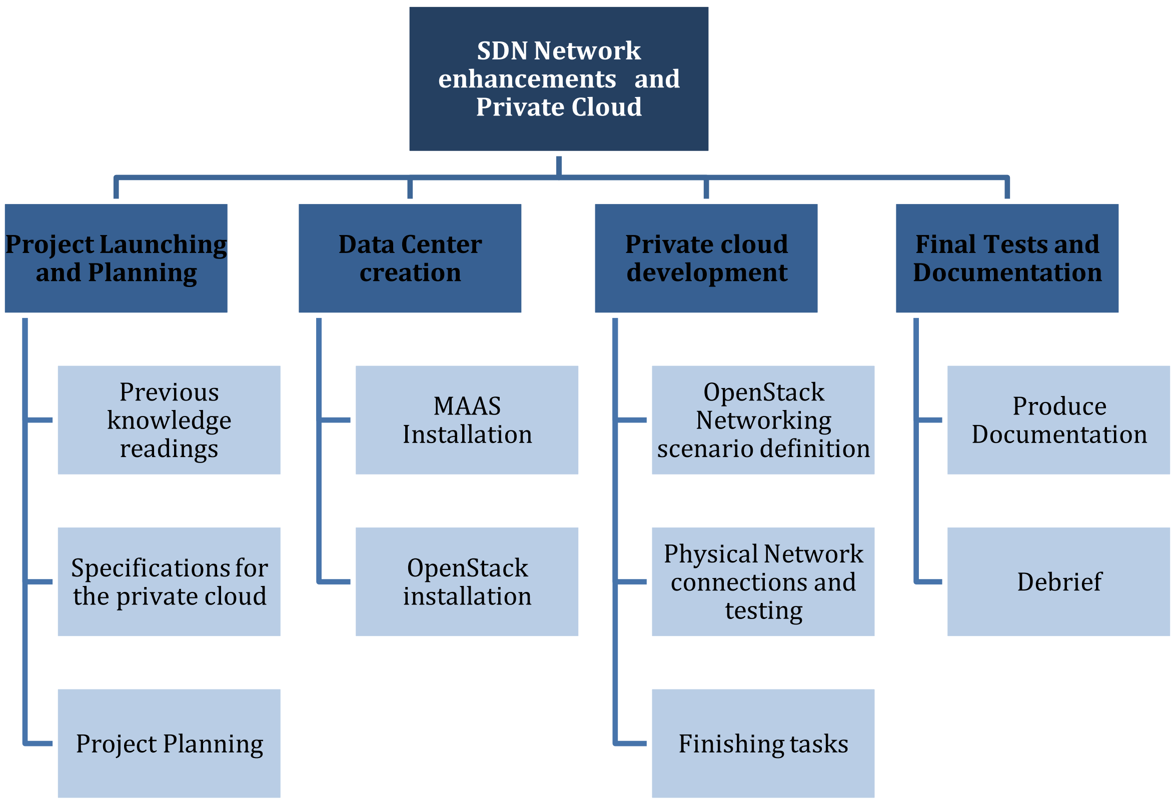
1.3. Methods and procedures

This work is not a continuation of any another project and don’t use any applications, or algorithms previously developed by other authors.

The only thing that has been previously developed is the existing network infrastructure where this project has been gone through.

1.4. Work plan

Work Breakdown:


SDN Network enhancements and Private Cloud
Project Launching and Planning
Data Center creation
Private cloud development
Final Tests and Documentation
Previous knowledge readings
Specifications for the private cloud
Project Planning
MAAS Installation
OpenStack installation
OpenStack Networking scenario definition
Physical Network connections and testing
Finishing tasks
Produce Documentation
Debrief

Work Packages:

Project: SDN Network enhancements and Private Cloud	WP ref: (WP1)	
Major constituent: Project Launching and Planning	Sheet 1 of 4	
Short description: Initial requirements specification and technology brief.	Planned start date: 06/10/2017 Planned end date: 04/12/2017	
	Start event: End event:	
Internal task T1: Previous knowledge readings Internal task T2: Specifications for the private cloud and SDN Network enhancements Internal task T3: Project Planning	Deliverables: Requirement Specification Project Plan	Links: Requirement Specification Project Plan

Project: SDN Network enhancements and Private Cloud	WP ref: 2	
Major constituent: Data Center creation	Sheet 2 of 4	
Short description: Placement of the required mechanisms to create the data center	Planned start date: 07/12/2017 Planned end date: 21/02/2018	
	Start event: End event:	
Internal task T1: MAAS Installation. Internal task T2: Openstack Installation	Deliverables:	Links:

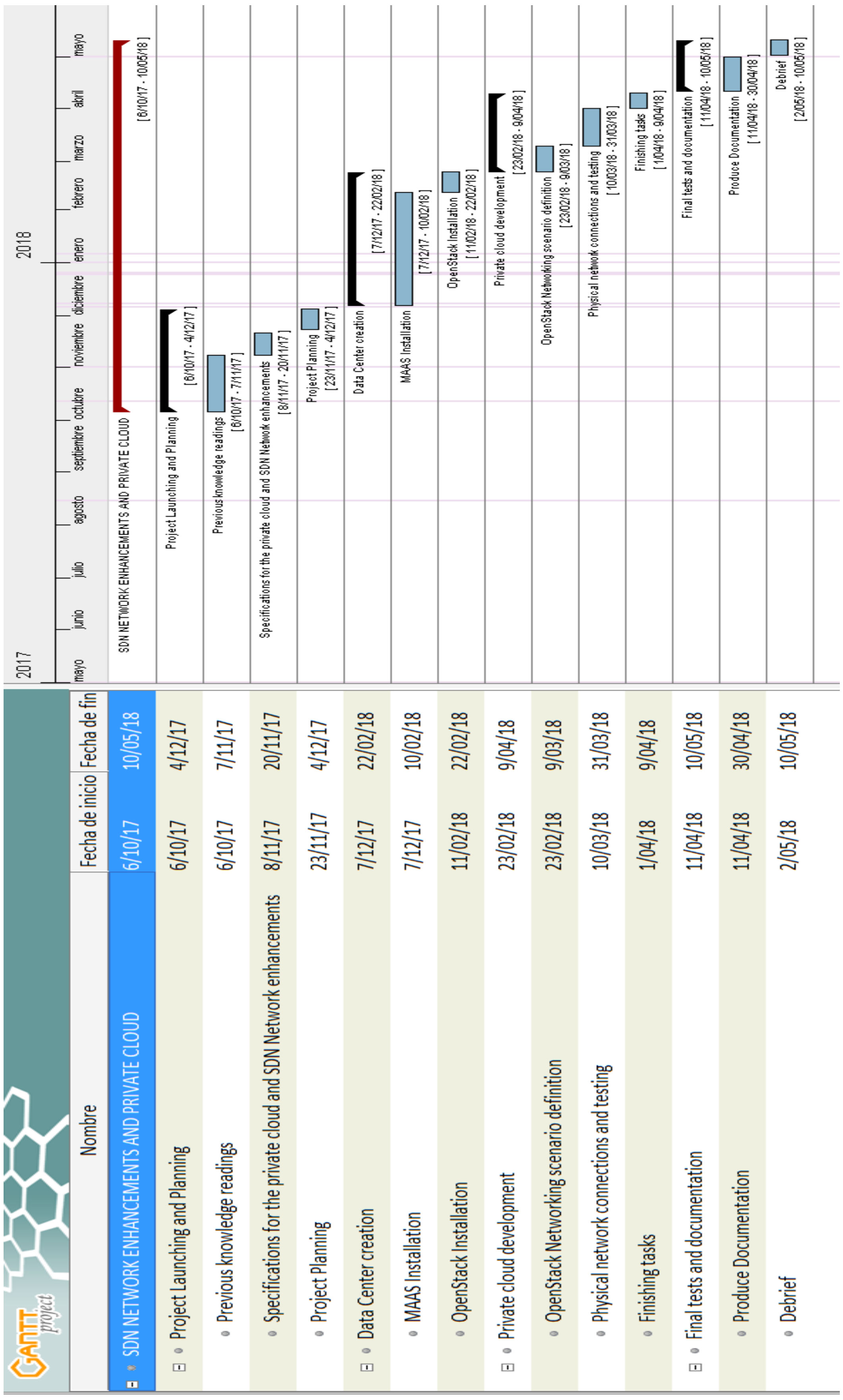
Project: SDN Network enhancements and Private Cloud	WP ref: 3	
Major constituent: Private cloud development	Sheet 3 of 4	
Short description: Designing of the private cloud implementation and Data Center network.	Planned start date: 23/02/2018 Planned end date: 09/04/2018	
	Start event: End event:	
Internal task T1: Openstack Networking scenario definition Internal task T2: Physical Network connections and testing Internal task T3: Finishing tasks	Deliverables:	Links:

Project: Results presentation	WP ref: (WP4)	
Major constituent: Final Tests and Documentation	Sheet 4 of 4	
Short description: Procurement of the documentation on the implementation.	Planned start date: 11/04/2018 Planned end date: 11/05/2018	
	Start event: End event:	
Internal task T1: Produce Documentation Internal task T2: Debrief	Deliverables: System Documentation	Links:

Milestones:

WP#	Task#	Short title	Milestone / deliverable	Date (week)
1	1	Previous knowledge readings	---	06/10/2017
1	2	Specifications for the private cloud and SDN enhancements	Requirement Specification	08/11/2017
1	3	Project Planning	Project Plan	23/11/2017
2	1	MAAS Installation	---	06/12/2017
2	2	OpenStack Installation	---	10/01/2018
3	1	OpenStack Networking scenario definition	---	15/02/2018
3	2	Physical network connection and testing	---	10/03/2018
3	3	Finishing tasks	---	01/04/2018
4	1	Produce Documentation	System Documentation	11/04/2018
4	2	Debrief		02/05/2018

Time plan:



Nombre	Fecha de inicio	Fecha de fin
SDN NETWORK ENHANCEMENTS AND PRIVATE CLOUD	6/10/17	10/05/18
Project Launching and Planning	6/10/17	4/12/17
Previous knowledge readings	6/10/17	7/11/17
Specifications for the private cloud and SDN Network enhancements	8/11/17	20/11/17
Project Planning	23/11/17	4/12/17
Data Center creation	7/12/17	22/02/18
MAAS Installation	7/12/17	10/02/18
OpenStack Installation	11/02/18	22/02/18
Private cloud development	23/02/18	9/04/18
OpenStack Networking scenario definition	23/02/18	9/03/18
Physical network connections and testing	10/03/18	31/03/18
Finishing tasks	1/04/18	9/04/18
Final tests and documentation	11/04/18	10/05/18
Produce Documentation	11/04/18	30/04/18
Debrief	2/05/18	10/05/18

1.5. Deviations from the initial plan

This project was initially planned to include the implementation of the controller of the SDN network of the university in the private cloud, but due to some delays this goal couldn’t have been achieved.

Actually this project had to dedicate more time than expected on the documentation of Cloud Computing and SDN network (in spite of it was useless at the end) and on MAAS installation and OpenStack Networking scenario definition, due to my absolute previous lack of knowledge about these concepts and to some personal extern mobility problems as well.

2. State of the art of the technology used or applied in this thesis:

The main objective in this point is to provide the relevant background information to this thesis. It introduces the reader into a number of various fundamental concepts including, Cloud Computing, Metal as a Service, Juju and Openstack.

2.1. Cloud computing

Before starting with the definition of Cloud Computing, a short explanation of the concept of **virtualization** is necessary.

Virtualization is a fundamental element of Cloud Computing since it guarantees the delivery of shared computer resources to the same server, thus reducing infrastructure costs and increasing the flexibility and reliability of existing hardware.

Virtualization is a technique consisted on creating a virtual versions of a device or a resource such as a server, a storage device, a network or even an operating system where the resource is divided into one or more execution environments.

In general terms, virtualization represents a physical server partitioned into multiple virtual servers. Each virtual machine (VM) can interact independently with other devices, applications, data and users, as if it was a separate physical resource. Different virtual machines can run different operating systems and multiple applications simultaneously using a single physical computer. The software that enables virtualization is called Hypervisor. This software, also known as 'virtualization administrator', is located between the hardware and operating system, separating the operating system and hardware applications.

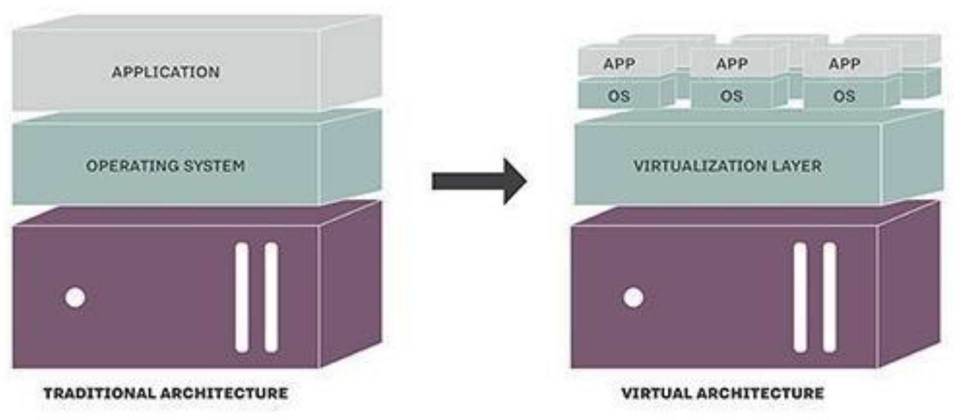


Figure 1: Traditional and virtual architecture comparison

Currently there are three major types of virtualization:

- Full virtualization: It is composed of three main layers: hardware, hypervisor and operating system instances. The hypervisor interacts with the hardware and keeps each virtual server isolated from the rest of the services. The hypervisor consumes a lot of resources like RAM, CPU, etc. Each instance can run its own operating system and these do not need to be modified. KVM and XEN provide support for this type of virtualization.

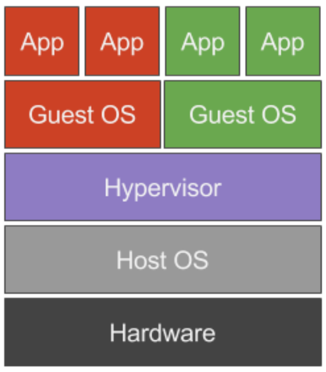


Figure 2: Full virtualization scheme

- Paravirtualization: Paravirtualization is similar to complete virtualization, but now virtual servers "know" that they are being virtualized and work together with other virtual servers. For this reason, paravirtualization requires changes in the operating system of virtual machines but it needs fewer resources for the same workload. In other words, paravirtualization allows greater scalability at the expense of some changes in the Guest operating system. In this case the hypervisor is in the "Virtualization Software Layer" layer. XEN supports this type of virtualization.

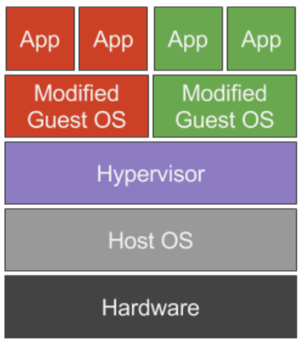


Figure 3: Paravirtualization scheme

- OS level virtualization: This type of virtualization does not require any type of hypervisor since the virtualization mechanisms are part of the operating system. It is the most scalable type of virtualization, since it does not need a hypervisor and the instances are much lighter than the previous cases. On the other hand, it presents the limitation that the operating system of the instance must be equal to that of the Host. LXC Containers fall within this classification.

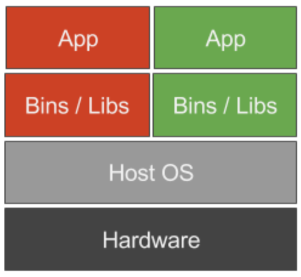


Figure 4: OS level virtualization scheme

Why using virtualization?

As we have said, virtualization in Cloud Computing offers several benefits, such as saving time and energy, reducing costs and minimizing risks:

- Provides ability to manage resources effectively.
- Increase productivity, as it provides secure remote access.
- Provides prevention against possible loss of data.

To make virtualization possible, software known as hypervisor, virtual machine monitor or virtualization administrator is used. This software is placed between the hardware and the operating system and allocates the amount of access that the applications and operating systems have with the processor and other hardware resources.

Cloud Computing

Cloud Computing term refers to a new technological conception and business model in which services of storage, access and use of computer resources located in the network are provided. All hardware management is carried out by the Cloud Computing provider, ensuring certain availability and power under certain service levels (SLAs).

Thus, the client or user can ignore physical resources and focus on the central activity of their business (web development, private business network, etc.). On the other hand, an end user does not need a high-performance device since higher-performance tasks can be transferred to cloud computing.

An example of cloud computing would be Google Docs / Google Apps, a system of electronic documents and applications. For its use software needs to be installed or have a server. The only requirement needed to use any of their services is an Internet connection.

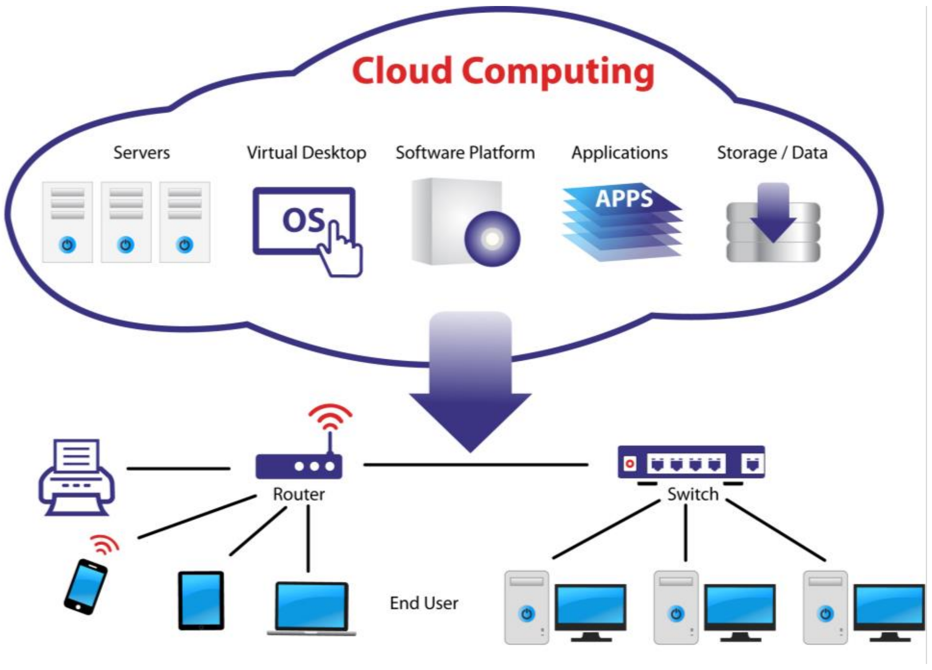


Figure 5: Cloud computing scheme

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics:

- <u>On-demand self-service:</u> A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

- <u>Broad network access:</u> Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

- <u>Resource pooling:</u> The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.

- <u>Rapid elasticity:</u> Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

- <u>Measured service:</u> Cloud systems automatically control and optimize resource use by leveraging a metering capability1 at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

Cloud computing is usually described in one of two ways. Either based on the cloud location, or on the service that the cloud is offering.

Based on a cloud **location**, we can classify cloud as:

- **Private cloud**: The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

- **Public cloud**: The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.

- **Community cloud:** The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.

- **Hybrid cloud**: The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

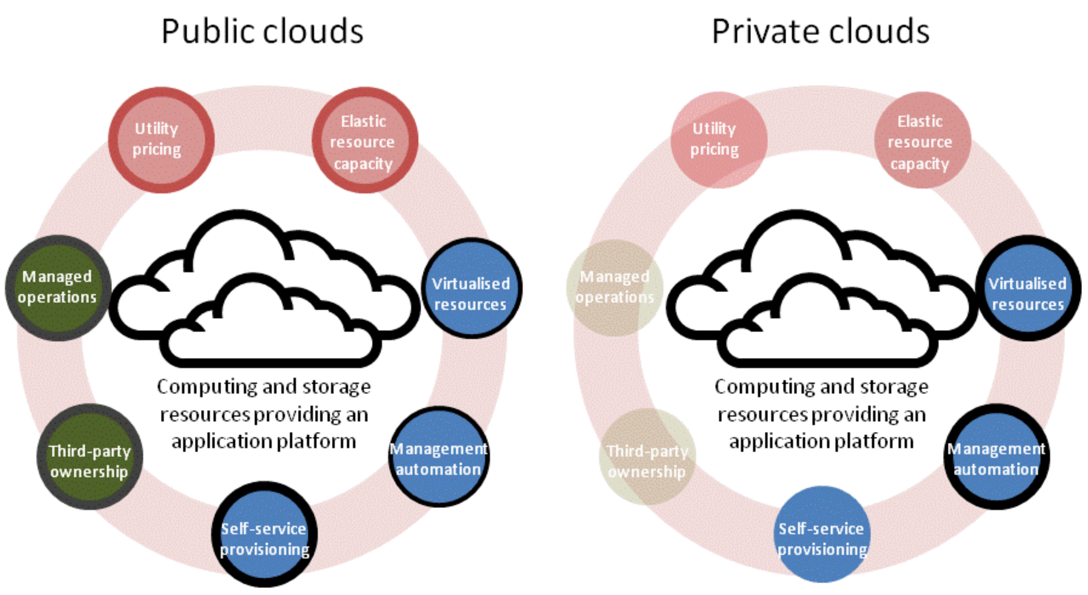

Public clouds
Utility pricing
Elastic resource capacity
Managed operations
Virtualised resources
Computing and storage resources providing an application platform
Third-party ownership
Management automation
Self-service provisioning
Private clouds
Utility pricing
Elastic resource capacity
Managed operations
Virtualised resources
Computing and storage resources providing an application platform
Third-party ownership
Management automation
Self-service provisioning

Figure 6: Public cloud and private cloud

Based on a **service** that the cloud is offering, we are speaking of either:

- **Software as a Service (SaaS):** The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited userspecific application configuration settings.

- **Platform as a Service (PaaS):** The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider.3 The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

- **Infrastructure as a Service (IaaS):** The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

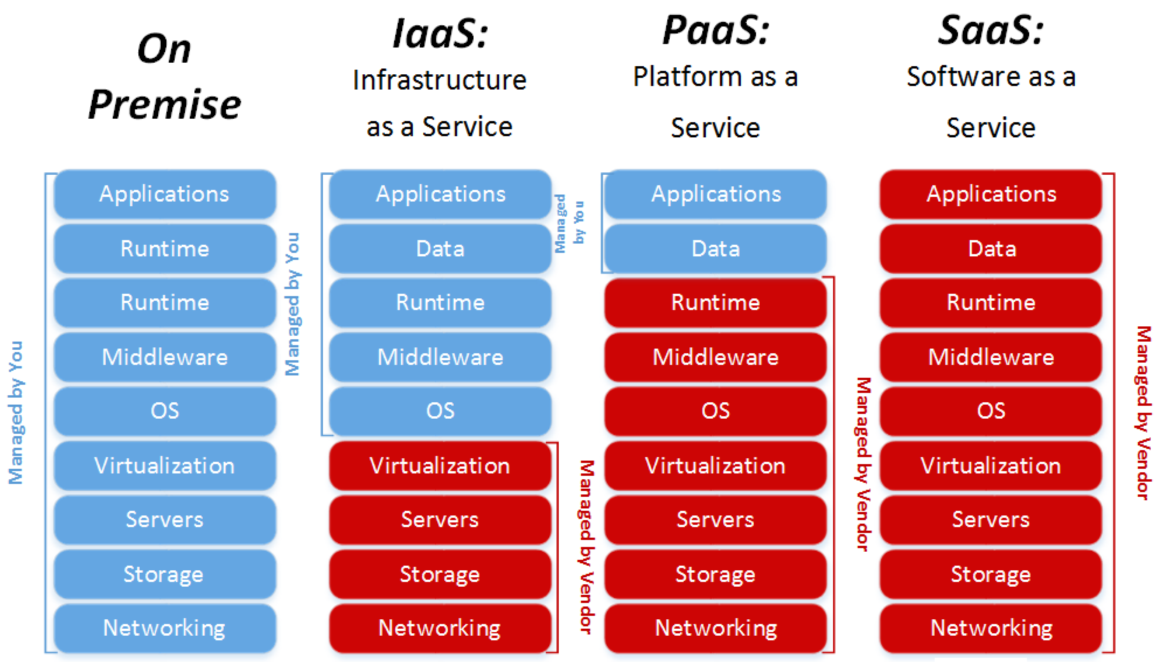


Figure 7: Cloud services

A new classification of Cloud Computing is currently available: **Metal as a Service (MAAS).**

MaaS allows the user to have a full control of the physical servers including the BIOS and its network connection, paying for the exact time of use. Thus, the operator can build a network from the lowest level controlling the security of all software levels.

The concept of MAAS is going to be explained in the next section.

