Pedro Miguel Guerreiro Ribeiro

Bachelor of Science in Computer Engineering

Evaluation and Deployment of a Private Cloud Framework at DI-FCT-NOVA

Dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Informatics Engineering

Advisers: Paulo Orlando Reis Afonso Lopes,
Assistant Professor, NOVA University Lisbon
João M. S. Lourenço, Assistant Professor,
NOVA University Lisbon

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

April, 2020

Evaluation and Deployment of a Private Cloud Framework at DI-FCT-NOVA

This document was created using the (pdf)LaTeX processor, based in the "novathesis" template[1], developed at the Dep. Informática of FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

To my parents, always unwavering in their love and support.

Acknowledgements

I would like to express my sincerest appreciation and gratitude towards my advisers, Professors Paulo Lopes and João Lourenço, for this great opportunity, as well as all the trust and responsibility they've given me. I would also like to credit them not only for their kindness and support during my work, but also for their constructive and incisive criticism. Together they've allowed me to see the person I am, and steered me towards who I could become.

Furthermore, I would be remiss if I did not acknowledge the institutional support I relied on through this dissertation: A heartfelt thank you to the Faculty of Sciences and Technology of the NOVA University Lisbon, especially the Department of Informatics, for all the great people working there, and for making mine and so many others' academic lives a possibility. I would further like to thank the NOVA LINCS Research Center (UID/CEC/04516/2013) and the HiPSTr Project (PTDC/CCI-COM/32456/2017 & LISBOA-01-0145-FEDER-032456) for supporting this work.

Last but not least, I am grateful for those whose belief in me never wavered: my girlfriend, for being the amazing woman she is, showing nothing but care and understanding; my friends, far and near, who provided advice, moments of laughter, and listened when needed. Thank you for keeping me sane and on the path, and may we have many more years of joy and mirth.

Abstract

In today's technological landscape, there is an ever-increasing demand for computing resources for simulations, machine learning, or other use-cases. This demand can be seen across the business world, with the success of Amazon's AWS and Microsoft's Azure offerings, which provide a cloud of on-demand computing resources to any paying customer.

The necessity for computing resources is no less felt in the academic world, where departments are forced to assign researchers and teachers to time-consuming system administrator roles, to allocate resources to users, leading to delays and wasted potential. Allowing researchers to request computing resources and then get them, on-demand, with minimal input from any administrative staff, is a great boon. Not only does it increase productivity of the administrators themselves, but it also allows users (teachers, researchers and students) to get the resources they need faster, and more securely. This goal is attainable through the use of a cloud management framework to assist in the administration of a department's computing infrastructure.

This dissertation aims to provide a critical evaluation on the adequacy of three cloud management frameworks, evaluating the requirements for a private cloud at the DI-FCT-NOVA, as well as which features of the selected cloud framework may be used in the fulfilment of the department's needs. The final goal is to architect and deploy the selected framework to DI-FCT-NOVA, which will give the department a maintainable state-of-the-art private cloud deployment, capable of adequately responding to the needs of its users.

Keywords: Private cloud, OpenStack, Cloud computing, Cloud management

Resumo

No cenário tecnológico atual, existe uma necessidade crescente por recursos computacionais quer para simulações, aprendizagem automática, ou outros fins. Essa necessidade pode ser vista no mundo dos negócios, traduzindo-se no sucesso da Amazon AWS e a da Microsoft Azure, entre outras, que oferecem clouds de recursos computacionais a qualquer cliente, sujeito a diferentes formas de pagamento.

A necessidade de recursos computacionais não é menos sentida no mundo académico, onde departamentos são forçados a atribuir a investigadores e professores tarefas onerosas que desperdiçam o seu potencial, como administração de sistemas computacionais com o fim de alocar recursos quem deles necessita (docentes, investigadores e estudantes). Permitir que se peçam recursos computacionais, e estes sejam alocados com o mínimo de interacção de uma equipa administrativa, é um grande benefício. Isto não só aumenta a produtividade dos próprios administradores, como também permite que se obtenham os recursos mais depressa, e de forma mais segura. Esta meta é alcançável através do uso de uma framework de gestão de cloud, cujo objectivo é assistir na administração da infraestrutura computacional de um departamento.