2.2. MAAS

- ***What is MAAS?***

Metal As A Service is a server provisioning tool that lets you treat physical servers like virtual machines (instances) in the cloud. It delivers real servers on demand just like a cloud delivers virtual machines. So rather than having to manage each server individually, MAAS turns the bare metal into an elastic cloud-like resource. Machines can be quickly provisioned and then destroyed again easily with instances in a public cloud like Amazon AWS, Google GCE, and Microsoft Azure, among others.

It also can act as a standalone PXE/preseed (Preboot Execution Environment) service or it can be integrated with other technologies. In particular, it is designed to work especially well with Juju, the service and model management service. We'll talk about Juju later in this paper.

MAAS delivers the fastest OS installation times on bare metal in the industry thanks to its optimised image-based installer. This Metal as a Service tool is a provisioning construct created by Canonical, developers of the Ubuntu Linux-based operating system and serves as a layer underneath Infrastructure-as-a-Service (IaaS) to coordinate applications and workloads, deploying hardware and services that can dynamically scale up and down.

This tool is designed to help facilitate and automate the deployment and dynamic provisioning of hyperscale computing environments such as big data workloads and cloud services.

- ***Hardware configuration:***

With MAAS, the user only touchs the power button once. During the initial startup of a new server, MAAS indexes it, provisions it, and makes it cloud ready. A catalogue is maintained of not only the servers, but also the inventory of devices available in them. This is a key aspect of future provision automation.

- ***Ongoing infrastructure operations:***

Beyond initial configuration, MAAS also handles ongoing physical IP and DNS management. A "lights out" datacentre, with a near-zero need for hands-on operations, is realized with MAAS.

- ***Accessible***:

MAAS provides a REST API, Web-based interface and command line interface. It is designed with automation and hardware-at-scale in mind. Developers can even leverage it for bare metal workload management.

- ***Integration:***

Since there’s an API, as well as a CLI, automation tools like Juju, Chef, Puppet, SALT, Ansible, and more, are all easily integrated with MAAS.

That means legacy, scripted automation, like Puppet and Chef, are easily integrated, whilst modern modelling tools, like Juju, can naturally rely on MAAS for hardware information.

- ***How MAAS works?***

MAAS manages a pool of nodes. After registering ("Enlisting" state) a new system and preparing it for service ("Commissioning" state), the system joins the pool and is available for use ("Ready" state). MAAS controls machines through IPMI (or another BMC) or converged chassis controller such as Cisco UCS.

Its worth to mention that a machine destined for MAAS will have its disk space overwritten. A node in the pool is under MAAS's exclusive control and should not be provisioned using other methods.

Once the node is ready, users of the MAAS then allocate them for their own use ("Acquire") when they go into use. Any subsequently installed operating system will contain the user's SSH public key for remote access (the user's MAAS account first needs to import the key). The web UI also allows for manual allocation in the sense of reserving hardware to specific users for later use.

When allocating from the API/CLI, the user can specify requirements ("constraints") for a machine. Common constraints are: memory, CPU cores, connected networks, and what physical zone they should be in.

An allocated MAAS node is not like a virtual instance in a cloud: you get complete control, including hardware drivers and root access. To upgrade a BIOS, for example, an administrator could allocate a node to themselves, and run a vendor-supplied upgrade utility.

Once the user is done with a node that has been allocated, it can be sent back to the pool for re-use.

As we have previously mentioned, Juju is designed to work with MAAS. In this case, MAAS becomes a sort of backend (resource pool) for Juju, or a "cloud provider" in Juju terminology. However, everything that was stated earlier still applies. For instance, if Juju removes a machine then MAAS will, in turn, release that machine to the pool.

CONCEPTS AND TERMS

At this point it is important to learn about some common terms that are essential to grasp in order to fully enjoy MAAS.

2.2.1. Nodes

A node is a general term that refers to multiple, more specific objects. Nodes are managed by MAAS through a life cycle, from adding and enlistment into MAAS, through commissioning, allocation and deployment. Nodes are then either released back into the pool of nodes or retired.

Nodes include Controllers, Machines and Devices.

- *Controllers*

There are two types of controllers: a region controller and a rack controller. The region controller deals with operator requests while one or more rack controllers provide the high-bandwidth services to multiple server racks, as typically found in a data centre.

A region controller consists of:

- REST API server (TCP port 5240)
- PostgreSQL database
- DNS
- caching HTTP proxy
- web UI

A region controller can be thought of as being responsible for a data centre, or a single region. Multiple fabrics are used by MAAS to accommodate subdivisions within a single region, such as multiple floors in a data centre.

A rack controller provides:

- DHCP
- TFTP
- HTTP (for images)
- iSCSI
- Power management

A rack controller is attached to each "fabric" (VLAN namespace mechanism). As the name implies, a common setup is to have a rack controller in each data centre server rack. The rack controller will cache large items for performance, such as operating system install images, but maintains no exclusive state other than the credentials required to talk to the region controller.

Both the region controller and the rack controller can be scaled-out as well as made highly available. A *tag* (not to be confused with VLAN tags) that are user-created and associated with nodes based on their physical properties. These can then be used to identify nodes with particular abilities which can be useful during the deployment of services.

- *Machines*

A *machine* is a node that can be deployed by MAAS.

- *Devices*

A *device* is a non-deployable node. This entity can be used to track routers, for example.

Devices can be assigned IP addresses (static or dynamic) and DNS names. They can also be assigned a parent node and will be automatically deleted (along with all the IP address reservations associated with it) when the parent node is deleted or released. This is designed to model and manage the virtual machines or containers running inside a MAAS-deployed node.

2.2.2. Zones

A *physical zone*, or just *zone*, is an organizational unit that contains nodes where each node is in one, and only one, zone. Later, while in production, a node can be taken (allocated) from a specific zone (or not from a specific zone). Since zones, by nature, are custom-designed (with the exception of the 'default' zone), they provide more flexibility than a similar feature offered by a public cloud service (ex: availability zones).

Some prime examples of how zones can be put to use include fault-tolerance, service performance, and power management:

Fault tolerance: Fault tolerance is "the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components". To assist with this, multiple MAAS zones can be employed. For this, a zone can be defined in different ways. It can be based on power supply for instance, or it can represent a portion of your network or an entire data centre location. Machines that work in tandem in order to provide an instance of a service should be allocated in the same zone. The entire service should be replicated in another zone.

- Service performance: Service performance is the ability of your service to operate in the most efficient manner possible where the typical criteria used is speed. Multiple MAAS zones can be used to help. Nodes should be allocated in the zone closest to the performance-critical resources they need. For example, for applications that are highly sensitive to network latency, it may make sense to design a network topology consisting of several smaller networks, and have each of those represented as a zone. The zones can then be used to allocate nodes that have the best performance depending on the service offered.

- Power management: Power management is concerned with power usage density and cooling. This topic can be addressed with the use of several MAAS zones. Nodes can be distributed in such a way that power-hungry and/or "hot" systems are located in different zones. This can help mitigate power consumption and heat problems.

A newly installed MAAS comes with a default zone, and unless a new zone is created all nodes get placed within it. The user can therefore safely ignore the entire concept if its not interested in leveraging zones. The 'default' zone cannot be removed and its name cannot be edited.

2.2.3. Images

An *image* is used to provision an OS to a MAAS machine. MAAS images are imported based on what series have been selected. This is typically done once the install of MAAS is complete. MAAS only becomes functional once images have been imported from the boot source.

2.2.4. Subnets

A *subnet* is a "layer 3" network. It is defined by a network address and a network mask length (in bits) and is usually written in "CIDR" format. MAAS supports IPv4 and IPv6 subnets. In MAAS, a subnet is always associated with a single space.

Subnets can be grouped by fabrics or spaces:

Fabrics: A *fabric* could be described as a VLAN namespace mechanism. It's a switch or a combination of switches that use trunking to provide the same VLANs, in other words is a set of consistent interconnected VLANs that are capable of mutual communication. A default fabric ('fabric-0') is created for each detected subnet when MAAS is installed. The following conceptual diagram shows two fabrics in the same data centre or region, each using distinct VLAN ranges and their associated subnets:

- Spaces: A *space* is a logical grouping of subnets that are able to communicate with each other. Subnets within each space need not belong to the same fabric. In other words is a grouping of networks (VLANs and their subnets) that are able to mutually communicate with each other (subnets within a space do not need to belong to the same fabric).

  A default space ('space-0') is created when MAAS is installed and includes all detected subnets.

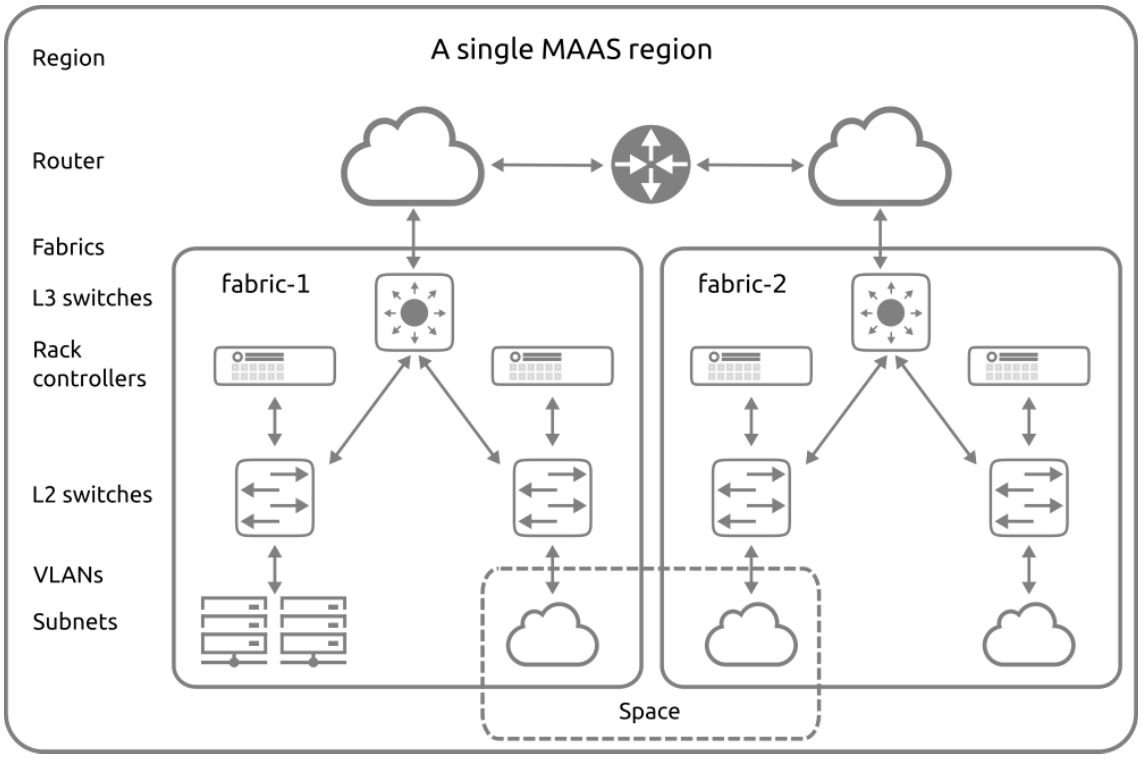


Figure 8: MAAS architecture scheme

2.2.4.1. IP ranges

IP addresses can be reserved by adding one or more *reserved ranges* to the subnet configuration. There are two types of ranges that can be defined:

- Reserved range: An IP range that MAAS will never use. It can be used for infrastructure systems, network hardware, external DHCP or the namespace for an OpenStack cloud being built.

Mode operates differently depending on whether the subnet is managed or unmanaged:

   - Managed (subnet): MAAS will never assign IP addresses inside this range. They can be used for anything (e.g. infrastructure systems, network hardware, external DHCP, or the namespace for an OpenStack cloud being built).

   - Unmanaged (subnet): MAAS will only assign IP addresses inside this range.

- Reserved dynamic range: An IP range that MAAS will use for enlisting, commissioning and (if MAAS-managed DHCP is enabled on the node's VLAN during commissioning) deploying. An initial range is created as part of the DHCP enablement process if done with the web UI.

2.2.4.2. VLANs

VLANs (Virtual LANs) are a common way to create logically separate networks using the same physical infrastructure.

Managed switches can assign VLANs to each port in either a "tagged" or an "untagged" manner. A VLAN is said to be "untagged" on a particular port when it is the default VLAN for that port, and requires no special configuration in order to access it.

"Tagged" VLANs can also be used with nodes in MAAS. That is, if a switch port is configured such that "tagged" VLAN frames can be sent and received by a MAAS node, that MAAS node can be configured to automatically bring up VLAN interfaces, so that the deployed node can make use of them.

A "Default VLAN" is created for every fabric, to which every new VLAN-aware object in the fabric will be associated with by default (unless specified otherwise).

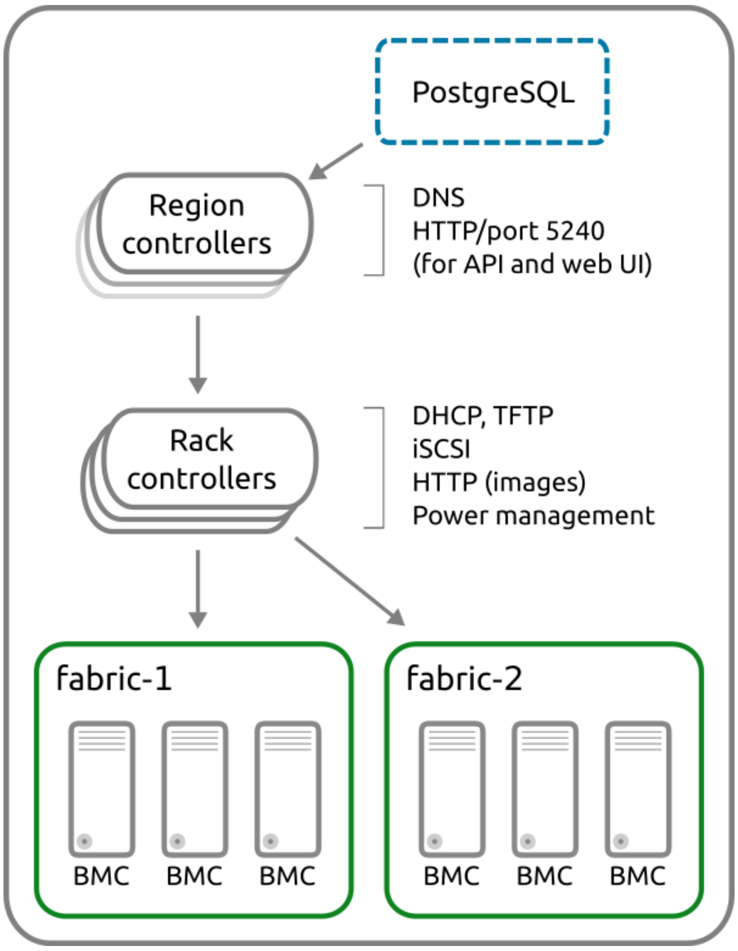


Figure 9: Architectural overview

2.3. Juju

Juju is an open source application modelling tool developed by Canonical Ltd. Juju focuses on reducing the operation overhead of today's software by facilitating quickly deploying, configuring, scaling, integrating, and performing operational tasks on a wide choice of public and private cloud services along with bare metal servers and local container based deployments.

- ***Modeling complex software topologies***

Juju's mission is to provide a modeling language for users that abstracts the specifics of operating complex big software topologies. Doing so reduces the cost of operations and provides flexibility. A Juju Model is an environment to manage and operate a set of software applications. Models can be operated on many clouds: Amazon Web Services, Microsoft Azure, Google Compute Engine, OpenStack, etc. A Juju Controller is the service that tracks the events, state, and user activity across multiple models. You can think of it a bit like a database servers where the controller is the server and models are different databases available on that server. Each model can have different configuration, sets of operating software, and different users with access at various levels. Some examples of models could be a web application, load balancer, and database in a "web-app" model. By using models deployments can be isolated into logical solutions and managed separately.

Juju can also be used to operate software on bare-metal servers by using Canonical's Metal as a Service, in containers using LXD, and more. Juju models provide an abstraction with allows the operations know-how to be cloud agnostic. This means that Charms and Bundles can help operate the same software with the same tooling on a public cloud, private cloud, or a local laptop.

- ***Charms***

Juju has two components: a client and a bootstrap node. Currently clients exist for Ubuntu, CentOS, Mac and Windows. After installing the client, one or more environments can be bootstrapped. Juju environments can be bootstrapped on many clouds: Amazon Web Services, HP Cloud Services, Microsoft Azure, OpenStack, *etc.* By creating a Juju Provider, additional cloud environments can be supported. Juju can also be bootstrapped on bare-metal servers. Large deployments can use Canonical's Metal as a Service. Small deployments can use the manual provider, which allows any SSH Ubuntu machine to be converted into a Juju-managed machine. Juju can also be installed on a local Ubuntu machine via LXC operating system–level virtualization and the local provider.

- ***Command line and GUI***

Juju has both a command line and a GUI. The GUI allows users to visually see what software is currently running in which models in a very clear fashion. It also gives the user the power to search the Charmstore and browse results with detailed Charm information presented. It also allows the deployment of complex software stacks via drag-and-drop. The Juju GUI is automatically available on every controller.

- ***Bundles***

Juju also has a concept of Bundles. A Bundle is a portable specification for a model with charms, configuration, and relations all specified in a declarative YAML format. A Bundle YAML file can later be imported into another Juju model and shared with others. Bundles can also be uploaded to the charm store, allowing anybody to deploy a bundle via drag-and-drop or one command. In this example bundle, two applications are modeled; mediawiki, and mysql. Users can modify attributes declared in the bundle to customize their deployment:

Juju also has a concept of Bundles. A Bundle is a portable specification for a model with charms, configuration, and relations all specified in a declarative YAML format. A Bundle YAML file can later be imported into another Juju model and shared with others. Bundles can also be uploaded to the charm store, allowing anybody to deploy a bundle via drag-and-drop or one command. In this example bundle, two applications are modeled; mediawiki, and mysql. Users can modify attributes declared in the bundle to customize their deployment:

```
services:
  mediawiki:
    charm: cs:trusty/mediawiki-3
    num_units: 1
    options:
      debug: false
      name: Please set name of wiki
      skin: vector
  mysql:
    charm: cs:trusty/mysql-29
    num_units: 1
    options:
      binlog-format: MIXED
      dataset-size: 80%
      tuning-level: safest
series: trusty
relations:
- - mediawiki:db
  - mysql:db
```

- ***Strenghts and weaknesses***

Juju is often compared to configuration management tools like Puppet, Chef, Ansible, *etc.*due to its software provisioning capabilities. This comparison however is not taking into account Juju's main strengths: instant integration and scaling. Juju allows services to be instantly integrated via relationships. By creating a relationship between, for instance, MySQL and WordPress, MySQL will share with WordPress any IPs, user, password and other configuration items. This will enable WordPress to create tables and import data automatically. Relations allow the complexity of integrating services to be abstracted from the user.

Also all complexities regarding service scaling can be abstracted. Users just need to specify the number of units they want and scaling will happen automatically. The charm will be responsible for choosing the best strategy.

Juju is also often compared to Platform as a Service. Although in definition Juju is not a platform-as-a-Service, it can be used to run multiple PaaS on top of it. Juju can be seen as a DIY PaaS that allows PaaS platforms to run next to "legacy" software stacks in order to create a customized PaaS.

Juju lacks several of the more advanced features server provisioning systems like Chef, Puppet, Ansible, etc. have. However, Juju allows Charms to be written in any language. As such, Chef, Puppet, Ansible, *etc.* can be used inside a charm to do the server provisioning.

CONCEPTS AND TERMS

2.3.1. Cloud

To Juju, a *cloud* (or backing cloud) is a resource which provides machines (instances), and possibly storage, in order for application units to be deployed upon them. This includes public clouds such as Amazon Web Services, Google Compute Engine, and Microsoft Azure as well as private OpenStack-based clouds. Juju can also make use of environments which are not clouds per se, but which Juju can nonetheless treat as a cloud. MAAS and LXD fit into this last category.

2.3.2. Controller

The Juju *controller* is the initial cloud instance which is created in order for Juju to gain access to a cloud. It is created by having the Juju client contact the cloud's API. The controller is a central management node for the chosen cloud, taking care of all operations requested by the Juju client. Multiple clouds (and thus controllers) are possible and each one may contain multiple models and users.

2.3.3. 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.

The controller model is the management model and is intended to contain a single machine, the actual controller. All other models are considered regular and are used to run workloads.

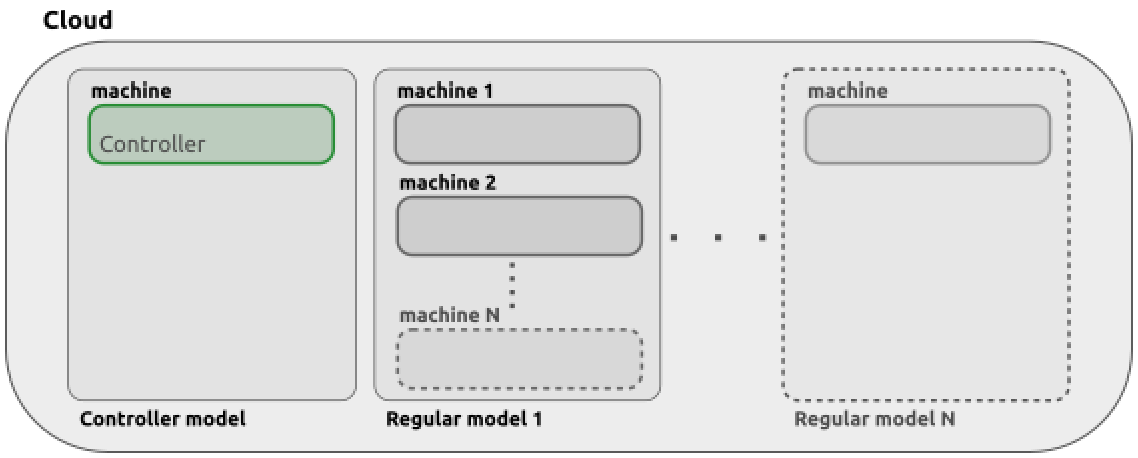


Figure 10: Juju models in a cloud

2.3.4. Charm

A Juju *charm* contains all the instructions necessary for deploying and configuring application units. Charms are publicly available in the online Charm Store and represent the distilled knowledge of experts. Charms make it easy to reliably and repeatedly deploy applications, then scale up (and down) as desired with minimal effort.

The simplest scenario is when a charm is deployed (by the Juju client) with the juju deploy command without any options to qualify the request. By default, a new instance will be created in the backing cloud and the application will be installed within it:

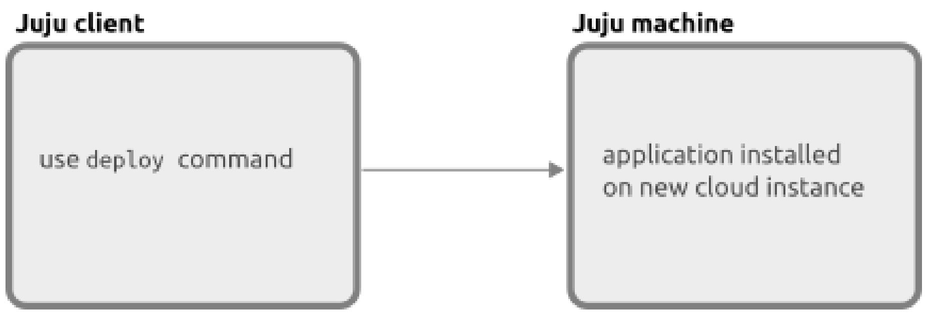


Figure 11: Juju charm deployment scenario

2.3.5. 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.

2.3.6. 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. If directed by the user a machine may house several units (e.g. to conserve resources) or possibly no units at all: a machine can be created independently of applications (juju add-machine), though usually this is with the intention of eventually running an application on it.
Represented below is a very standard Juju machine. It has a single deployed charm:

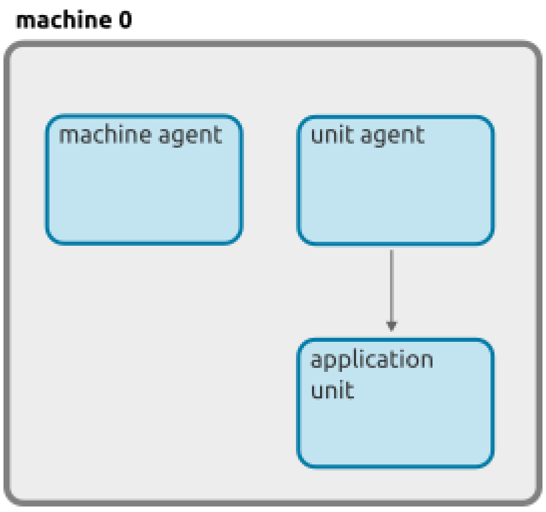


Figure 12: Juju machine scheme

Here we have a machine with a deployed charm in addition to a charm deployed on a LXD container within that machine:

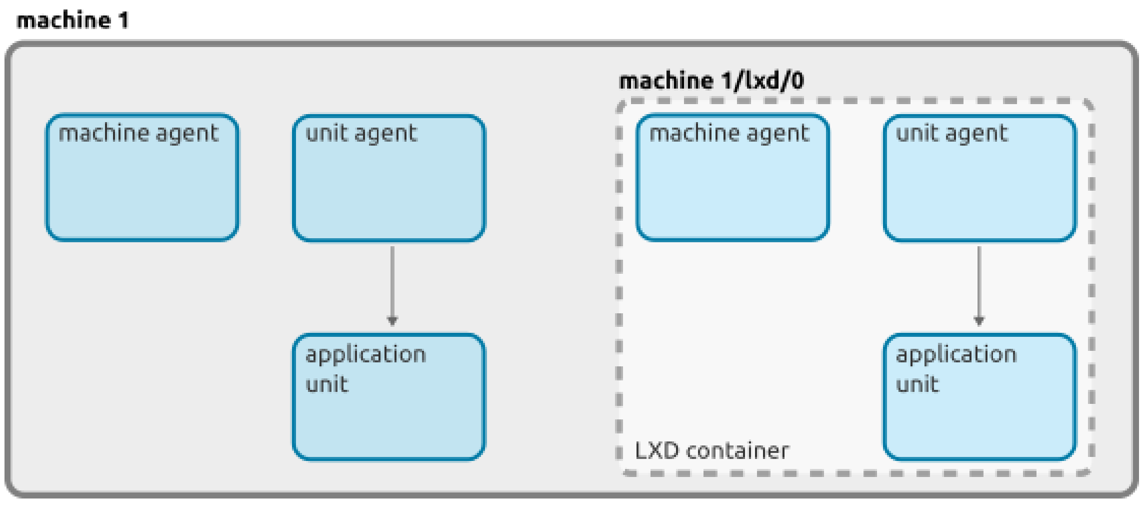


Figure 13: Juju machines comparison

2.3.7. 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.
For example, one may deploy a single MongoDB application, and specify that it should run three units (with one machine per unit), so that the replica set is resilient to failures. Internally, even though the replica set shares the same user-provided configuration, each unit may be performing different roles within the replica set, as defined by the charm.

The following diagram represents the scenario described above. For simplicity, the agents have been omitted:

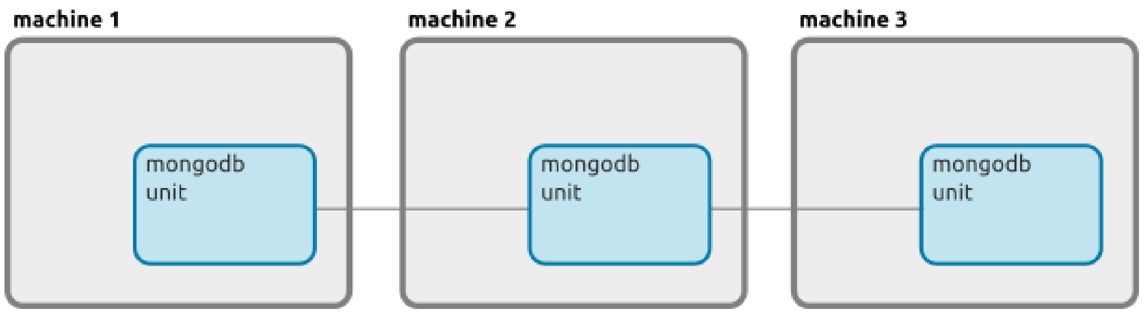


Figure 14: Juju Unit scenario

2.3.8. 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* by the collection of three properties: a *role*, a *name*, and an *interface*.
There are three types of roles:

- Requires: The endpoint can optionally make use of services represented by another charm's endpoint over the given interface.
- Provides: The endpoint represents a service that another charm's endpoint can make use of over the given interface.
- Peers: The endpoint can coexist with another charm's endpoint in a peer-to-peer manner (i.e. only between units of the same application). This role is often used in a cluster or high availability context.

For example, the pertinent excerpt of the *metadata.yaml* file for the 'wordpress' charm is as follows:

```
requires:
  db:
    interface: mysql
  nfs:
    interface: mount
  cache:
    interface: memcache
provides:
  website:
    interface: http
peers:
  loadbalancer:
    interface: reversenginx
```

Here, there are three 'requires' endpoints ('db', 'nfs', and 'cache'), one 'provides' endpoint ('website'), and one 'peers' endpoint ('loadbalancer'). For instance, we can say that "the 'db' endpoint can make use of services offered by another charm over the 'mysql' interface".

Despite the term 'requires', the three cited endpoints are not hard requirements for the 'wordpress' charm. You will need to read the charm's entry in the Charm Store (e.g. wordpress) to discover actual requirements as well as how the charm works. For instance, it is not obvious that the 'wordpress' charm comes bundled with an HTTP server (nginx), making a separate HTTP-based charm not strictly necessary.

2.3.9. Interface

An *interface* is the communication protocol used over a relation between applications. In the example shown in the Endpoint section, the interfaces for the corresponding endpoints are clearly discerned.

2.3.10. 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 (requires to provides, provides to requires, peers to peers).
For example, the 'wordpress' charm supports, among others, an 'http' interface ("provides" the website) and a 'mysql' interface ("requires" a database). Any other application which also has such interfaces can connect to this charm in a meaningful way.

Below we see WordPress with relations set up between both MySQL and Apache (a potential relation is shown with HAProxy):

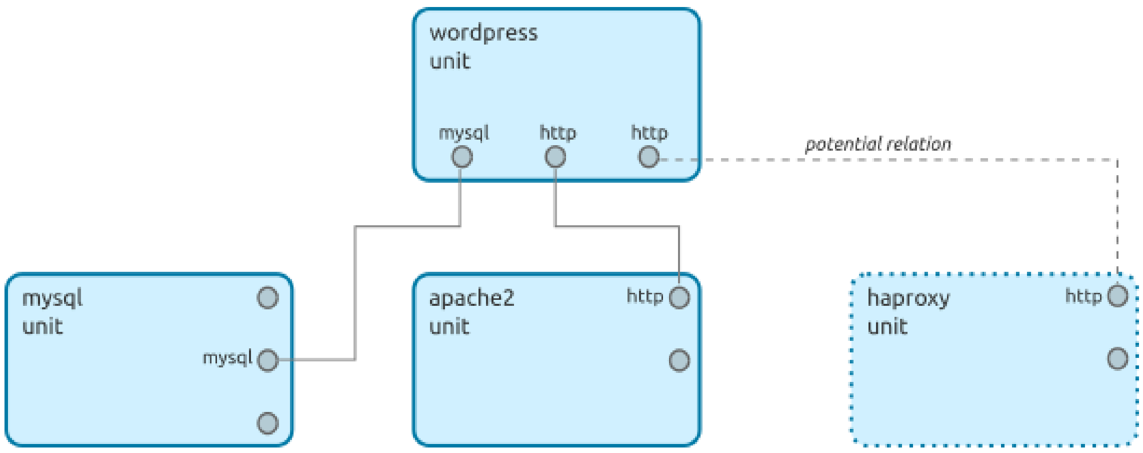


Figure 15: Juju relations diagram

2.3.11. Client

The Juju *client* is command line interface (CLI) software that is used to manage Juju, whether as an administrator or as a regular user. It is installed onto one's personal workstation. This software connects to Juju controllers and is used to issue commands that deploy and manage application units running on cloud instances.

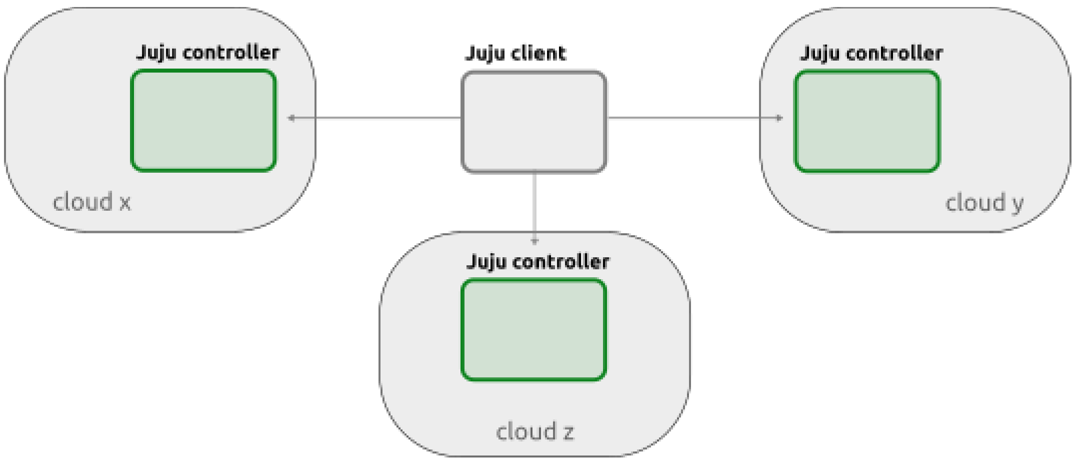


Figure 16: Juju client scenario

In the case of the localhost cloud (LXD), the cloud is housed within the same system as the Juju client:

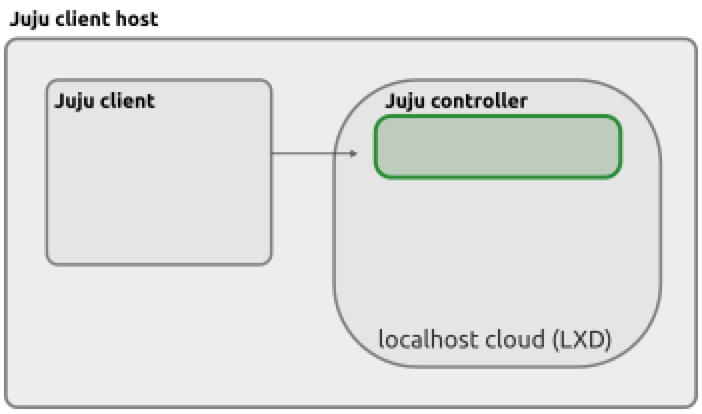


Figure 17: Juju client host

Although LXD itself can operate over the network, Juju does not support this. The client *must* be local to the LXD containers.

2.3.12. Agent

A Juju *agent* is software that runs on every Juju machine. There is a *machine agent* that operates at the machine level and a *unit agent* that works at the application unit level. Thus there are typically at least two agents running on each regular (non-controller) machine: one for the machine and one for a deployed application/charm. The controller normally has a single machine agent running.

A machine agent manages its respective unit agents as well as any containers that may be requested on that machine. In particular, it is the machine agent that creates the unit agent. The unit agents are responsible for all charm related tasks.

In general, all agents track state changes, respond to those changes, and pass updated information back to the controller. A model's status (juju status command) is built up from the communication between a controller and all the agents running in that model. Agents are also responsible for all logging that goes on in Juju.
The agent's software version is generally consistent across a controller (and its models) and is thus determined at controller-creation time. By default the agent uses the same version as that of the local Juju client but this can be tweaked if desired.

2.4. <u>Openstack</u>

The use of OpenStack is a key fact to achieve what is called orchestration of heterogeneous resources of a data center. Up to now, the provisioning of traditional data centers consisted of optimizing computational resources, without putting special attention to the network. Thus, the orchestration aims to solve this problem, in addition to making more efficient use of physical resources. The orchestrator must make the appropriate decisions taking into account the technological resources such as the location within the data center to coordinate the allocation of resources requested by the client. Thus, the use of orchestration allows having a dynamic data center, allowing to scale the desired infrastructure according to the requirements of an application. In addition, savings in time and energy are achieved due to its automation and centralized management.

In this way, an infrastructure such as OpenStack is required to achieve orchestrate the resources offered by a data center and thus achieve a cloud environment. The motivation to use a platform for management of cloud environments such as OpenStack is given to what is called open source and, in addition, the fact that it is one of the references in terms of the management of cloud environments in the market.

- ***General overview:***

OpenStack is a collection of open source software projects designed to work together to form the basis of a cloud. Primarily, it is used for private cloud implementations, but it can be just as applicable for cloud service providers to build public cloud resources. It's important to understand that OpenStack is not a single product, but rather a group of projects.

- ***Modular:***

From its inception, OpenStack was designed to be modular and to be integrated with additional tools and plugins via APIs. Any single project from OpenStack could be choosen to use to accomplish a particular task, or several of them, to build out a more complete cloud. Canonical integrates the projects, along with additional components, into a fully fledged enterprise Cloud Platform known as Ubuntu OpenStack.

- ***Core projects and more:***

The core projects of OpenStack consist of Nova (compute), Neutron (networking), Horizon (dashboard), Swift (object storage), Glance (image storage), and Keystone (identity). Beyond the core projects, there are additional solutions and tools in the industry to enhance the deployment, integration and daily operation of an OpenStack cloud. This core projects are going to be explained in more detail in the next pages.

CHALLENGES

- ***Hardware configuration:***

Most organisations still manage some hardware that after racking and connecting it, an initial configuration must be done. Some use vendor tools, some write proprietary scripts, others leverage ever-growing teams of people and also there are ones that use a combination of all of these approaches and more.

The issue with these approaches is economic scalability. If you change hardware configuration in any way, you need to pay to add/modify an ever-growing collection of scripts. If you change hardware vendor, you need to add, configure and maintain a new tool, while maintaining all previous hardware management tools. If you add more servers, you have to hire more people. None of this scales with cloud economics.

- ***Hardware integration:***

Beyond the initial configuration, integration must happen. Network services must be set up and maintained, including DHCP or static IP address pools for the host NICs, DNS entries, VLANs, etc. Again, these integration tasks can be accomplished with scripts, vendor tools or personnel, but the same potential issues arise as with configuration.

- ***Openstack installation:***

Another major obstacle to OpenStack success is the initial installation. The aforementioned scripting approach is common, as are growing teams of expensive personnel.

There are also OpenStack projects to perform installation, but they are often vendor-driven, not neutral and lack feature completeness.

Organisations that try to use them often find themselves doing significant, ongoing development work to make the project useful.

- ***Additional challenges:***

On-going challenges all lend to increasing cost and decreasing economic scalability.

Additional considerations include:

- Upgrades
- Rebuilding
- New clouds
- Repeatable best practices
- Scaling out
- Reducing cost of consultants

- ***A scalable, practical approach:***

A better and easier approach are vendor hardware and platform neutral tools.

Tools that include APIs for automation of not just software, but your datacenter, as well.

Tools with graphical interfaces, designed with scalable cloud economics in mind.

Putting the intelligence of installation and integration complexity directly into the tools themselves is how you make OpenStack easy and achieve economic scalability.

OpenStack installation and integration challenges are best solved by a thoughtful approach, using technologies designed for modern clouds. Legacy scripting technologies might work now, but likely won’t scale as your cloud’s needs change and grow. The same goes for personnel.

COMPONENTS

From the point of view of software, OpenStack is a free software collection projects maintained by the community which include several components. Through these services, OpenStack pro- vides a complete operating platform for administration and management of clouds. The most important components of OpenStack are explained in more detail:

2.4.1. Compute (Nova)

OpenStack Compute (Nova) is a cloud computing fabric controller, which is the main part of an IaaS system. It is designed to manage and automate pools of computer resources and can work with widely available virtualization technologies, as well as bare metal and high-performance computing (HPC) configurations. **KVM**, VMware, and Xen are available choices for hypervisor technology (virtual machine monitor), together with Hyper-V and Linux container technology such as LXC.

It is written in Python and uses many external libraries such as Eventlet (for concurrent programming), Kombu (for AMQP communication), and SQLAlchemy (for database access). Compute's architecture is designed to scale horizontally on standard hardware with no proprietary hardware or software requirements and provide the ability to integrate with legacy systems and third-party technologies.

Due to its widespread integration into enterprise-level infrastructures, monitoring OpenStack performance in general, and Nova performance in particular, at scale has become an increasingly important issue. Monitoring end-to-end performance requires tracking metrics from Nova, Keystone, Neutron, Cinder, Swift and other services, in addition to monitoring RabbitMQ which is used by OpenStack services for message passing. All these services generate their own log files, which, especially in enterprise-level infrastructures, also should be monitored.

2.4.2. Networking (Neutron)

OpenStack Networking (Neutron) is a system for managing networks and IP addresses. OpenStack Networking ensures the network is not a bottleneck or limiting factor in a cloud deploymen and gives users self-service ability, even over network configurations.

OpenStack Networking provides networking models for different applications or user groups. Standard models include flat networks or VLANs that separate servers and traffic. OpenStack Networking manages IP addresses, allowing for dedicated static IP addresses or DHCP. Floating IP addresses let traffic be dynamically rerouted to any resources in the IT infrastructure, so users can redirect traffic during maintenance or in case of a failure.

Users can create their own networks, control traffic, and connect servers and devices to one or more networks. Administrators can use software-defined networking (SDN) technologies like OpenFlow to support high levels of multi-tenancy and massive scale. OpenStack networking provides an extension framework that can deploy and manage additional network services such as intrusion detection systems(IDS), load balancing, firewalls, and virtual private networks (VPN).

2.4.3. Block storage (Cinder)

OpenStack Block Storage (Cinder) provides persistent block-level storage devices for use with OpenStack compute instances. The block storage system manages the creation, attaching and detaching of the block devices to servers. Block storage volumes are fully integrated into OpenStack Compute and the Dashboard allowing for cloud users to manage their own storage needs. In addition to local Linux server storage, it can use storage platforms including Ceph, CloudByte, Coraid, EMC (ScaleIO, VMAX, VNX and XtremIO), GlusterFS, Hitachi Data Systems, IBM Storage (IBM DS8000, Storwize family, SAN Volume Controller, XIV Storage System, and GPFS), Linux LIO, NetApp, Nexenta, Nimble Storage, Scality, SolidFire, HP (StoreVirtual and 3PAR StoreServ families) and Pure Storage. Block storage is appropriate for performance sensitive scenarios such as database storage, expandable file systems, or providing a server with access to raw block level storage. Snapshot management provides powerful functionality for backing up data stored on block storage volumes. Snapshots can be restored or used to create a new block storage volume.

2.4.4. Identity (Keystone)

OpenStack Identity (Keystone) provides a central directory of users mapped to the OpenStack services they can access. It acts as a common authentication system across the cloud operating system and can integrate with existing backend directory services like LDAP. It supports multiple forms of authentication including standard username and password credentials, token-based systems and AWS-style (i.e. Amazon Web Services) logins. Additionally, the catalog provides a queryable list of all of the services deployed in an OpenStack cloud in a single registry. Users and third-party tools can programmatically determine which resources they can access.

2.4.5. Image (Glance)

OpenStack Image (Glance) provides discovery, registration, and delivery services for disk and server images. Stored images can be used as a template. It can also be used to store and catalog an unlimited number of backups. The Image Service can store disk and server images in a variety of back-ends, including Swift. The Image Service API provides a standard REST interface for querying information about disk images and lets clients stream the images to new servers.

Glance adds many enhancements to existing legacy infrastructures. For example, if integrated with VMware, Glance introduces advanced features to the vSphere family such as vMotion, high availability and dynamic resource scheduling (DRS). vMotion is the live migration of a running VM, from one physical server to another, without service interruption. Thus, it enables a dynamic and automated self-optimizing datacenter, allowing hardware maintenance for the underperforming servers without downtimes.

Other OpenStack modules that need to interact with Images, for example Heat, must communicate with the images metadata through Glance. Also, Nova can present information about the images, and configure a variation on an image to produce an instance. However, Glance is the only module that can add, delete, share, or duplicate images

2.4.6. Object storage (Swift)

OpenStack Object Storage (Swift) is a scalable redundant storage system. Objects and files are written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster. Storage clusters scale horizontally simply by adding new servers. Should a server or hard drive fail, OpenStack replicates its content from other active nodes to new locations in the cluster. Because OpenStack uses software logic to ensure data replication and distribution across different devices, inexpensive commodity hard drives and servers can be used.

In August 2009, Rackspace started the development of the precursor to OpenStack Object Storage, as a complete replacement for the *Cloud Files* product. The initial development team consisted of nine developers.SwiftStack, an object storage software company, is currently the leading developer for Swift with significant contributions from HP, Red Hat, NTT, NEC, IBM and more.

2.4.7. Dashboard (Horizon)

OpenStack Dashboard (Horizon) provides administrators and users with a graphical interface to access, provision, and automate deployment of cloud-based resources. The design accommodates third party products and services, such as billing, monitoring, and additional management tools. The dashboard is also brand-able for service providers and other commercial vendors who want to make use of it. The dashboard is one of several ways users can interact with OpenStack resources. Developers can automate access or build tools to manage resources using the native OpenStack API or the EC2 compatibility API.

2.4.8. Orchestration (Heat)

Heat is a service to orchestrate multiple composite cloud applications using templates, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.

The table below shows the components and their description:

Service	Project Name	Description
Dashboard	**Horizon**	Computational resources on demand management
Compute	**Nova**	Computational resources on demand management
Networking	**Neutron**	Networking automation
	Storage	
Object Storage	**Swift**	Storing objects in distributed environment
Block Storage	**Cinder**	Management volumes to storage blocks (IBM, EMC, HP, Red Hat / Gluster, Ceph / RBD, NetApp, SolidFire and Mexenta)
	Shared Services	
Identity Service	**Keystone**	Control images of O.S
Image Service	**Glance**	Authentication and authorization control
Telemetry	**Ceilometer**	Measurement and monitoring features
	Higher-level services	
Orchestration	**Heat**	Service orchestration
Database Service	**Trove**	Database as a service

Table 1: Openstack Components

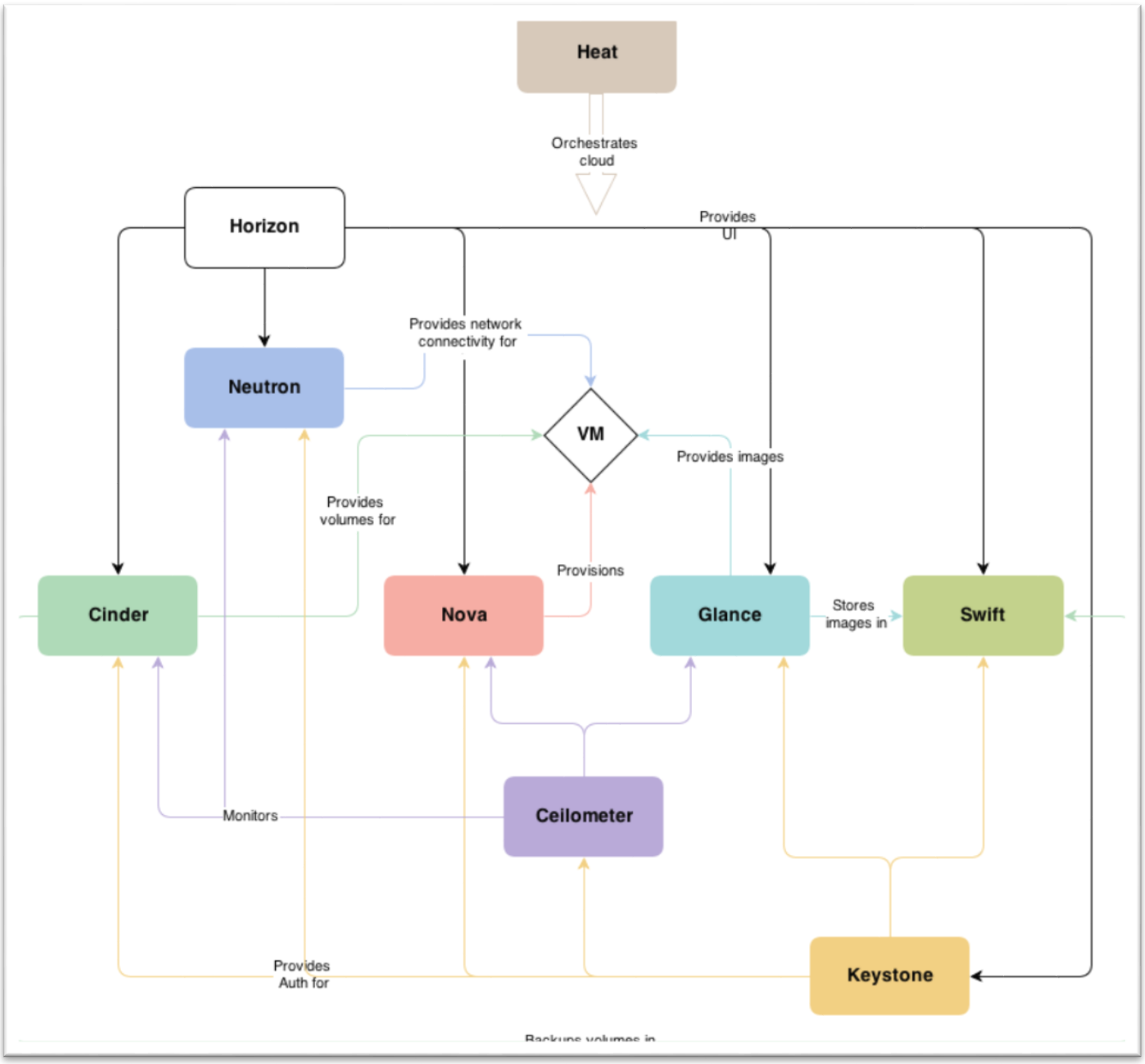


Figure 18: Openstack architecture

- ***How it Works?***

Depending on the installed components and their distribution, the process of initiating an instance will be more or less complex, but we can summarize it in the following steps:

1) The user authenticates to Keystone, either directly or through the Horizon web interface, obtains a session token that will allow it to perform actions with the rest of the OpenStack components without having to authenticate again. These actions will be limited by the role permissions that the user has in Keystone.

2) The user requests to Glance the list of available images. These images can be hosted by the Swift module or directly on the user's computer.

3) After selecting an image and specifying the characteristics of the instance (flavors), the user asks New to instantiate it. New automatically chooses the best cloud node (server) to execute the instance if the user does not specify it.

4) Finally, Neutron is in charge of configuring the virtual network for the instance.

3. Methodology / project development:

The Methodology is included in this chapter and embrace all relevant methods that were utilized as well as research methods and measurements, software and hardware development

3.1. Environment

The aim of this section is to describe the existing network infrastructure on top of the which the project was built: an OpenFlow based campus network that serves as a test bed for a series of research algorithms.

The network communicates with the Internet via a legacy Layer 2 VLAN in order to avoid a direct connection to the gateway router that is not physically available.

The SDN network is implemented in a building of the University campus in parallel with the existing infrastructure. There is a layer 2 network in the building that connects to the Internet via a L3 gateway. The L2 network is divided in two different VLANs, one that we will call the DEE VLAN that has the subnet 172.16.4.0/22 and another VLAN, that we call Staff VLAN with the IP subnet 10.164.24.0/22. The gateway of the DEE VLAN routes traffic to the Internet trough the Staff VLAN (that has the physical connection to the L3 Gateway of the building). The figure below illustrates the existing L2 networks, how they interconnect at the DEE VLAN gateway and where the SDN network its placed.

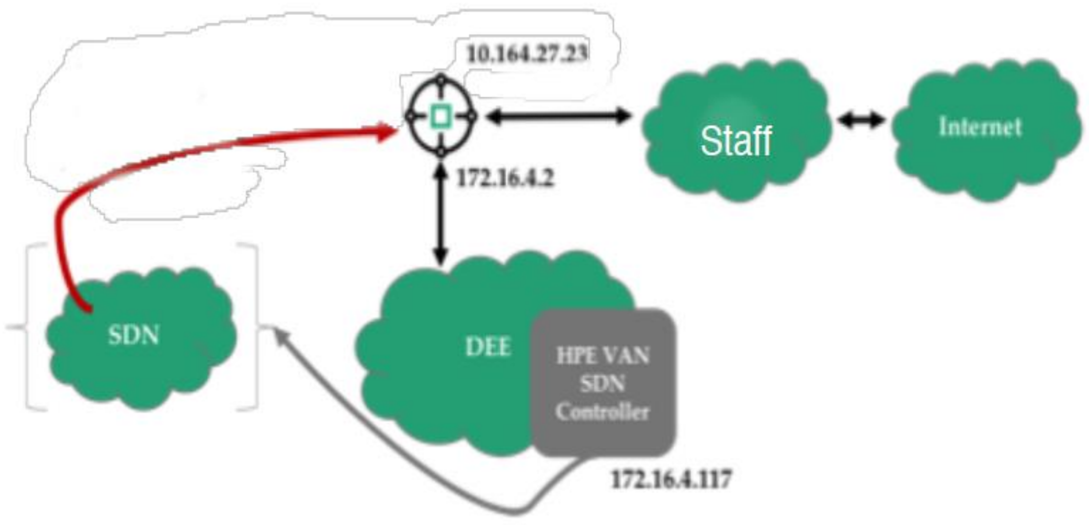


Figure 19: The SDN network and existing infrastructure

The SDN network serves an entire floor of the building providing network access to students in the classrooms and labs of that floor in parallel with the rest of the infrastructure. It also connects to the Internet via the Staff VLAN since no direct physical connection to the L3 gateway of the building is available.

The control network that connects the SDN controller to the OpenFlow switches is the mentioned DEE VLAN. That DEE VLAN has a few ports in two rooms used by students preparing their MSc and Ph.D. thesis and serves as the legacy network providing Internet access via the Staff VLAN.

In terms of security it exist a risk higher than using a dedicated and isolated control network, but it provides a ready control infrastructure with easy access to the students that conduct experiments in the network. It also opens possibilities to expose the control network to the outside world for security studies and experiments since the DEE VLAN is connected to the Internet via a NAT router/firewall that routes Internet traffic to the staff VLAN.

The access from the Internet to hosts inside of the DEE VLAN is possible if that NAT router /firewall is configured to provide public access to a specific DEE VLAN host. Allowing this for the controller server poses a severe security risk, but since the network is used for research and experimentation it was interest of the designers to expose the controller to outside attack vectors.

Wireless access hosts have their IP provided by DHCP servers running in the APs.

There are two D-Link DAP 2690 dual band PoE Access Points that provide WiFi access to undergraduate students and are connected to one HP E3800-24G-POE+-2ESFP access switch (HP-2). This switch also connects to three HPE ProLiant DL 160 Gen9 servers where the controller and radius server are installed. This access switch is then connected to another switch (HP-1) that provides access ports for two research workrooms and serves as the gateway of the SDN VLAN and connects to the Staff VLAN for Internet connectivity.

The network has the topology depicted in the figure in the next page.

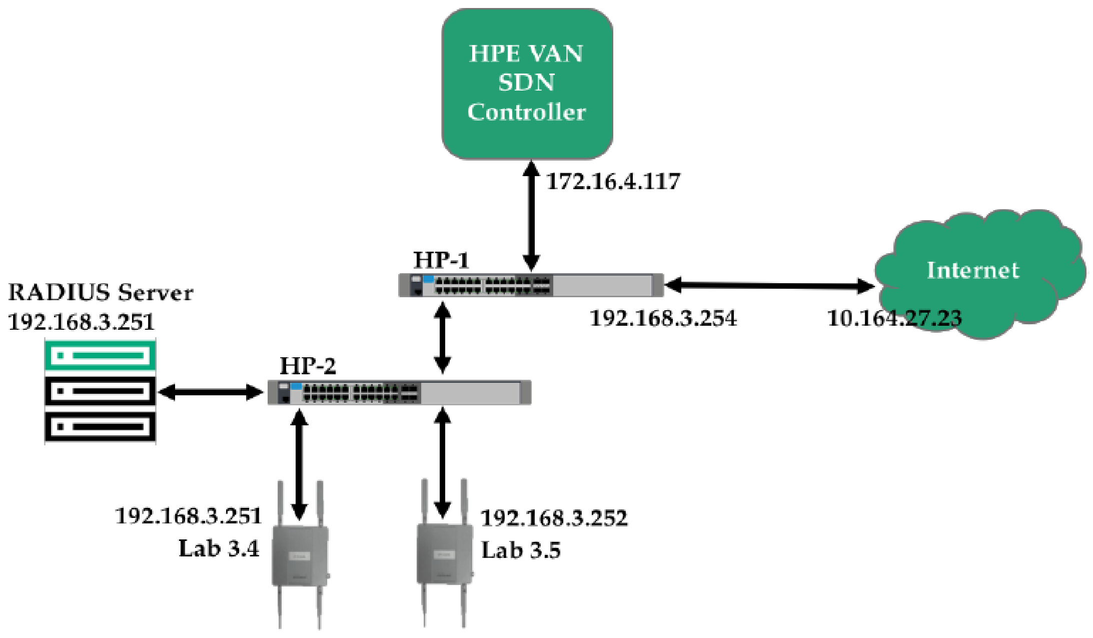


Figure 20: Network topology

3.2. Development

In this section I am going to present all the documentation that has been made for the OpenStack install on actual hardware using MAAS in the Laboratório de Apoio a Dissertação of the Departamento de Engenharia Electrotécnica da Faculdade de Ciências e Tecnologia (FCT). I'll be using the Ubuntu server relase and the landscape OpenStack installer.

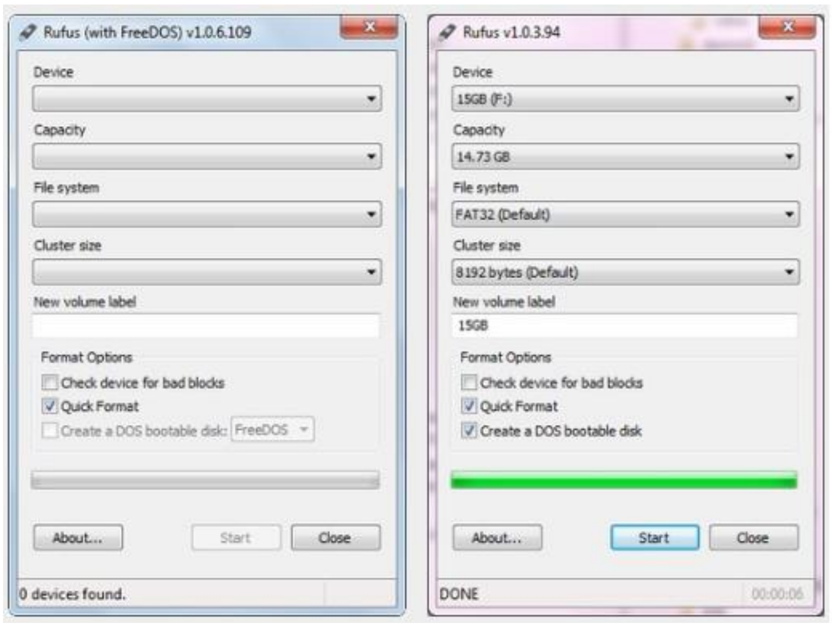
Here I will be going through all the steps I took to create the cloud, giving everything from hardware details to configuration file changes and also ilustrationg with screenshots taken as well.

3.2.1. Ubuntu Linux bootable usb creation

First step is to create a Ubuntu server bootable flash drive for the server install. Afterwards with a bootable Ubuntu USB stick a user can:

- Install or upgrade Ubuntu
- Test out the Ubuntu desktop experience without touching your PC configuration
- Boot into Ubuntu on a borrowed machine or from an internet cafe
- Use tools installed by default on the USB stick to repair or fix a broken configuration

At www.ubuntu.com the Ubuntu Server 16.10 can be found, I took it and installed Ubuntu in a PC from the lab through an USB flash drive. To insert the mentioned OS there, I used Rufus 2.18 as a universal USB installer going to their webside.


Rufus (with FreeDOS) v1.0.6.109

Device:
Capacity:
File system:
Cluster size:
New volume label:

Format Options
- [ ] Check device for bad blocks
- [✓] Quick Format
- [ ] Create a DOS bootable disk: FreeDOS

0 devices found.

Rufus v1.0.3.94

Device: 15GB (F:)
Capacity: 14.73 GB
File system: FAT32 (Default)
Cluster size: 8192 bytes (Default)
New volume label: 15GB

Format Options
- [ ] Check device for bad blocks
- [✓] Quick Format
- [✓] Create a DOS bootable disk

DONE 00:00:06

Figure 21: Rufus installation process

Rufus is a small application that will allow to convert a normal and current pendrive, into a USB boot from which to start directly in DOS. Setting up the application is as simple as entering the device you want to format, marking the options you want and clicking on 'start'. In a few seconds the new 'DOS boot disk' ready.

3.2.2. Ubuntu Linux server installation

Here is where the standard install of Ubuntu server is done. This server will be used for MAAS (Metal as a service) afterwards. The hardware we are using for the MAAS server is a HPE ProLiant DL 160 Gen9 and it has 2 Ethernet cards to create a "bridge" to allow us access to both public and private networks. Open SSH was also installed so we can get to this box via our SSH client. After the login its done, typing the command line *ifconfig -a* it shows all the Ethernet interfaces.

```
toni@ubu:~$ ifconfig -a
eno1  Link encap:Ethernet  HWaddr 50:65:f3:61:bb:72
      inet addr:172.16.4.50  Bcast:172.16.7.255
      Mask:255.255.252.0
      inet6 addr: fe80::5265:f3ff:fe61:bb72/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:341146 errors:0 dropped:0 overruns:0 frame:0
      TX packets:184773 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:497059977 (497.0 MB)  TX bytes:14429815 (14.4 MB)

eno2  Link encap:Ethernet  HWaddr 50:65:f3:61:bb:73
      BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:0 errors:0 dropped:0 overruns:0 frame:0
      TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:0 (0.0 B)  TX bytes:0 (0.0 KB)

lo    Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      inet6 addr: ::1/128 Scope:Host
      UP LOOPBACK RUNNING  MTU:65536  Metric:1
      RX packets:1257065 errors:0 dropped:0 overruns:0 frame:0
      TX packets:1257065 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1
      RX bytes:2247092770 (2.2 GB)  TX bytes:2247092770 (2.2 GB)
```

In the screen above we can see that our system has multiple network interfaces in it: *eno1* and *eno2*. The first one its connected to a router that is connected to the internet, and it will be our primary source where we will download all required packages. On the other hand *eno2* will be connected to the MAAS region and used as our private network.

As *eno1* is connected to the internet we are going to use that as our primary soure so we can get out and download the packages it needs. Later on the next fields are required:

- *Hostname*: **ubu**
- *Full name for the new user:* **toni**

3.2.3. Ubuntu server MAAS install

In this section we are going to go through the initial install of the MAAS (Metal as a Service) software. We will update our Ubuntu install and install MAAS, then superuser will be added as well as kicked on the import of the boot images.

Updating repositories:

```
toni@ubu:~$ sudo apt update
```

Upgrading any packages available to get the most current versions of the software we have:

```
toni@ubu:~$ sudo apt upgrade
```

Upgrading the distribution:

```
toni@ubu:~$ sudo apt dist-upgrade
```

Installing these packages we enable add-apt-repository:

```
toni@ubu:~$ sudo apt-get install python-software-properties
```

```
toni@ubu:~$ sudo apt-get install software-properties-common
```

To continue the install at least 7 machines are required for JuJu to allow him to spread things out how he sees fit: 1 MAAS server and 6 nodes.

We also need to setup a private network with all machines plugged in, with the network divided into three logical ranges:

- *Dynamic range*: That has as many IPs as there are total NIC's connected to the network.
- *Static range*: That has as many IPs as there are machines connected to the network.
- *Floating IP range*: That has as many IPs as instances that we'll have in our cloud.

Then we add the newest version of the software that we are going to use, the stable juju repository:

```
toni@ubu:~$ sudo add-apt-repository ppa:juju/stable
```

To get the cloud going the first thing we need is Metal as a Service, that is the reason why we build this server. Doing that MAAS will be able to control all cloud nodes, starting and stopping them while keeping track of the software checking what its installed on what.

Adding stable MAAS repository:

```
toni@ubu:~$ sudo add-apt-repository ppa:maas-maintainers/stable
```

Adding the Openstack cloud stable repository:

```
toni@ubu:~$ sudo add-apt-repository ppa:cloud-installer/stable
```

Now we are ready to do the software install for MAAS. First of all we run our update again updating the repositories as we did before. Then we install MAAS package:

```
toni@ubu:~$ sudo apt install maas
```

Here we are able to access our MAAS install at this address, using the server IP address:

http://172.16.4.50/MAAS

The GUI will let the user set the superuser entering this command:

```
toni@ubu:~$ sudo maas-region-admin createadmin
```

At this point, MAAS tell us that we have to create a superuser, to make things easier, the same username as **toni** is entered.

To have a view tool we instaled *screen*. This is a program that allows you to have multiple screens on your server:

```
toni@ubu:~$ sudo apt install screen
```

It can be started just typing *screen*. Pressing CTRL+A then C we add a screen and with CTRL+A then P we can go to the next screen.

Installing network load monitor was also considered. This program lets you know if your interfaces have any activity:

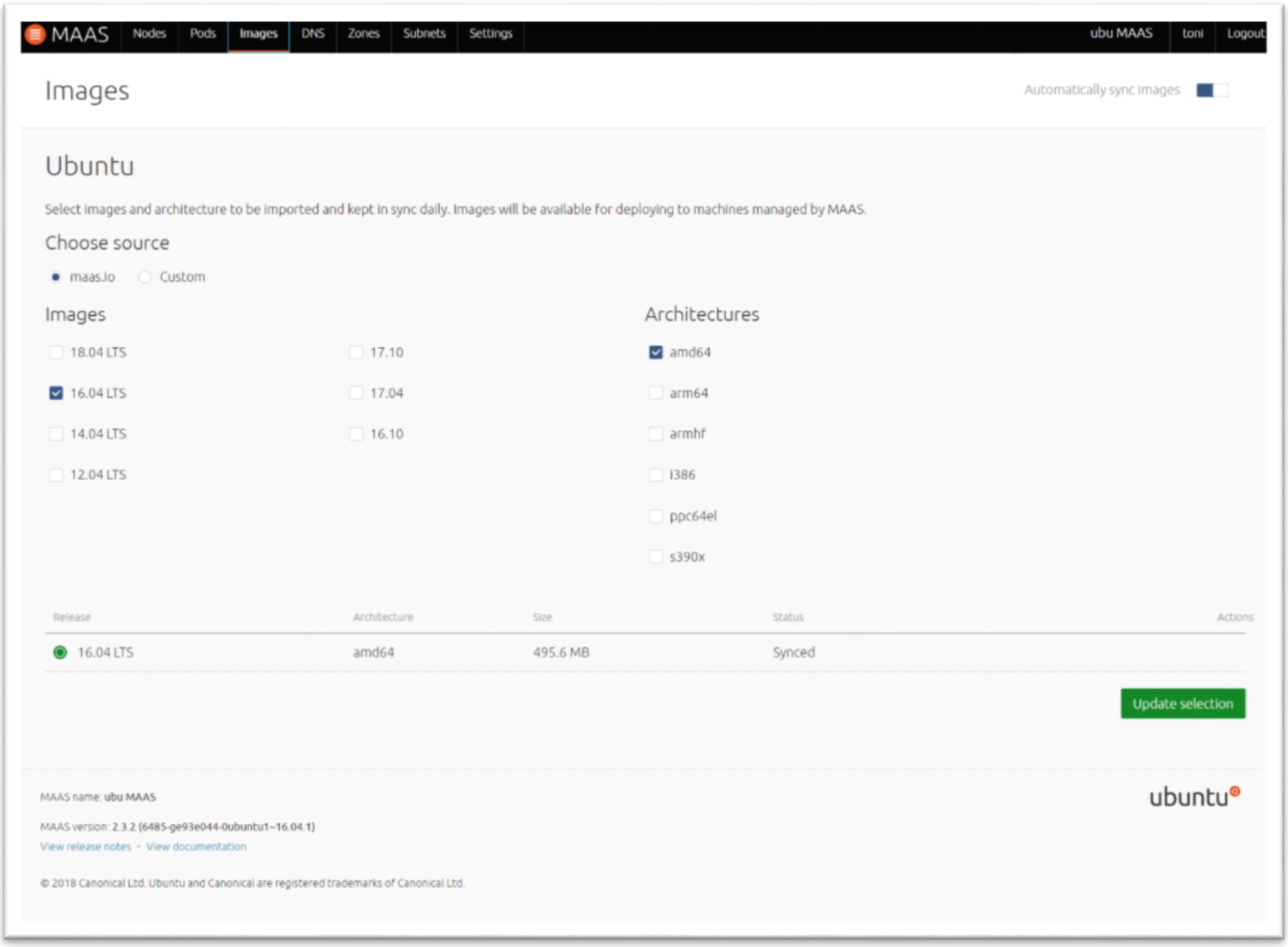
```
toni@ubu:~$ sudo apt install nload
```

Installing ATOP was done because it is a really handy tool for performance monitoring:

```
toni@ubu:~$ sudo apt install atop
```

At this point MAAS reflects there is no boot image available. Those images are boots of Ubuntu software that MAAS is going to store out in its repository and use them to boot up the MAAS nodes.

Now we have to kick the boot image downloaded off manually the first time.



Ubuntu

Select images and architecture to be imported and kept in sync daily. Images will be available for deploying to machines managed by MAAS.

Choose source

● maas.io ○ Custom

Images: ☐ 18.04 LTS, ☑ 16.04 LTS, ☐ 14.04 LTS, ☐ 12.04 LTS, ☐ 17.10, ☐ 17.04, ☐ 16.10

Architectures: ☑ amd64, ☐ arm64, ☐ armhf, ☐ i386, ☐ ppc64el, ☐ s390x

Release	Architecture	Size	Status	Actions
● 16.04 LTS	amd64	495.6 MB	Synced	

MAAS name: ubu MAAS
MAAS version: 2.3.2 (6485-ge93e044-0ubuntu1~16.04.1)
View release notes · View documentation
© 2018 Canonical Ltd. Ubuntu and Canonical are registered trademarks of Canonical Ltd.

Figure 22: MAAS gui Images tab

3.2.4. MAAS install network and etherwake

In this section we are going to talk about the network setup for MAAS and how etherwake plays a role in our setup.

We have to edit our interfaces file to config these interfaces as DHCP *eno1* is out internet facing NIC and *eno2* is our private network NIC.

Command to edit network config:

```
toni@ubu:~$ sudo nano /etc/network/interfaces
```

```
# This file describes the network interfaces available on your
# system and how to activate them.
# For more information, see interfaces.

# The loopback network interface
      auto lo
            iface lo inet loopback

# The primary network interface
      auto eno1
            iface eno1 inet dhcp

      auto eno2
            iface eno2 inet static
            address 10.1.1.100
            netmask 255.255.255.0
```

After this changes its better to reboot the machine to pick them up.

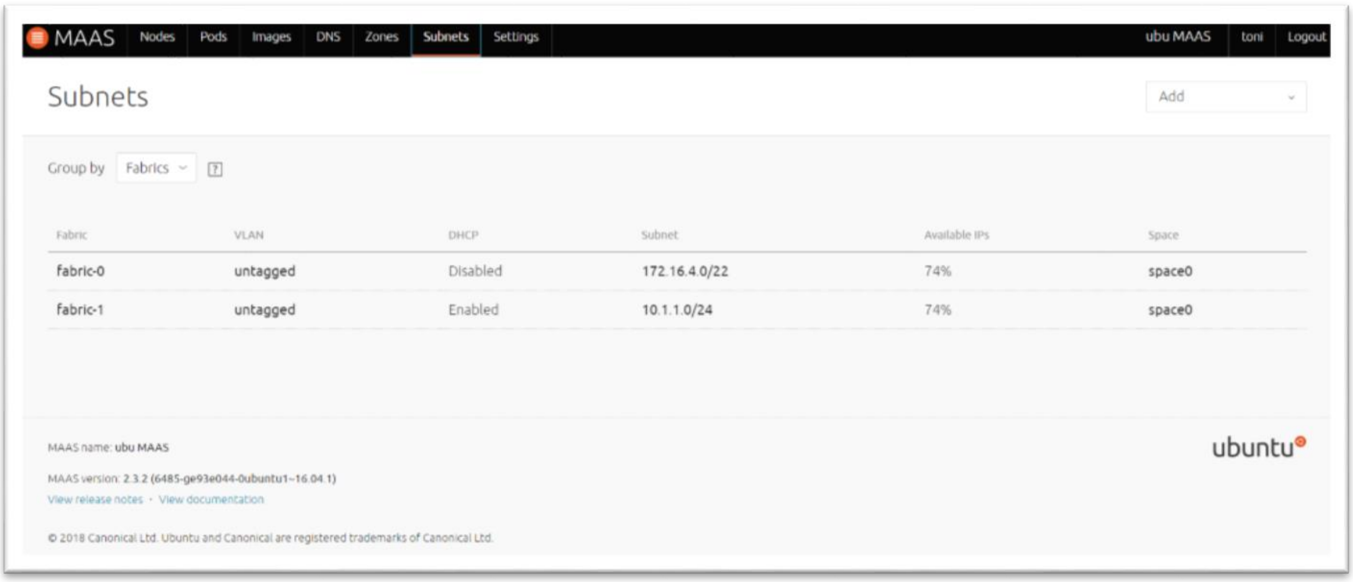
```
toni@ubu:~$ sudo reboot now
```

Now typing the proper *ifconfig –a* command linr we can see the following available interfaces:

```
toni@ubu:~$ ifconfig -a
eno1  Link encap:Ethernet  HWaddr 50:65:f3:61:bb:72
      inet addr:172.16.4.50  Bcast:172.16.7.255
      Mask:255.255.252.0
      inet6 addr: fe80::5265:f3ff:fe61:bb72/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:341146 errors:0 dropped:0 overruns:0 frame:0
      TX packets:184773 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:497059977 (497.0 MB)  TX bytes:14429815 (14.4 MB)
      Memory:92c00000-92cfffff

eno2  Link encap:Ethernet  HWaddr 50:65:f3:61:bb:73
      inet addr:10.1.1.100  Bcast:10.1.1.255  Mask:255.255.255.0
      inet6 addr: fe80::5265:f3ff:fe61:bb73/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:0 errors:0 dropped:0 overruns:0 frame:0
      TX packets:3052 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:0 (0.0 B)  TX bytes:701955 (701.9 KB)
      Memory:92b00000-92bfffff

lo    Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      inet6 addr: ::1/128 Scope:Host
      UP LOOPBACK RUNNING  MTU:65536  Metric:1
      RX packets:1257065 errors:0 dropped:0 overruns:0 frame:0
      TX packets:1257065 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1
      RX bytes:2247092770 (2.2 GB)  TX bytes:2247092770 (2.2 GB)
```

Our private and public networks have an IP now.

Now it is time to go to Subnets in the MAAS gui to setup both of our network interfaces.



Subnets

Group by: Fabrics

Fabric	VLAN	DHCP	Subnet	Available IPs	Space
fabric-0	untagged	Disabled	172.16.4.0/22	74%	space0
fabric-1	untagged	Enabled	10.1.1.0/24	74%	space0

MAAS name: ubu MAAS

MAAS version: 2.3.2 (6485-ge93e044-0ubuntu1~16.04.1)

View release notes · View documentation

© 2018 Canonical Ltd. Ubuntu and Canonical are registered trademarks of Canonical Ltd.

Figure 23: MAAS gui Subnets tab

Public network setup

First of all we setup the Gateway IP and the DNS IP, which will be the same:

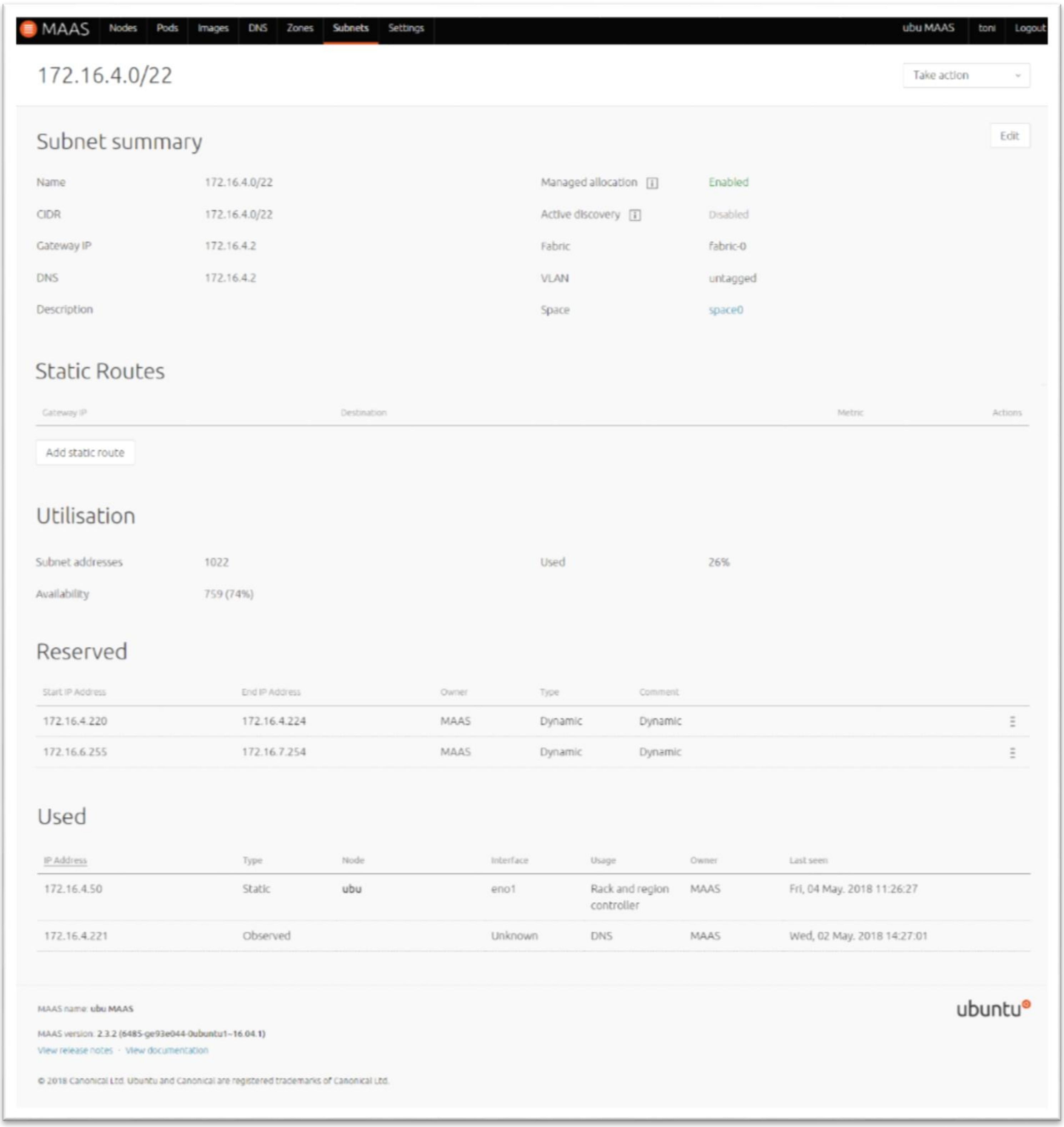
- Gateway IP: 172.16.4.2
- DNS: 172.16.4.2

We are going to reserve another dynamic range of IPs apart from the default one in order to serve as posible future utilities:

- Start IP Address A: 172.16.4.220
- End IP Address A: 172.16.4.224

- Start IP Address B: 172.16.4.255
- End IP Address B: 172.16.7.254

Right below we are showing how the MAAS gui looks like after having realized the previous configurations in the public network.



172.16.4.0/22

Subnet summary

Name	172.16.4.0/22	Managed allocation	Enabled
CIDR	172.16.4.0/22	Active discovery	Disabled
Gateway IP	172.16.4.2	Fabric	fabric-0
DNS	172.16.4.2	VLAN	untagged
Description		Space	space0

Static Routes

Gateway IP	Destination	Metric	Actions

Utilisation

Subnet addresses	1022	Used	26%
Availability	759 (74%)		

Reserved

Start IP Address	End IP Address	Owner	Type	Comment
172.16.4.220	172.16.4.224	MAAS	Dynamic	Dynamic
172.16.6.255	172.16.7.254	MAAS	Dynamic	Dynamic

Used

IP Address	Type	Node	Interface	Usage	Owner	Last seen
172.16.4.50	Static	**ubu**	eno1	Rack and region controller	MAAS	Fri, 04 May. 2018 11:26:27
172.16.4.221	Observed		Unknown	DNS	MAAS	Wed, 02 May. 2018 14:27:01

MAAS name: **ubu MAAS**

MAAS version: **2.3.2 (6485-ge93e044-0ubuntu1~16.04.1)**

View release notes · View documentation

© 2018 Canonical Ltd. Ubuntu and Canonical are registered trademarks of Canonical Ltd.

Figure 24: MAAS gui Public network configuration

Private network setup

First of all, as we did in the public network, we setup the Gateway IP and the DNS IP. This gateway is going to be the Gateway of this box despite eventually it's going to route out this box.

The DNS is also this box because it will be a DNS server for the private network.

- Gateway IP: 10.1.1.1
- DNS: 10.1.1.1

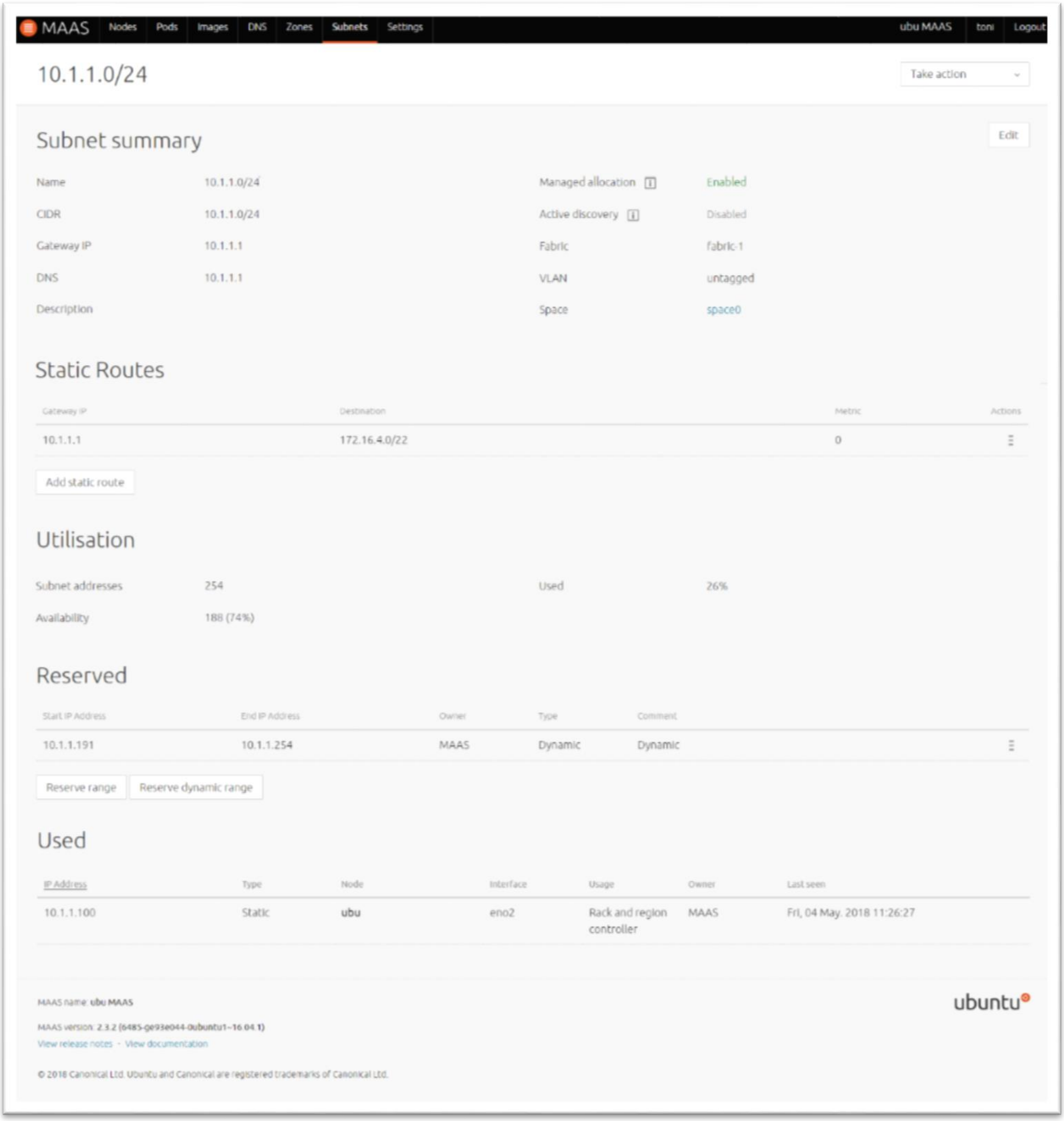
As this box is going to serve DHCP we need to give them a range of addresses he can use. We take the Dynamic range its set but default:

- Start IP Address: 10.1.1.191
- End IP Address: 10.1.1.254

An important configuration here its to fix a **Static Route** to make this network able to have connectivity to the outside world, and it will be done through the public network:

- Gateway IP: 10.1.1.1
- Destination: 172.16.4.0/22

Right below we are showing how the MAAS gui looks like after having realized the previous configurations in the private network.



10.1.1.0/24

Subnet summary

Name	10.1.1.0/24	Managed allocation	Enabled
CIDR	10.1.1.0/24	Active discovery	Disabled
Gateway IP	10.1.1.1	Fabric	fabric-1
DNS	10.1.1.1	VLAN	untagged
Description		Space	space0

Static Routes

Gateway IP	Destination	Metric	Actions
10.1.1.1	172.16.4.0/22	0	⋮

Utilisation

Subnet addresses	254	Used	26%
Availability	188 (74%)		

Reserved

Start IP Address	End IP Address	Owner	Type	Comment	
10.1.1.191	10.1.1.254	MAAS	Dynamic	Dynamic	⋮

Used

IP Address	Type	Node	Interface	Usage	Owner	Last seen
10.1.1.100	Static	ubu	eno2	Rack and region controller	MAAS	Fri, 04 May. 2018 11:26:27

MAAS name: ubu MAAS

MAAS version: 2.3.2 (6485-ge93e044-0ubuntu1~16.04.1)

View release notes · View documentation

© 2018 Canonical Ltd. Ubuntu and Canonical are registered trademarks of Canonical Ltd.

Figure 25: MAAS gui Private network configuration

To be sure everything is properly configured let’s confirm that we can do DNS lookups to the outside world:

```
toni@ubu:~$ ping google.com
PING google.com (172.217.17.14) 56(84) bytes of data.
64 bytes from mad07s09-in-f14.1e100.net (172.217.17.14): icmp_seq=1
ttl=51 time=14.5 ms
64 bytes from mad07s09-in-f14.1e100.net (172.217.17.14): icmp_seq=2
ttl=51 time=13.7 ms
64 bytes from mad07s09-in-f14.1e100.net (172.217.17.14): icmp_seq=3
ttl=51 time=9.85 ms
64 bytes from mad07s09-in-f14.1e100.net (172.217.17.14): icmp_seq=4
ttl=51 time=9.85 ms
64 bytes from mad07s09-in-f14.1e100.net (172.217.17.14): icmp_seq=5
ttl=51 time=14.8 ms
^C
--- google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 9.854/12.591/14.858/2.261 ms
```

```
toni@ubu:~$ nslookup google.com
Server:         172.16.4.2
Address:        172.16.4.2#53

Non-authoritative answer:
Name:   google.com
Address: 172.217.17.14
```

Also it is good to confirm it can also communicate with the other network:

```
toni@ubu:~$ ping 10.1.1.100
PING 10.1.1.100 (10.1.1.100) 56(84) bytes of data.
64 bytes from 10.1.1.100: icmp_seq=1 ttl=64 time=0.030 ms
64 bytes from 10.1.1.100: icmp_seq=2 ttl=64 time=0.024 ms
64 bytes from 10.1.1.100: icmp_seq=3 ttl=64 time=0.026 ms
64 bytes from 10.1.1.100: icmp_seq=4 ttl=64 time=0.040 ms
^C
--- 10.1.1.100 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.024/0.030/0.040/0.006 ms
```

The next step is to generate an SSH key on the MAAS server.

We need a private key set on this server and a public one that will set on any server we want to allow this server to be able to SSH to without a password. Thus, when we spin up nodes on MAAS the software they will automatically hand him without a login handshake.

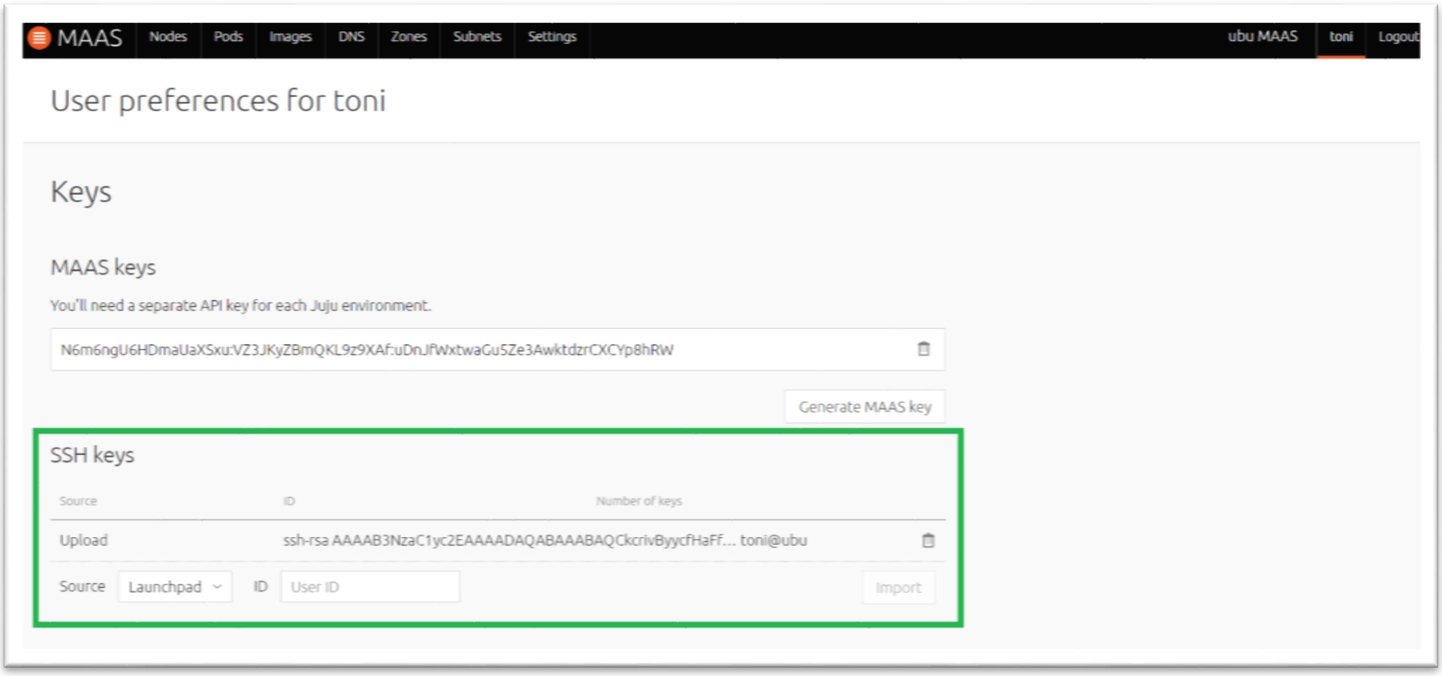
Having said that now we setup an SSH key for MAAS to login to the node machines with the following command line:

```
toni@ubu:~$ ssh-keygen -t rsa
```

Now we just copy the key to the gui and display the key with the cat command:

```
toni@ubu:~$ cat /home/toni/.ssh/id_rsa.pub
```

When its generated we copy and paste the key into the gui:



User preferences for toni

Keys

MAAS keys

You'll need a separate API key for each Juju environment.

N6m6ngU6HDmaUaXSxu:VZ3JKyZBmQKL9z9XAf:uDnJfWxtwaGu5Ze3AwktdzrCXCYp8hRW

SSH keys

Source	ID	Number of keys
Upload	ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCkcrivByycfHaFf... toni@ubu	

Source Launchpad ID User ID

Figure 26: MAAS & SSH keys

At this point we have to install a tool called Etherwake. Etherwake is a shell script wrapper around netcat. It sends a Magic Pke-On-LAN packet, optionally containing a password for those few cards that require it, and will cause the receiving machine to wake up.

MAAS spins up nodes and shuts them down until he needs to use them. To wake them up we have to use wake-on-lan, so the hardware needs to be compatible with wake-o-lan and we must have etherwake installed to be able to send them that Magic packet to wake up whenever we need them to.

```
toni@ubu:~$ sudo apt install etherwake
```

Since our nodes exist on the private network we always need the magic packet to be send to *eno2*, because the other network isn't routable and it will not route that packet back to the node that we need to go to.

Now its needed to edit the etherwake template to remove wakeonlan and add sudo access and use the *eno2* interface:

```
toni@ubu:~$ sudo nano /etc/maas/templates/power/ether_wake.template
```

Now the template file should look like this:

```
# -*- mode: shell-script -*-
#
# Control node power through WOL, via `wakeonlan` or `etherwake`.
#

mac_address={{mac_address}}
power_change={{power_change}}

if [ "${power_change}" != 'on' ]
then
    echo "There is no way to power down a node through etherwake."
>&2
    exit 1
#elif [ -x /usr/bin/wakeonlan ]
#then
#   sudo /usr/bin/wakeonlan $mac_address
elif [ -x /usr/sbin/etherwake ]
then
    sudo /usr/sbin/etherwake -i eno2 $mac_address
else
    echo "No wakeonlan or etherwake program found." >&2
fi

exit 0
```

It is also needed to add etherwake to our sudoers list so the system can run it as sudo. It will be done editing the file with this command:

```
toni@ubu:~$ sudo nano /etc/sudoers.d/99-maas-sudoers
```

Adding this line to the bottom of the text:

```
aas ALL= NOPASSWD: /bin/systemctl status maas-dhcpd
maas ALL= NOPASSWD: /bin/systemctl start maas-dhcpd
maas ALL= NOPASSWD: /bin/systemctl restart maas-dhcpd
maas ALL= NOPASSWD: /bin/systemctl stop maas-dhcpd
maas ALL= NOPASSWD: /bin/systemctl status maas-dhcpd6
maas ALL= NOPASSWD: /bin/systemctl start maas-dhcpd6
maas ALL= NOPASSWD: /bin/systemctl restart maas-dhcpd6
maas ALL= NOPASSWD: /bin/systemctl stop maas-dhcpd6
maas ALL= NOPASSWD: /bin/systemctl status tgt
maas ALL= NOPASSWD: /bin/systemctl start tgt
maas ALL= NOPASSWD: /bin/systemctl disable maas-rackd
maas ALL= NOPASSWD: /bin/systemctl stop maas-rackd
maas ALL= NOPASSWD: SETENV: /usr/sbin/tgt-admin,
/usr/bin/uec2roottar
maas ALL= NOPASSWD: /usr/sbin/etherwake
```

Now is a good time to reboot the MAAS server to allow all these changes to take affect.

```
toni@ubu:~$ sudo reboot now
```

3.2.5. MAAS network hardware

Adding a node to MAAS is typically done via a combination of DHCP (and TFTP), which should, by now be enabled in our MAAS environment and PXE, in which you tell the system in question to use when it boots. This unattended manner of adding a node is called *enlistment*.

Regardless of how a node is added, there are no special requirements for the underlying machine. In particular, there is no need to install an operating system on it.

Once MAAS is working to the point of adding nodes it is important to understand node statuses and node actions.

At this point we are going to spin up nodes to enlist them to MAAS. To enlist, the underlying machine needs to be configured to netboot. Such a machine will undergo the following process:

1) DHCP server is contacted
2) kernel and initrd are received over TFTP
3) machine boots
4) initrd mounts a Squashfs image ephemerally over HTTP
5) cloud-init runs enlistment scripts
6) machine shuts down

The enlistment scripts will send the region API server information about the machine, including the architecture, MAC address and other details which will be stored in the database. This information-gathering process is known as automatic discovery.

Since any system booting off the network can enlist, the enlistment and commission steps are separate. This allows an administrator to "accept" an enlisted machine into MAAS.

In this section we are going to show how the BIOS is configured to allow wake-on-lan/etherwake and how to enable it in PXE boot. Also, it is important to notice that a virtual technology capable hardware is needed and also to have it enabled in the BIOS.

First, its important to make sure boot from the NIC card is enabled, it was necessary to disable all other boot devices to get this to work every time.

Secondly, we had to make sure wake-on-lan is enabled in the BIOS to allow the machine to be woke up from the network.

Then, when setting up the power options for the nodes from the GUI we had to make sure to enter the MAC address of the card the *eno2* MAAS NIC talked to during the PXE boot.

- @**MAC**: d0:17:c2:d1:cd:12

This step is mandatory, otherwise he will not be able to wake him up.

Last but not least, its nedded to name the nodes something you can easily tell which box it is, making sure the internal domain its used on the node name. The domain will be whatever the MAAS node name is by default and it can be changed it in the GUI.

It is necessary to boot each one of the nodes and they will appear in the Node tab on the MAAS gui. As they boot we are going to edit them and enter our wake-on-lan.

Another thing to point out is that these nodes are going to be running virtual machines, so they have to have Intel virtualized technology to be able to do 64-bit virtual machines.

In my case I just did this progress once, so I had to take a pc which has this hardware specifications:

node1

Samsung syncmaster 701n

Intel® Core™ i7-6700K CPU @4.00Ghz

4GB DDR4 RAM

1 250GB SATA hard drive

1 80GB SATA hard drive

1 internal on board NIC

1 PCI NIC card

I'm going to show a MAAS nodes hardware specification case were 6 PC's its nodes on the power type screen has to be the same.

node1780

Dell optiplex 780 Q45 Chipset motherboard VTx capable

Intel duo2core 2.6Ghz Q45 processor VTx capable

8GB DDR3 RAM

1 250GB SATA hard drive

1 80GB SATA hard drive

1 internal on board NIC

1 PCI NIC card

node2780

Dell optiplex 780 Q845 Chipset motherboard VTx capable

Intel duo2core 2.6Ghz Q45 processor VTx capable

7GB DDR3 RAM

1 250GB SATA hard drive

1 80GB SATA hard drive

1 internal on board NIC

node3780

Dell optiplex 780 Q45 Chipset motherboard VTx capable

Intel duo2core 2.6Ghz Q45 processor VTx capable

8GB DDR3 RAM

1 250GB SATA hard drive

1 80GB SATA hard drive

1 internal on board NIC

node4780

Dell optiplex 780 Q45 Chipset motherboard VTx capable

Intel duo2core 3.0Ghz Q45 processor VTx capable

8GB DDR3 RAM

1 120GB SSD SATA hard drive

1 160GB SATA hard drive

1 internal on board NIC

1 PCI NIC card

node5780

Dell optiplex 780 Q45 Chipset motherboard VTx capable

Intel duo2core 2.93Ghz Q45 processor VTx capable

8GB DDR3 RAM

1 250GB SATA hard drive

1 80GB SATA hard drive

1 internal on board NIC

node1dc7900

HPDC7900 Q45 Chipset motherboard VTx capable

Intel duo2core 3.0Ghz Q45 processor VTx capable

6GB DDR2 RAM

2 80GB SATA hard drive

1 internal on board NIC

node1dc7900

HPDC7900 Q45 Chipset motherboard VTx capable

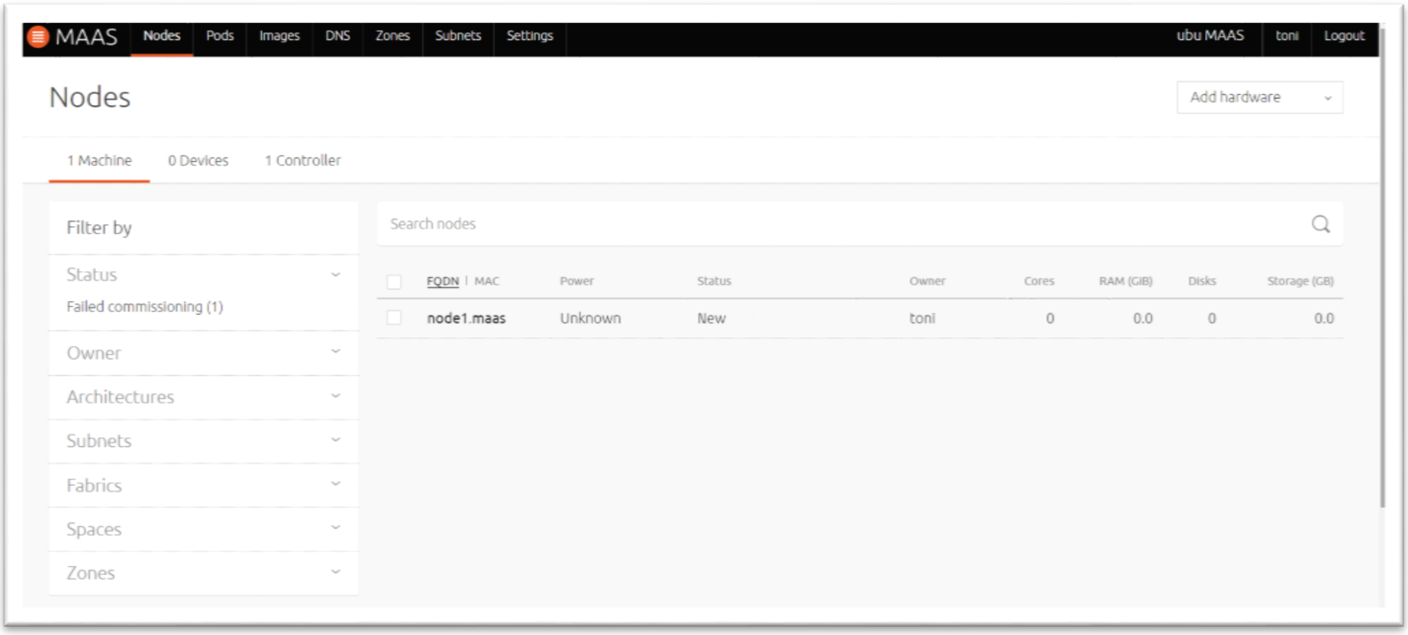
Intel duo2core 3.0Ghz Q45 processor VTx capable

6GB DDR2 RAM

2 80GB SATA hard drive

1 internal on board NIC

Once we did the proper PXE boot configuration we exit the boot menu, the machine will spin up and its going to boot from our MAAS server via PXE. It will grab a boot image and it's going to register with the MAAS server.



FQDN | MAC	Power	Status	Owner	Cores	RAM (GiB)	Disks	Storage (GB)
node1.maas	Unknown	New	toni	0	0.0	0	0.0

Figure 27: Node 1 MAAS gui tab

The menu inside the node includes links to the following:

- Machine summary: Overview of CPU, memory, storage, tag and general settings.
- Interfaces: Network and interface configuration for a node.
- Storage: File system, partitioning and storage overview.
- Commissioning: Timestamped completion and status log from the commissioning process.
- Logs: Raw log output, switchable between YAML and XML output.
- Events: Timestamped status updates for events and actions performed on the node.
- Configuration: Machine and power configuration options.

3.2.6. MAAS node configuration

In this section we are going to commission all the nodes to MAAS, deploying one node to a standard Ubuntu image.

Firstly, we have to add the MAAS server to the DNS-nameservers list in the */etc/network/interfaces* file to avoid future network properties issues.

```
# This file describes the network interfaces available on your
# system and how to activate them.
# For more information, see interfaces.

# The loopback network interface
      auto lo
            iface lo inet loopback

# The primary network interface
      auto eno1
            iface eno1 inet dhcp
            dns-nameserver 10.1.1.100

      auto eno2
            iface eno2 inet static
            address 10.1.1.100
            netmask 255.255.255.0
```

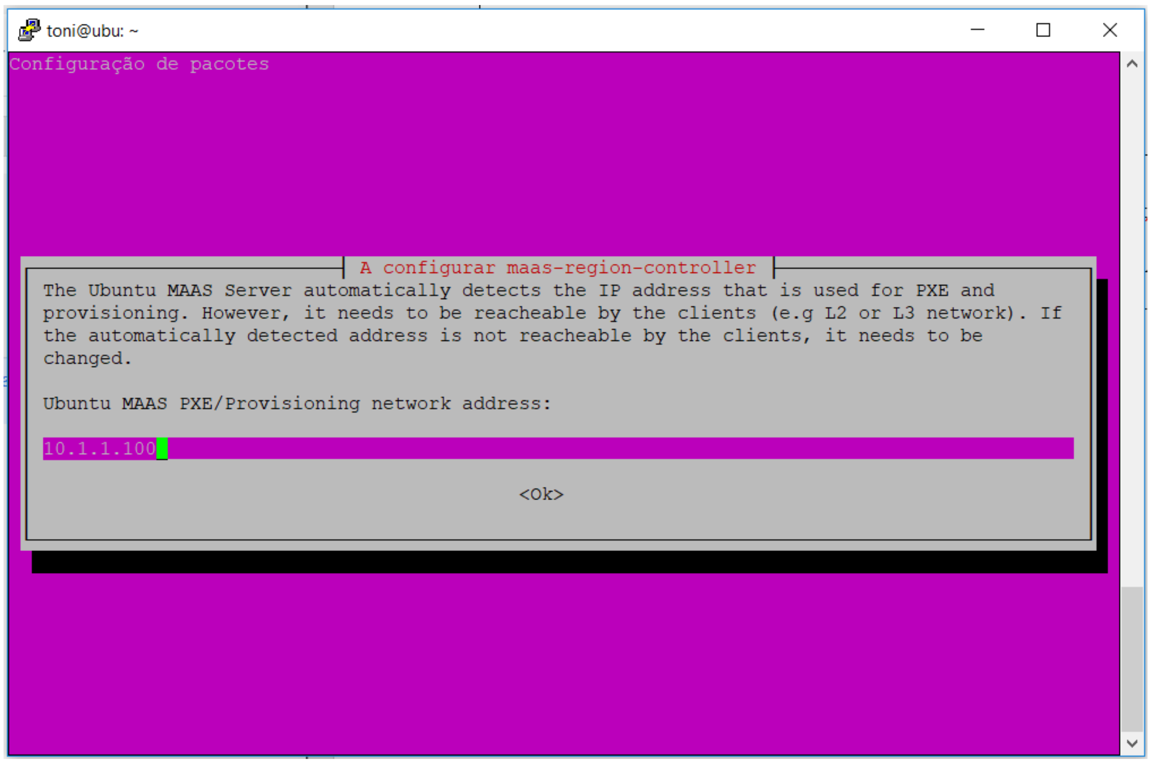
With the previous unfixed configuration our name server was 172.16.4.2 as well as the Gateway. When trying to find another node with *nslookup* we'd get an arbitrary IP address, it tried to go find it on the internet and just stucked, doesn't knowing anything about it. That's because we are doing NAT routing out to a DNS server somewhere on the internet with a completely unknowledge about our node on the private network.

Thus, for the MAAS node to be able to look up another node doing a *nslookup* by its name it is necessary to have the MAAS server in our name server list

Next step will be running this command to enter our region controller IP, I changed mine to the private network interface:

```
toni@ubu:~$ sudo dpkg-reconfigure maas-region-controller
```

Then we change that @IP to 10.1.1.100



```
toni@ubu: ~
Configuração de pacotes

A configurar maas-region-controller
The Ubuntu MAAS Server automatically detects the IP address that is used for PXE and
provisioning. However, it needs to be reacheable by the clients (e.g L2 or L3 network). If
the automatically detected address is not reacheable by the clients, it needs to be
changed.

Ubuntu MAAS PXE/Provisioning network address:

10.1.1.100

                                   <Ok>
```

Figure 28: Maas region controller configuration

3.2.7. MAAS commission nodes

Once a node is added to MAAS, the next logical step is to commission it.

To commission, the underlying machine needs to be configured to netboot (this should already have been done during the enlistment stage).

The commissioning scripts will talk to the region API server to ensure that everything is in order and that eventual deployment will succeed.

The image used is, by default, the latest Ubuntu LTS release and should not require changing. However, it can be configured in the web UI in the 'Settings' page.

To commission, on the 'Nodes' page, select a node and choose 'Commission' under the 'Take action' dropdown menu.

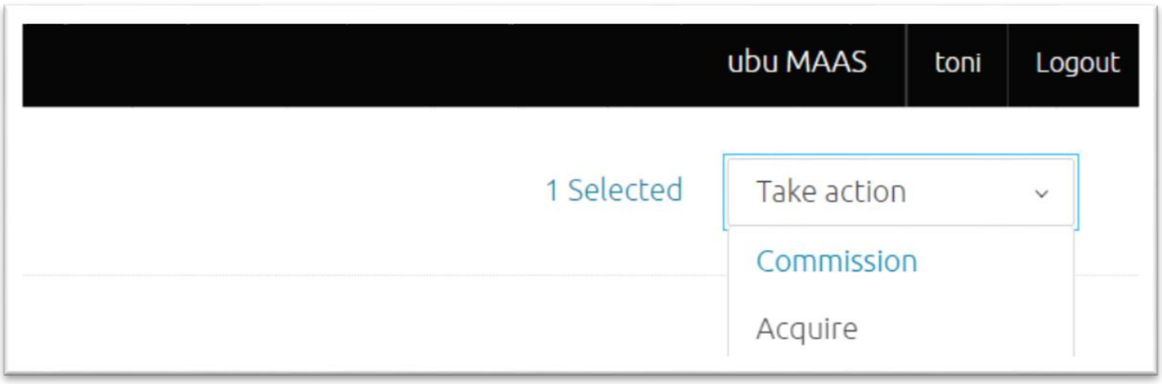


Figure 29: Node commissioning

We have the option of selecting some extra parameters (checkboxes) and performing hardware tests.

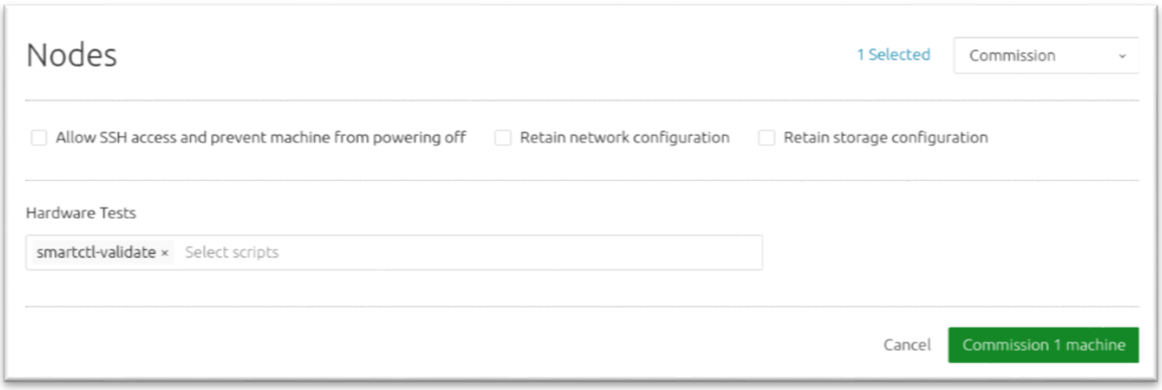

Nodes
1 Selected
Commission
Allow SSH access and prevent machine from powering off
Retain network configuration
Retain storage configuration
Hardware Tests
smartctl-validate ×
Select scripts
Cancel
Commission 1 machine

Figure 30: Scripts adding

Then we finalize the directive by hitting 'Commission machine'.

While a node is commissioning its status will change to *Commissioning*. During this time the node's network topology will be discovered. This will prompt one of the node's network interfaces to be connected to the fabric, VLAN, and subnet combination that will allow it to be configured. By default, a static IP address will be assigned out of the reserved IP range for the subnet. That is, an IP assignment mode of 'Auto assign' will be used. See the next section for details on assignment modes.

Once a node is commissioned its status will change to *Ready* and an extra tab for the node called 'Commissioning' will become available. This tab contains the results of the scripts executed during the commissioning process.

Now it's time to verify the networking by going to the details page for the node(s) that have multiple NICs and check that the second NIC (the non-PXE one) and check if it:

- Is connected to the subnet
- Has the "IP address" field set to "unconfigured"
- The first NIC should be the same except the IP address field will be set to "Auto assign"

The next step will be to *deploy* the node.

3.2.8. Deploying nodes

Deploying a node means to install an operating system on it, in our case Ubuntu Server 16.04 LTS. In this section we are going to explain how this can be done using MAAS and Openstack respectively.

3.2.8.1. Deploying nodes from MAAS

The agent that triggers deployment may vary. For instance, if the nodes are destined to be running complex, inter-related services that may involve scaling up or down, that is, if they will be regarded as a "cloud" resource, then Juju is the recommended deploy agent (it will also install & configure services on the deployed nodes). But in our case, as we want to install a base operating system and work on the machines manually then we will deploy a node directly with MAAS.

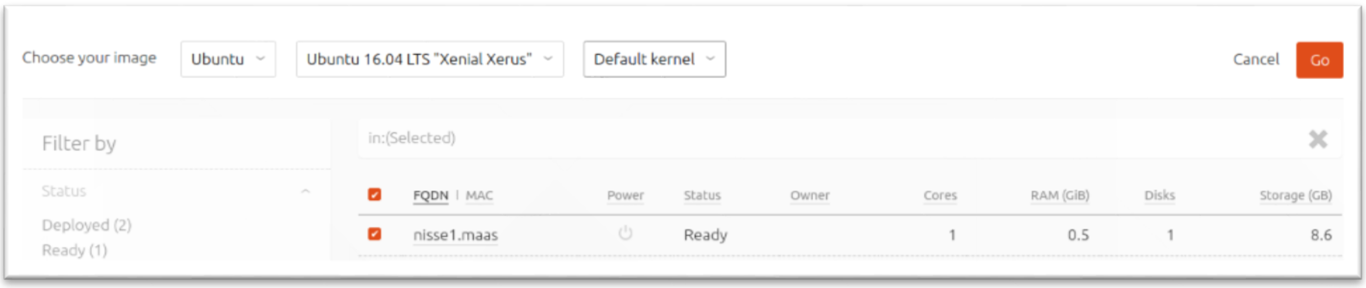
The node, only if deployed with MAAS, will also be ready to accept connections via SSH to the 'ubuntu' user account providing an SSH key has been imported to the user's MAAS account.

Before deploying it is advisable to:

- Review and possibly set the Ubuntu kernels and the Kernel boot options that will get used by deployed nodes.
- Ensure any pertinent SSH keys are imported (see SSH keys) to MAAS so connections can be made to deployed nodes.

To deploy directly from MAAS we have to select the chosen node and press the 'Deploy' option from the 'Take action' selector we have previously seen.

Then have the option of deviating from the default OS appears, release, and kernel. When its ready we just have to press the 'Go' button.



Choose your image: Ubuntu · Ubuntu 16.04 LTS "Xenial Xerus" · Default kernel

Status
Deployed (2)
Ready (1)

FQDN | MAC	Power	Status	Owner	Cores	RAM (GiB)	Disks	Storage (GB)
nisse1.maas		Ready		1	0.5	1	8.6

Figure 31: MAAS gui before deploying

While a node is deploying its status will change to Deploying 'OS', where 'OS' is the name of the operative system being deployed, in our case 'Deploying Ubuntu 16.04 LTS'. Once a node has finished deploying its status changed to ‘Deployed Ubuntu 16.04 LTS'.

To access to the deployed node we have to do an SSH to it. Since we have the key set up we don’t need to have a login and typing its @IP is enough:

```
toni@ubu:~$ ssh ubuntu@10.1.1.151
```

To finish this section, let’s realice some testings:

```
ubuntu@node1:~$ nslookup node1.maas
Server:     10.1.1.100
Address:    10.1.1.100#53

Name: node1.maas
Address: 10.1.1.151
```

```
ubuntu@node1:~$ ping google.com
PING google.com (216.58.218.174) 56(84) bytes of data.
64 bytes from dfw06s46-in-f14.1e100.net (216.58.218.174):
icmp_seq=1 ttl=52 time=27.2 ms
64 bytes from dfw06s46-in-f14.1e100.net (216.58.218.174):
icmp_seq=2 ttl=52 time=37.9 ms
64 bytes from dfw06s46-in-f14.1e100.net (216.58.218.174):
icmp_seq=3 ttl=52 time=27.3 ms

^C
--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 3026ms
```

```
ubuntu@node1:~$ sudo apt install nload
```

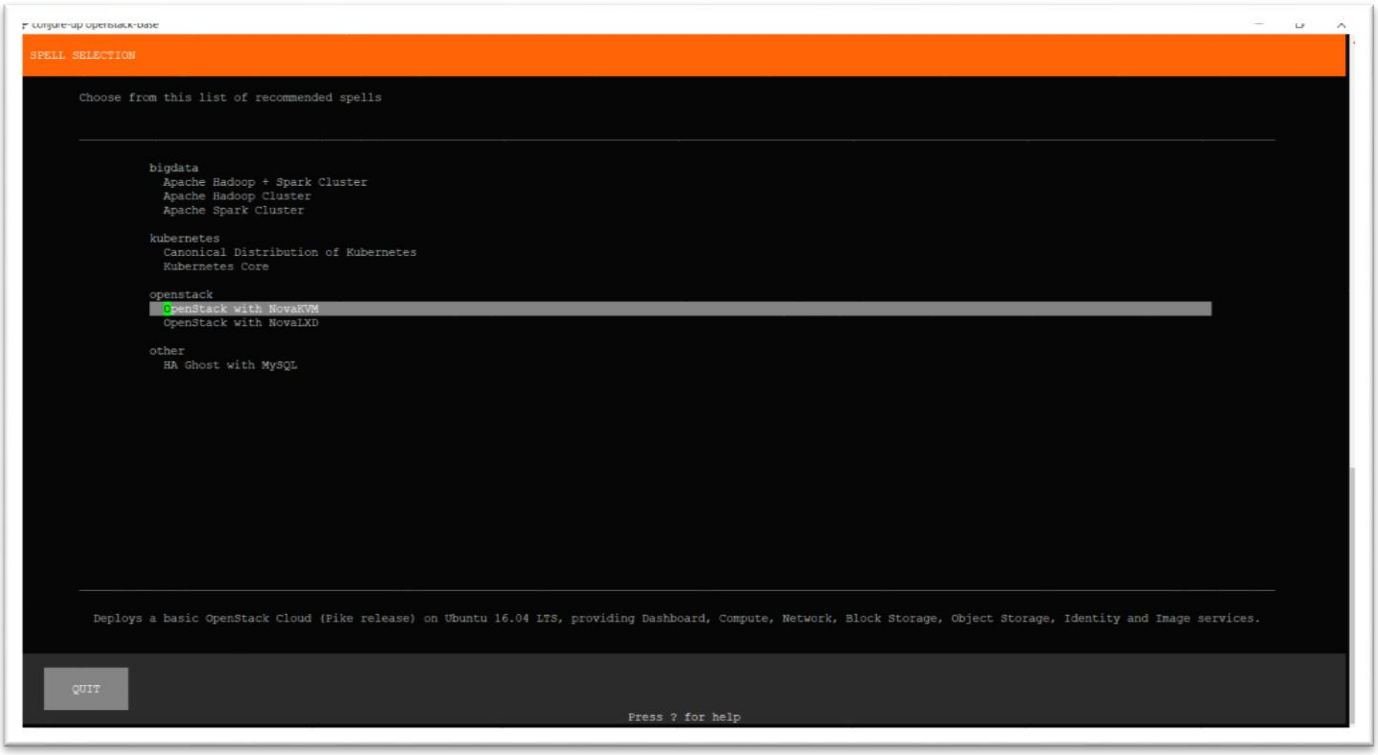
The previous command lines reflects that the node can ping Google going out to the internet and also can install packages, so we can asume its working fine.

3.2.8.2. Deploying nodes from Openstack

First of all we are going to install the conjure-up snap on the MAAS server, in a terminal window:

```
toni@ubu:~$ sudo snap install conjure-up -classic
```

Typing the command *conjure-up* the following screen appears:



```
conjure-up openstack-base

SPELL SELECTION

      Choose from this list of recommended spells

              bigdata
                Apache Hadoop + Spark Cluster
                Apache Hadoop Cluster
                Apache Spark Cluster

              kubernetes
                Canonical Distribution of Kubernetes
                Kubernetes Core

              openstack
                OpenStack with NovaKVM
                OpenStack with NovaLXD

              other
                HA Ghost with MySQL

        Deploys a basic OpenStack Cloud (Pike release) on Ubuntu 16.04 LTS, providing Dashboard, Compute, Network, Block Storage, Object Storage, Identity and Image services.

  QUIT
                                        Press ? for help
```

Figure 32: Conjure-up main screen

Next step is to select one of the two install types OpenStack offers:

1) OpenStack with Nova-lxd

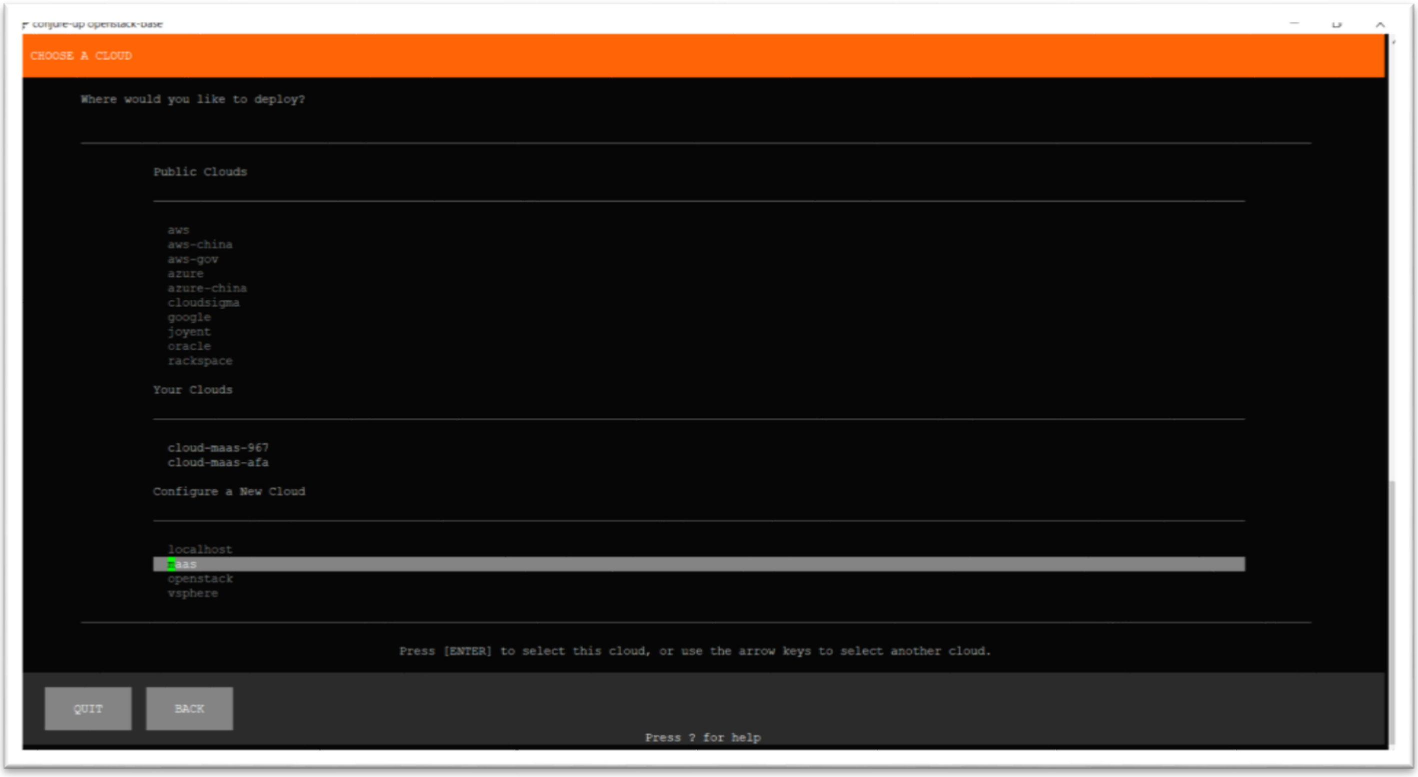
This installs OpenStack with nova configured to use the lxd hypervisor ready to launch machine containers.

2) OpenStack with KVM

This installs OpenStack with nova configured to use the KVM (Kernel-based Virtual Machine) hypervisor ready to launch full virtual machine. It provides Dashboard, Compute, Network, Block Storage, Object Storage, Identity and Image services.

As we are building a production cloud we are going to use the last option.

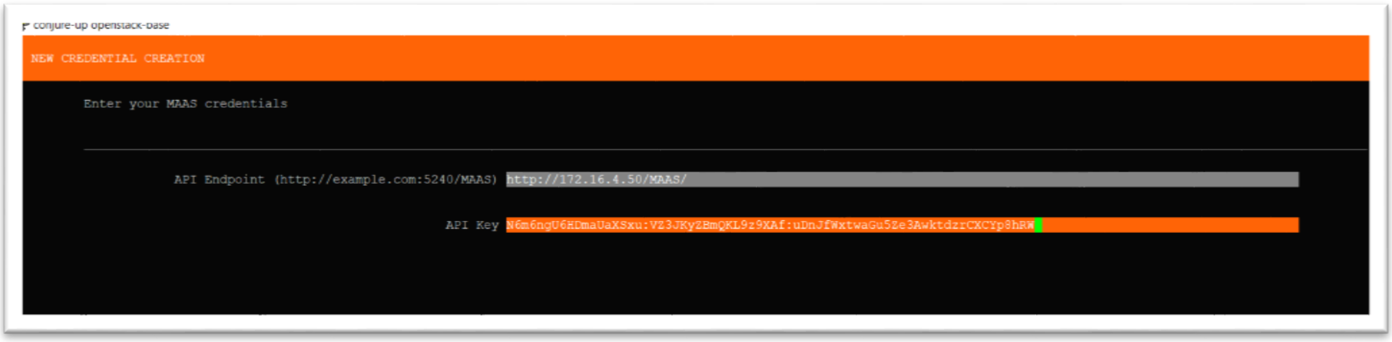
Once we have selected OpenStack with NovaKVM, we are prompted to create a new cloud with a single option provided: MAAS.



```
CHOOSE A CLOUD

    Where would you like to deploy?

          Public Clouds

            aws
            aws-china
            aws-gov
            azure
            azure-china
            cloudsigma
            google
            joyent
            oracle
            rackspace
          Your Clouds

            cloud-maas-967
            cloud-maas-afa
          Configure a New Cloud

            localhost
            maas
            openstack
            vsphere

              Press [ENTER] to select this cloud, or use the arrow keys to select another cloud.

                              Press ? for help
```

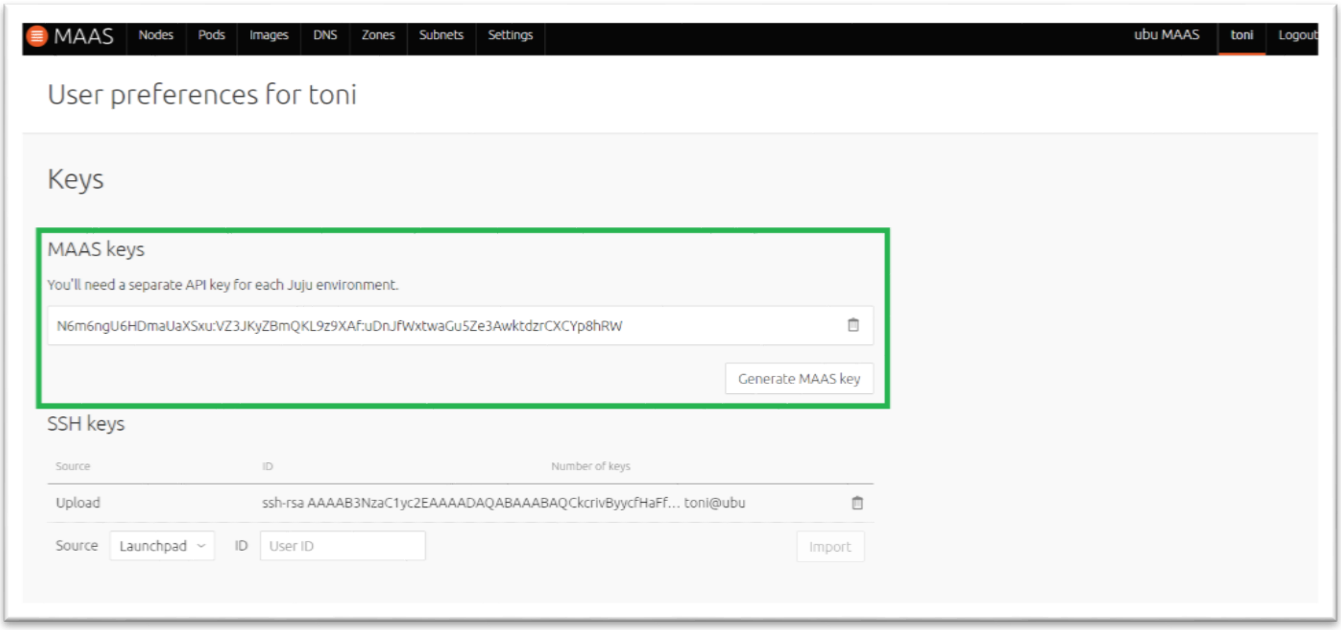
Figure 33: Cloud selector

To create a MAAS based cloud we need to enter the MAAS API endpoint (http://172.16.4.50/MAAS/) and the API key:



```
conjure-up openstack-base
NEW CREDENTIAL CREATION

    Enter your MAAS credentials

            API Endpoint (http://example.com:5240/MAAS) http://172.16.4.50/MAAS/

                                            API Key N6m6ngU6HDmaUaXSxu:VZ3JKyZBmQKL9z9XAf:uDnJfWxtwaGu5Ze3AwktdzrCXCYp8hRW
```

Figure 34: Credential creation



User preferences for toni

Keys

MAAS keys

You'll need a separate API key for each Juju environment.

N6m6ngU6HDmaUaXSxu:VZ3JKyZBmQKL9z9XAf:uDnJfWxtwaGu5Ze3AwktdzrCXCYp8hRW

Generate MAAS key

SSH keys

Source	ID	Number of keys
Upload	ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCkcrivByycfHaFf... toni@ubu	

Source: Launchpad ID: User ID Import

Figure 35: MAAS API key location

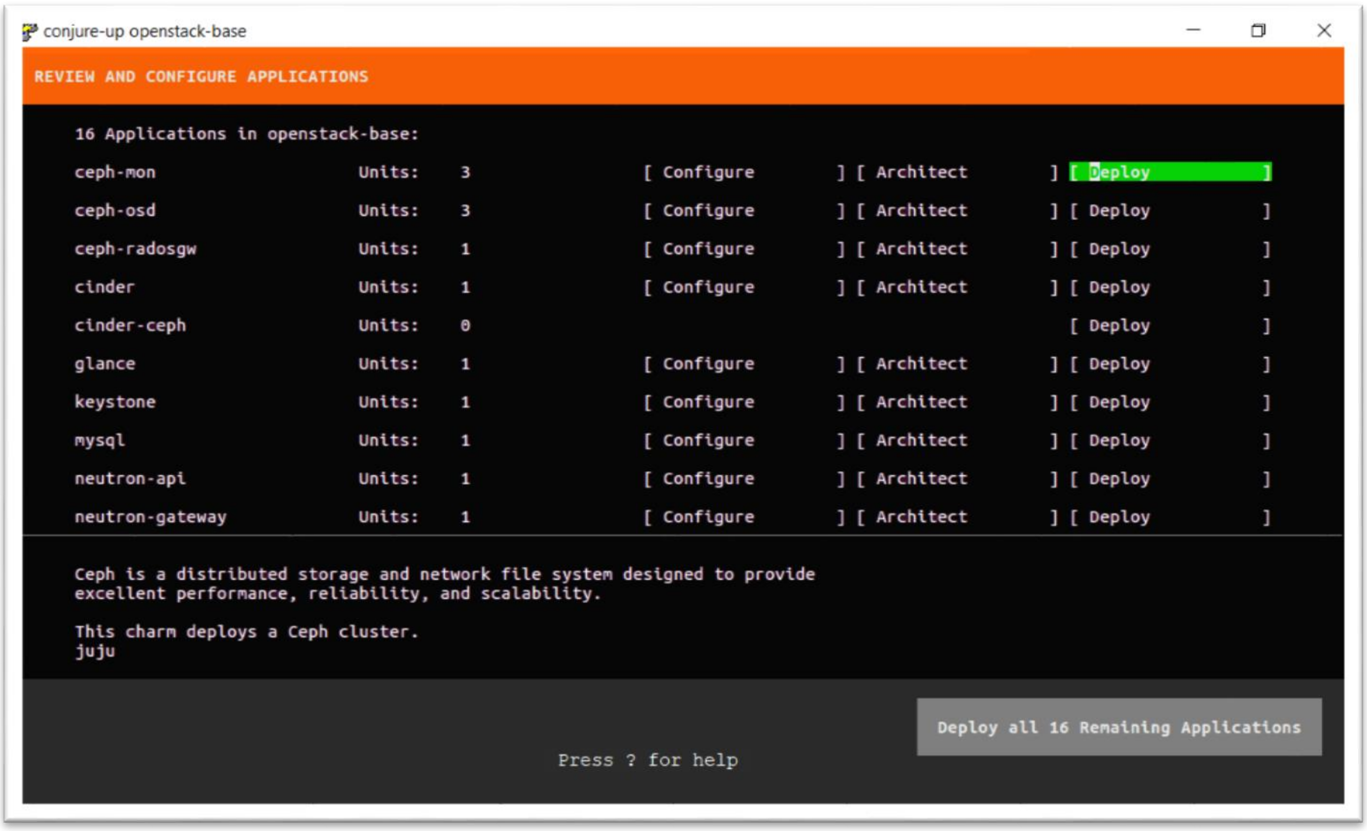
Then we have to import SSH keypairs into OpenStack. This will allows us to access the newly deployed instances via SSH with our current user.

- **SSH public key path**: ~/.ssh/id_rsa.pub

Once entered, the user have the opportunity to configure the individual services.

Those are the OpenStack services offered:

- **Configure**: this will give you the opportunity to change the config of the various OpenStack services.
- **Architect**: Choose where to place a service. You do not need to place services as conjure-up will do so automatically.
- **Deploy**: Deploy the service as configured according to the architecture.



```
conjure-up openstack-base

REVIEW AND CONFIGURE APPLICATIONS

  16 Applications in openstack-base:
  ceph-mon              Units:   3         [ Configure      ] [ Architect      ] [ Deploy          ]
  ceph-osd              Units:   3         [ Configure      ] [ Architect      ] [ Deploy          ]
  ceph-radosgw          Units:   1         [ Configure      ] [ Architect      ] [ Deploy          ]
  cinder                Units:   1         [ Configure      ] [ Architect      ] [ Deploy          ]
  cinder-ceph           Units:   0                                               [ Deploy          ]
  glance                Units:   1         [ Configure      ] [ Architect      ] [ Deploy          ]
  keystone              Units:   1         [ Configure      ] [ Architect      ] [ Deploy          ]
  mysql                 Units:   1         [ Configure      ] [ Architect      ] [ Deploy          ]
  neutron-api           Units:   1         [ Configure      ] [ Architect      ] [ Deploy          ]
  neutron-gateway       Units:   1         [ Configure      ] [ Architect      ] [ Deploy          ]

  Ceph is a distributed storage and network file system designed to provide
  excellent performance, reliability, and scalability.

  This charm deploys a Ceph cluster.
  juju

                                                  Deploy all 16 Remaining Applications
                          Press ? for help
```

Figure 36: Openstack conjure-up applications

Those are the 16 applications available in our OpenStack-base:

- ***Ceph-mon***: Ceph is a distributed storage and network file system designed to provide excellent performance, reliability and scalability. This charm deploys a Ceph monitor cluster. One or more instances of ceph-mon form a Paxos part-time parliament cluster that provides extremely reliable and durable storage of cluster membership, configuration, and state.

- ***Ceph-osd***: This charm deploys additional Ceph OSD (Object Storage Daemon) service units and should be used in conjunction with the “ceph” charm to scale out the amount of storage avaliable in a Ceph cluster. It is responsible for storing objects on a local file system and providing access to them over the network.

- ***Ceph-radosgw***: This charm deploys the RADOS Gateway, a s3 and Swift compatible HTTP gateway for online object storage on-top of a ceph cluster.

- ***Cinder***: This charm provides the Cinder volume service for OpenStack. It is intended to be used alongside the other OpenStack components, starting with the Folsom release. Cinder is made up of 3 separate services: an API service, a scheduler and a volume service. This charm allows them to be deployed in different combination, depending on user preference and requirements.

- ***Cinder-ceph***: This charm provides a Ceph storage backend for use with the Cinder charm; this allows multiple Ceph storage clusters to be associated with a single Cinder deployment, potentially alongside other storage backends from other vendors.

- ***Glance***: This charm provides the Glance image service for OpenStack. It is intended to be used alongside the other OpenStack components, starting with the Essex release in Ubuntu 12.04.

- ***Keystone***: This charm provides Keystone, the Openstack identity service. It's target platform is (ideally) Ubuntu LTS + Openstack.

- ***Mysql***: Percona XtraDB Cluster is a high availability and high scalability solution for MySQL clustering. Percona XtraDB Cluster integrates Percona Server with the Galera library of MySQL high availability solutions in a single product package which enables you to create a cost-effective MySQL cluster. This charm deploys Percona XtraDB Cluster onto Ubuntu.

- ***Neutron-api***: This principle charm provides the OpenStack Neutron API service which was previously provided by the nova-cloud-controller charm. When this charm is related to the nova-cloud-controller charm the nova-cloud controller charm will shutdown its api service, de-register it from keystone and inform the compute nodes of the new neutron url.

- ***Neutron-gateway***: Neutron provides flexible software defined networking (SDN) for OpenStack. This charm is designed to be used in conjunction with the rest of the OpenStack related charms in the charm store to virtualize the network that Nova Compute instances plug into.

- ***Neutron-openvswitch***: This subordinate charm provides the Neutron OpenvSwitch configuration for a compute node. Once deployed it takes over the management of the Neutron base and plugin configuration on the compute node.

- ***Nova-cloud-controller***: Cloud controller node for OpenStack nova. Contains nova-schedule, nova-api, nova-network and nova-objectstore. If console access is required then console-proxy-ip should be set to a client accessible IP that resolves to the nova-cloud-controller. If running in HA mode then the public vip is used if console-proxy-ip is set to local.
  Note: The console access protocol is baked into a guest when it is created, if you change it then console access for existing guests will stop working.

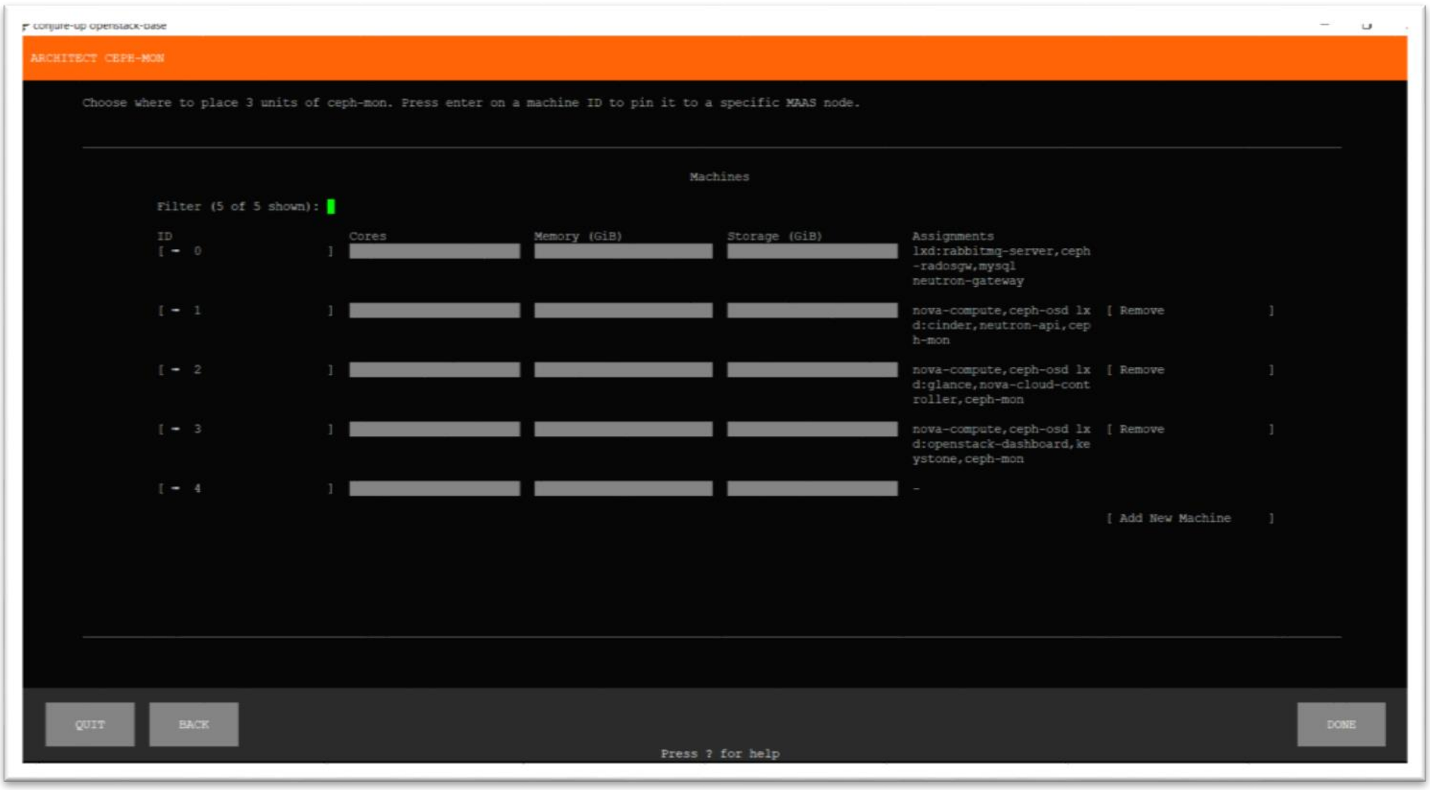
- ***Nova-compute***: This charm provides Nova Compute, the OpenStack compute service. Its target platform is Ubuntu (preferably LTS) + OpenStack.

- ***Ntp***: NTP (Network Time Protocol) provides network based time services to ensure synchronization of time across computers.

- ***Openstack-dashboard***: The OpenStack Dashboard provides a Django based web interface for use by both administrators and users of an OpenStack Cloud. It allows you to manage Nova, Glance, Cinder and Neutron resources within the cloud.

- ***Rabbitmq-server***: RabbitMQ is an implementation of AMQP, the emerging standard for high performance enterprise messaging. The RabbitMQ server is a robust and scalable implementation of an AMQP broker.

We took *Ceph-mon* as an example to show the available configurations we can manage:



```
conjure-up openstack-base

ARCHITECT CEPH-MON

Choose where to place 3 units of ceph-mon. Press enter on a machine ID to pin it to a specific MAAS node.

                                    Machines

   Filter (5 of 5 shown):

   ID                Cores        Memory (GiB)      Storage (GiB)     Assignments
   [ - 0        ]                                                     lxd:rabbitmq-server,ceph
                                                                      -radosgw,mysql
                                                                      neutron-gateway

   [ - 1        ]                                                     nova-compute,ceph-osd lx  [ Remove        ]
                                                                      d:cinder,neutron-api,cep
                                                                      h-mon

   [ - 2        ]                                                     nova-compute,ceph-osd lx  [ Remove        ]
                                                                      d:glance,nova-cloud-cont
                                                                      roller,ceph-mon

   [ - 3        ]                                                     nova-compute,ceph-osd lx  [ Remove        ]
                                                                      d:openstack-dashboard,ke
                                                                      ystone,ceph-mon

   [ - 4        ]                                                     -

                                                                                                [ Add New Machine ]
```

Figure 37: Ceph-mon architect

As we can see, the desired Cores, Memory and Storage can be configure on each unit before pinning it to a specific MAAS node.

Once satisfied with the config and architecture, the remain applications can be deployed.

3.2.9. Ubuntu Linux MAAS decommission nodes & Juju and Openstack install

In the last step we did some testing on one of the nodes that we deployed. Now it's time to return that node to the pool so that he can spin it up while he is doing his OpenStack install.

First we have to shut down the node:

```
ubuntu@node1:~$ sudo shutdown -h now
```

Now it's we are going to install OpenStack:

```
toni@ubu:~$ sudo apt install openstack
```

And now it's time for Juju:

```
toni@ubu:~$ sudo apt install juju
```

Juju requires an initial configuration to be able to work. We need the MAAS API key for Juju to know what MAAS server he's allowed to use.

We are going to make a Juju directory in the home directory without sudoing, because we want these directories and Juju to be as our user ID, in view of that is the SSH he's going to use:

```
toni@ubu:~$ mkdir ~/.juju
            cd ~/.juju
```

Create the *environments.yaml* file:

```
toni@ubu:~/.juju$ nano environments.yaml
```

Our environments file will look like this use your own server IP and API key:

```
environments:
  maas:
    type: maas
    maas-server: 'http://10.1.1.100:80/MAAS'
    maas-oauth:
'N6m6ngU6HDmaUaXSxu:VZ3JKyZBmQKL9z9XAf:uDnJfWxtwaGu5Ze3AwktdzrCXCYp
8hRW'
    admin-secret: 'toni4d12'
    default-series: xenial
```

Now we can close this edited file saving it before and check Juju's status:

```
toni@ubu:~/.juju$ juju status
ERROR Unable to connect to environment "maas".
Please check your credentials or use 'juju bootstrap' to créate a
new environment.

Error details:
Environment is not bootstrapped
```

Having the environment not bootstrapped is a good thing, because OpenStack will handle that

Now that Juju is installed, let's start the OpenStack install:

```
toni@ubu:~/.juju$ sudo openstack-install
```

The OpenStack GUI appears and we have to go throught the next steps:

1) Put an easy remembering password.
2) Select the full landscape OpenStak autopilot
3) Fill the following fields in the Landscape OpenStack Autopilot Setup:
    - Admin Email*: tony.ag@hotmail.com*
    - Admin Name: toni
    - MAAS Server IP: *10.1.1.100*
    - MAAS API KEY: *'N6m6ngU6HDmaUaXSxu:VZ3JKyZBmQKL9z9XAf:uDnJfWxtwaGu5Ze3AwktdzrCXCYp8hRW'*

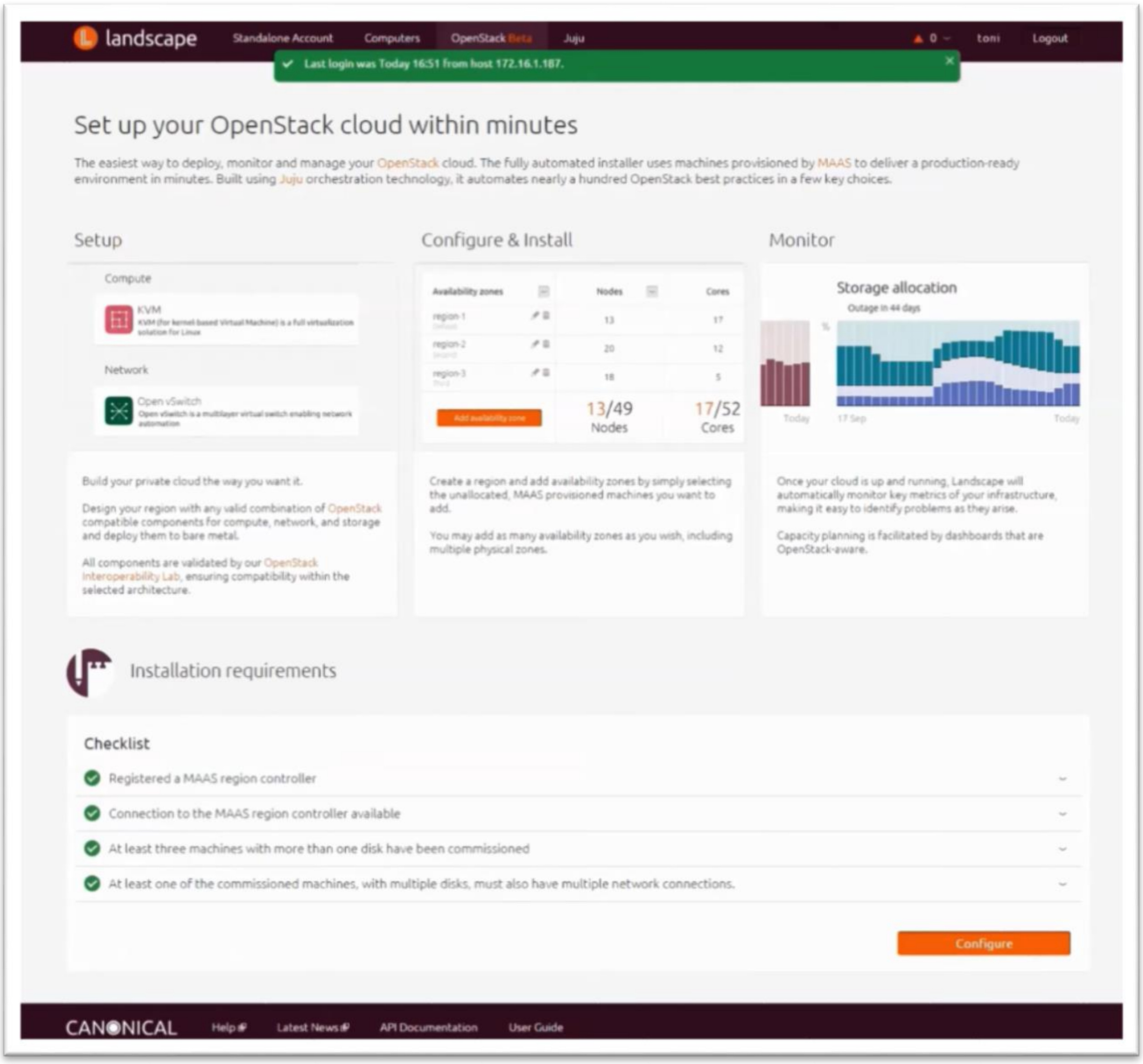
Having done that, it's going to come out and talk to MAAS and it will show many nodes to we have deployed.

To get the OpenStack install going we have to save one node for Juju to spin up. In the MAAS gui we can see how it grabbed a node and its allocating and deploying it, installing what its necessary to get Juju on that node.

While juju and landscape are installing, the bootstrap goes to about 400 seconds and the landscape download runs to about 1600 seconds. When its finished it is going to appear a message as the one below:

```
------------------------------  Info  -----------------------------

To continue with OpenStack instalation visit:

http:// 10.1.1.111/account/standalone/openstack

Landscape Login Credentials:
      Email: tony.ag@hotmail.com
      Passwork: toni4d12
```

OpenStack has now been installed: it went throught, deployed Juju, deployed OpenStack, download landscape and spun him up.



Last login was Today 16:51 from host 172.16.1.187.

Set up your OpenStack cloud within minutes

The easiest way to deploy, monitor and manage your OpenStack cloud. The fully automated installer uses machines provisioned by MAAS to deliver a production-ready environment in minutes. Built using Juju orchestration technology, it automates nearly a hundred OpenStack best practices in a few key choices.

Setup

Compute

KVM
KVM (for kernel-based Virtual Machine) is a full virtualization solution for Linux

Network

Open vSwitch
Open vSwitch is a multilayer virtual switch enabling network automation

Configure & Install

Availability zones	Nodes	Cores
region-1	13	17
region-2	20	12
region-3	18	5
Add availability zone	13/49 Nodes	17/52 Cores

Monitor

Build your private cloud the way you want it.

Design your region with any valid combination of OpenStack compatible components for compute, network, and storage and deploy them to bare metal.

All components are validated by our OpenStack Interoperability Lab, ensuring compatibility within the selected architecture.

Create a region and add availability zones by simply selecting the unallocated, MAAS provisioned machines you want to add.

You may add as many availability zones as you wish, including multiple physical zones.

Once your cloud is up and running, Landscape will automatically monitor key metrics of your infrastructure, making it easy to identify problems as they arise.

Capacity planning is facilitated by dashboards that are OpenStack-aware.

Installation requirements

Checklist

✓ Registered a MAAS region controller

✓ Connection to the MAAS region controller available

✓ At least three machines with more than one disk have been commissioned

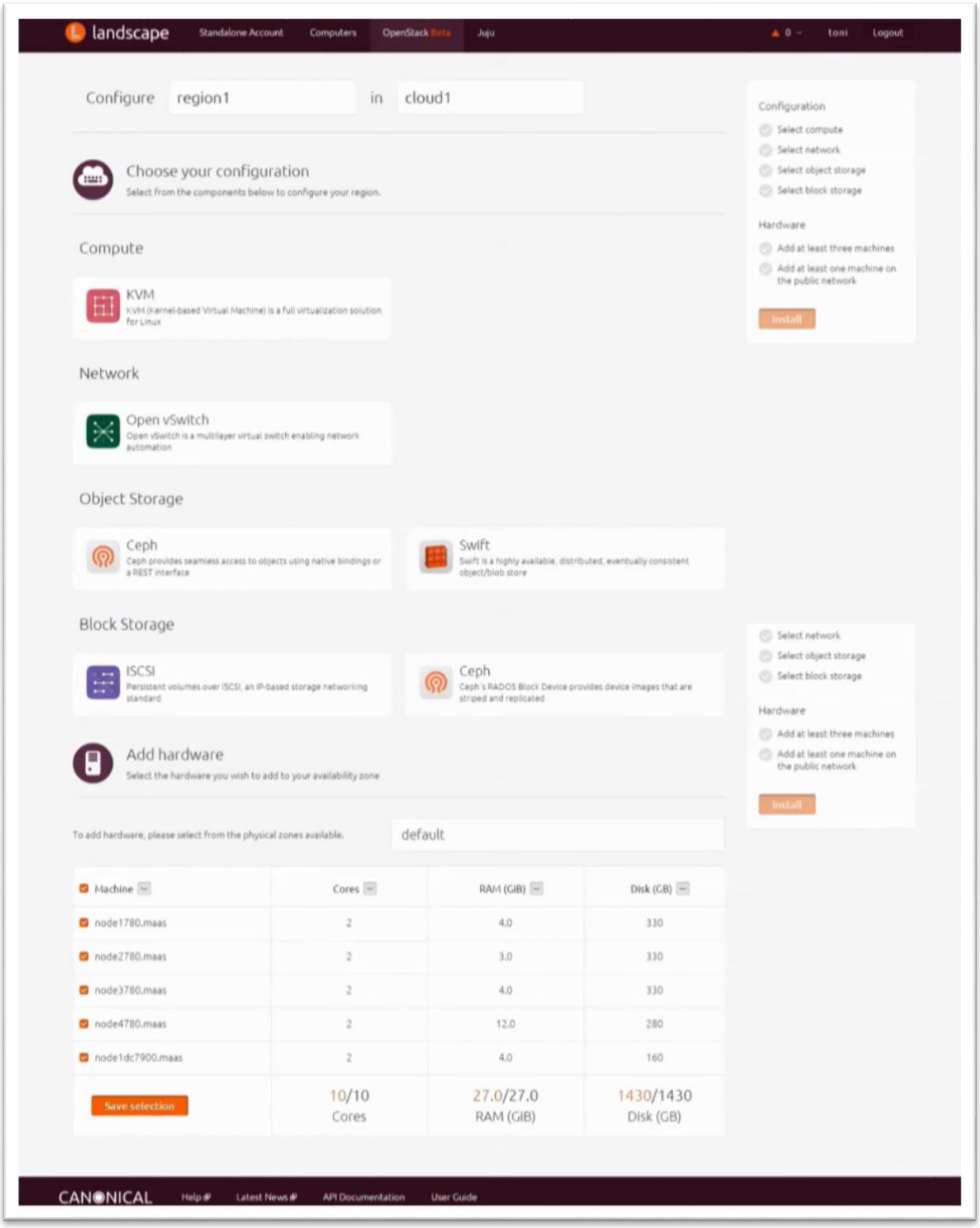
✓ At least one of the commissioned machines, with multiple disks, must also have multiple network connections.

Figure 38: Openstack cloud landscape gui A

The Checklist requirements must be all completed before continuing, those are:

- Registered a MAAS region controller
- Connection to the MAAS region controller available
- At least three machines with more tan one disk have been commissioned
- At least one of three machines with multiple disks, must also have multiple network connections

As we can see on the previous image all requirements are achieved so we can move forward. Clicking on ‘Configure’ we are being redirected to a page where we can select which machines we want to use in our cloud environment as well as the software we want to use:



Configure region1 in cloud1

Choose your configuration
Select from the components below to configure your region.

Compute

KVM
KVM (Kernel-based Virtual Machine) is a full virtualization solution for Linux

Network

Open vSwitch
Open vSwitch is a multilayer virtual switch enabling network automation

Object Storage

Ceph
Ceph provides seamless access to objects using native bindings or a REST interface

Swift
Swift is a highly available, distributed, eventually consistent object/blob store

Block Storage

ISCSI
Persistent volumes over ISCSI, an IP-based storage networking standard

Ceph
Ceph's RADOS Block Device provides device images that are striped and replicated

Add hardware
Select the hardware you wish to add to your availability zone

To add hardware, please select from the physical zones available. default

Machine	Cores	RAM (GiB)	Disk (GB)
node1780.maas	2	4.0	330
node2780.maas	2	3.0	330
node3780.maas	2	4.0	330
node4780.maas	2	12.0	280
node1dc7900.maas	2	4.0	160
	10/10 Cores	27.0/27.0 RAM (GiB)	1430/1430 Disk (GB)

Configuration
- Select compute
- Select network
- Select object storage
- Select block storage

Hardware
- Add at least three machines
- Add at least one machine on the public network

- Select network
- Select object storage
- Select block storage

Hardware
- Add at least three machines
- Add at least one machine on the public network

Figure 39: Openstack cloud landscape gui B

3.2.10. Openstack cloud install

In this section we are going to configure the landscape tool for OpenStack and install the cloud, deploying all the nodes with the software that it needs to get our VMs up and running and get our storage. It consists on selecting everything it takes to get the cloud going. To do that we are going to select the folowwing components:

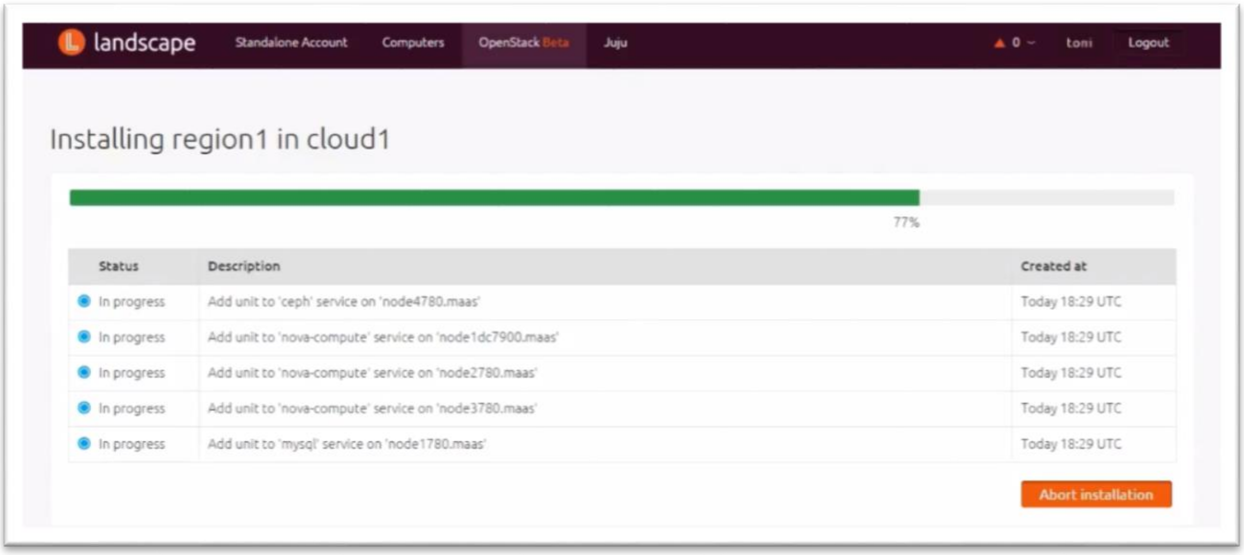
- **KVM** (Kernel-based Virtual Machines) for *Compute*. KVM is a virtualization infrastructure for the Linux kernel that turns it into a hypervisor. KVM requires a processor with hardware virtualization extensions and allows you to run virtual machines using disk images that contain unmodified operating systems where each virtual machine has its own virtualized hardware: a network card, hard drives, graphics card and so on.

- **Open vSwitch** for *Network*. OpenvSwitch, sometimes abbreviated as OVS, is an open-source implementation of a distributed virtual multilayer switch. The main purpose of Open vSwitch is to provide a switching stack for hardware virtualization environments, while supporting multiple protocols and standards used in computer networks. It is going to use the default configuration from MAAS.

- **Ceph** for *Objet Storage*. Ceph provides seamless Access to objects using native bindings or a REST interface. It is written to be compatible with applications written for S3 and Swift

- **Ceph** for *Block Storage*. Ceph’s RADOS Block Device provides device images that are striped and replicated. RADOS Block Devices (RBDs) integrate with Kerner Virtual Machines, bringing storage to VMs running on Ceph’s clients. Ceph automatically stripes and replicates the data across the cluster.

At this point we are going to let OpenStack have all the boxes unless the first node one, which has plugged into the public network as we have said before. So the first one goes to “In progress” status and it took at least 20 mintues.



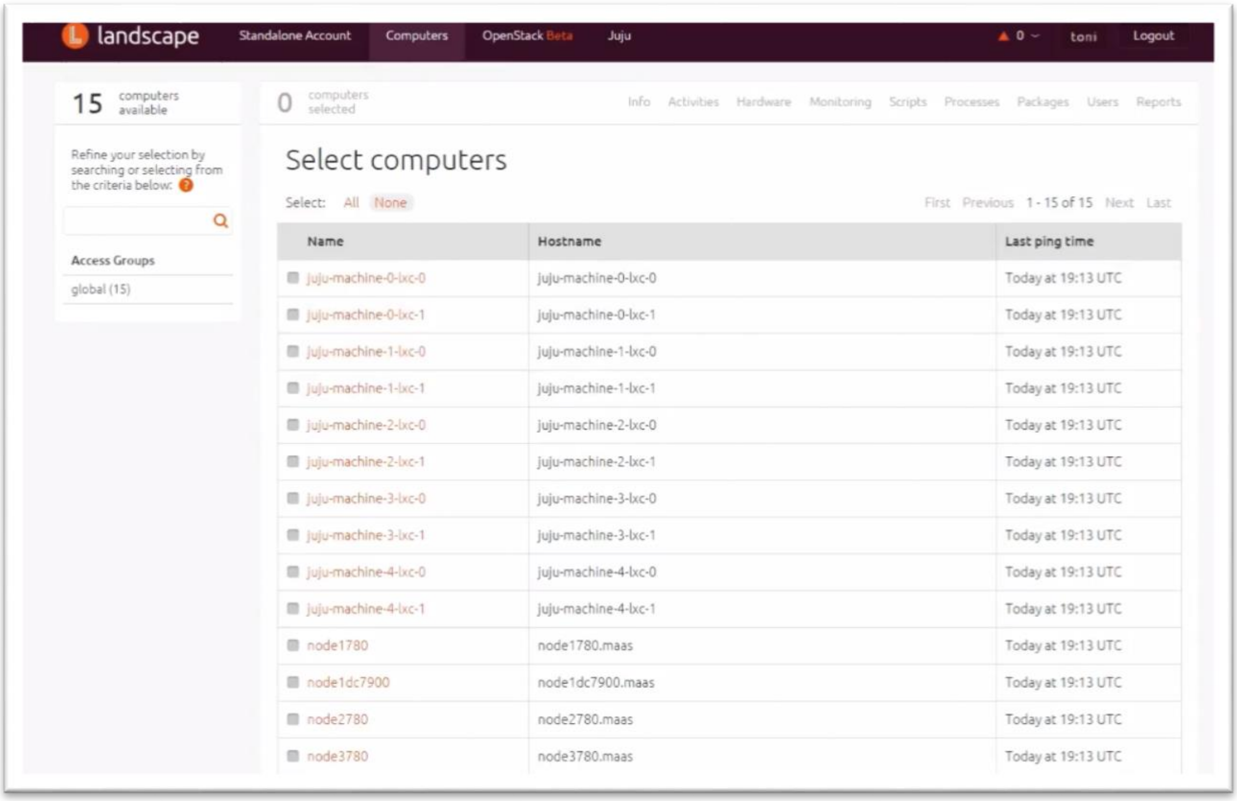
Installing region1 in cloud1

77%

Status	Description	Created at
In progress	Add unit to 'ceph' service on 'node4780.maas'	Today 18:29 UTC
In progress	Add unit to 'nova-compute' service on 'node1dc7900.maas'	Today 18:29 UTC
In progress	Add unit to 'nova-compute' service on 'node2780.maas'	Today 18:29 UTC
In progress	Add unit to 'nova-compute' service on 'node3780.maas'	Today 18:29 UTC
In progress	Add unit to 'mysql' service on 'node1780.maas'	Today 18:29 UTC

Figure 40: Openstack cloud landscape installing process

In the screenshot above there is a list of all virtual machines (the ones with '*lxc'* are containers), so it is running on that first virtual node that landscape installed originally. When this process is finished the OpenStack cloud install has been completed and the cloud is now available.



Refine your selection by searching or selecting from the criteria below:

Select computers

Name	Hostname	Last ping time
juju-machine-0-lxc-0	juju-machine-0-lxc-0	Today at 19:13 UTC
juju-machine-0-lxc-1	juju-machine-0-lxc-1	Today at 19:13 UTC
juju-machine-1-lxc-0	juju-machine-1-lxc-0	Today at 19:13 UTC
juju-machine-1-lxc-1	juju-machine-1-lxc-1	Today at 19:13 UTC
juju-machine-2-lxc-0	juju-machine-2-lxc-0	Today at 19:13 UTC
juju-machine-2-lxc-1	juju-machine-2-lxc-1	Today at 19:13 UTC
juju-machine-3-lxc-0	juju-machine-3-lxc-0	Today at 19:13 UTC
juju-machine-3-lxc-1	juju-machine-3-lxc-1	Today at 19:13 UTC
juju-machine-4-lxc-0	juju-machine-4-lxc-0	Today at 19:13 UTC
juju-machine-4-lxc-1	juju-machine-4-lxc-1	Today at 19:13 UTC
node1780	node1780.maas	Today at 19:13 UTC
node1dc7900	node1dc7900.maas	Today at 19:13 UTC
node2780	node2780.maas	Today at 19:13 UTC
node3780	node3780.maas	Today at 19:13 UTC

Figure 41: Openstack computers available list

4. Budget

In this section the costs of the project have been calculated.

The direct costs calculation includes the costs of hardware, software and human resources. For the calculation of hardware and software resources it has been taken into account their useful life and the number of hours that will be used during the project. Its important to mention that we worked with free and open-sources software platforms, so there will not be any cost reflected from them.
The human resources of which the project is composed corresponds to the amount of hours carried out by a junior engineer from the Polytechnic University of Catalonia (UPC), with a cost per hour of 8 euros.

Regarding the indirect costs, the cost of electricity and the ADSL fee have been taken into account during the process of developing the project with an approximate duration of 6 months and they represent a 40% percentage of the total cost of the project. To make this calculation it has been considered the work zone of the Laboratório de Apoio a Dissertação of the Departamento de Engenharia Electrotécnica da Faculdade de Ciências e Tecnologia (FCT) and the developer's private work zone.

	Price (€)	**Years**	**Amortization/hour (€)**	**Used (h)**	**Cost (€)**
Direct costs					
ASUS PC	1.200	5	0.0148	540	17.76
7 x Dell optiplex 780 Q45 Chipset motherboard VTx capable	1.925	5	0.01924	200	37.03
2 x D-Link DAP 2690	1.434	5	0.00493	500	7.07
2 x HP E3800-24G-POE+-2ESFP	4.100	10	0.00147	500	6.02
3 x HPE ProLiant DL 160 Gen9	5.100	10	0.00918	500	46.82
Microsoft Office	122	3	0.00132	540	0.161
Atenea UPC	1	0	-	0	0
PDF viewer	1	0	-	0	0
OpenStack	1	0	-	0	0
MAAS	1	0	-	0	0
Gantt Project	1	0	-	0	0
Human resources (Telecom engineer)	8	-	-	540	4.320
TOTAL DC					4.435
Indirect costs (40%)					1.774
TOTAL					**6.209**

Table 2: Budget

5. Conclusions and future development:

In this thesis, the vision of future networks as well as its challenges has been displayed. Also it has been showed how the future of data centers passes necessarily through Cloud Computing information technology paradigm.

In order to understand properly the different tools handled during this project, important concepts such as Cloud Computing, Software Defined Networking, Metal As A Service and OpenStack have been defined and studied, acquiring in that process a valuable knowledge about management of data centers.

The necessary information regarding MAAS and OpenStack is presented to guide the reader throught the entire learning process of creating and implementing a private cloud in a data center.

The solution implemented in this thesis allows to group some virtual machines into virtual nodes and provide the connectivity between them with a user-specified bandwidth.

As a future development, the interconnection between the SDN data transporting Network and the Data center network can be designed and implemented to allow the separation of the network control functions running in the cloud from user workloads. This process can be realized using SDN OpenDaylight as the controller to be introduced into the deployment of OpenStack.

Bibliography

[1] Livewire: for live carreers. Secure a sparkling career in cloud https://www.livewireindia.com/cloud_computing_training.php

[2] Cosign | Advanced Optical Hardware and SDN for Next-Generation Data Centres

http://www.fp7-cosign.eu/

[3] The NIST Definition of Cloud Computing http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[4] GlobalDots: Cloud computing Types of Clouds

https://www.globaldots.com/cloud-computing-types-of-cloud/

[5] Cloud computing types

https://aws.amazon.com/es/types-of-cloud-computing/

[6] Crucial broadcast: Types of Cloud Computing http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[7] Openstack community. Official webside:

https://www.openstack.org/

[8] Cloud service management and cloud monitoring for providers

http://searchcloudprovider.techtarget.com/feature/Cloud-service-management-and- cloud-monitoring-for-providers-A-primer

[9] OpenStack Foundation, *Chapter 1. Architecture OpenStack:*

http://docs.openstack.org/juno/install-guide/install/apt/content/ch_overview.html

[10] OpenStack Foundation, 2014. *All-In-One Single VM* http://docs.openstack.org/developer/devstack/guides/single- vm.html

[11] OpenStack documentation:

http://docs.openstack.org/queens/

[12] https://maas.io/how-it-works

[13] Manually deploying openstack with a virtual maas on ubuntu trusty:

http://chrisarges.net/2014/06/13/manually-deploying-openstack-with.html

[14] http://172.16.4.50/MAAS

[15] Transforming your physical infrastructure into a cloud:

https://www.ubuntu.com/server/maas

[16] https://jujucharms.com/

[17] Juju software:

https://es.wikipedia.org/wiki/Juju_(software)

Glossary

MAAS:	Metal As A Service
VM:	Virtual Machine
SDN:	Software Defined Network
RAM:	Random Access Memory
CPU:	Central Processing Unit
KVM:	Kernel-based Virtual Machine
LXC:	Linux Container
SLA:	Service Level Agreement
PXE:	Preboot Execution Environment
OS:	Operative System
IP:	Internet Protocol
DNS:	Domain Name System
REST:	Representational State Transfer
API:	Application Programming Interface
CLI:	Command Line Interface
IPMI:	Intelligent Platform Management Interface
BIOS:	Basic Input/Output System
UI:	User Interface
VLAN:	Virtual Local Area Network
CIDR:	Classless Inter-Domain Routing
IPv:	Internet Protocol Version
DHCP:	Dynamic Host Configuration Protocol
GUI:	Graphic User Interface
YAML:	YAML Ain’t Markup Language
HTTP:	Hypertext Transfer Protocol
DEE:	Departamento de Engenharia Electrotécnica
FCT:	Faculdade de Ciências e Tecnologia
UNL:	Universidade NOVA de Lisboa
USB:	Universal Serial Bus
DOS:	Disc Operative System
NIC:	Network Interface Card
TFTP:	Trivial File Transfer Protocol
XML:	Extensible Markup Language
RADOS:	Reliable Autonomic Distributed Object Storage
OSD:	Object Storage Daemon
AMQP:	Advanced Message Queuing Protocol