Esta dissertação tem como objectivo fornecer uma avaliação crítica da adequação de três frameworks de gestão de cloud, avaliar os requisitos necessários para uma cloud privada no DI-FCT-NOVA, e identificar que funcionalidades da framework selecionada podem ser utilizadas para a satisfação dos requisitos indicados. O objectivo final é desenhar e instalar a framework selecionada no DI-FCT-NOVA, oferecendo assim uma cloud privada de última geração, capaz de responder adequadamente às necessidades dos seus utilizadores - docentes, investigadores e estudantes.

Palavras-chave: Cloud privada, OpenStack, Computação em cloud, Gestão de cloud

Contents

Chapter 1

Introduction

1.1 Cloud Computing

The National Institute of Standards and Technology (NIST) has created a document, "The NIST Definition of Cloud Computing" [32], that distilled contributions from various players in the industry, academia and others. In short it states that "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." Cloud infrastructures may be distinguished according to their deployment type (e.g., public, private, hybrid, etc.) and the type of service they provide (e.g., infrastructure, platform, etc.).

1.1.1 Cloud consumers, tenants and providers

A provider is the organisation that manages the cloud infrastructure and offers its resources to consumers; as an example, Amazon is a public cloud provider, while CERN manages a private cloud infrastructure for its own use.

Consumers are organisations or individuals that have a business relationship with providers and are therefore allowed to consume cloud resources, paying for their use; as an example, a student will pay (or deduct from a grant he/she has been awarded) for an Azure VM, while in a private cloud a department will be cross-charged (in the company's accounting ledger) for the resources that it consumes.

A tenant is a consumer (whether individual or an organisation) of the cloud's services. This tenant is seen by the provider as a unique entity which must be isolated from other tenants by default (e.g., computations cannot leak information, data exchange cannot be snooped or modified, etc.).

In short, cloud consumers, using GUIs, CLIs, or programmatic interfaces gather resources that are used to provide services to end-users, which may be completely unaware of all of the above complexity. These resources (see the NIST definition above) are available on storage, networking and compute pools which are (usually) provided by hypervisors (e.g., KVM [28], Xen [57], vSphere ESXi [61], etc.) and, therefore, virtualised.

1.1.2 Why virtualisation?

In the vast majority of cloud sites, the infrastructure is virtualised before it is made available for consumers. The main reason for this is that virtualisation is i) the most convenient way to provide resources that can be rapidly provisioned and released; and ii) enables a high degree of isolation among those resources, thus tenants cannot interfere with each other.

This raises an important question: if virtualisation does provide shared pools of resources that can be rapidly provisioned and released, why do we need clouds? The answer is quite simple: because virtualisation-based infrastructures do require a fair amount of interaction between “consumers” (developers *et al* that provide the applications for the users) and the infrastructure administrators (system, storage, and network admins) that work together to provide the resources to those consumers. Thus cloud software stacks build on virtualisation capabilities to provide a level of automation that enables self-service portals for consumer use.

1.1.3 Infrastructure Components and Cloud Service Models

The three components of a computing infrastructure/service are: compute (nodes, a.k.a. servers), network and storage. Consumers that want to request these “lower-level” components, specifying characteristics such as number of CPUs, amount of memory, network bandwidth and storage space are addressing the cloud’s capabilities to provide Infrastructure-as-a-Service (IaaS) service.

But sometimes consumers do not wish to be involved with those lower-level characteristics, e.g., CPUs and RAM; instead they want some specific software product (e.g., MySQL) to be provisioned with a certain level of performance and a certain amount of storage space (e.g., 100 GB). In this case, the consumer is addressing the cloud’s capabilities to provide the Platform-as-a-Service (PaaS) service.

The highest level in the NIST-defined Service Model stack is the Software-as-a-Service (SaaS) service, where the required service configurations are so simple that the consumer is the end-user. Google’s Gmail is an example of a SaaS: the consumer (user) just provides some simple data and personal information and, voilà, the service is ready-for-use.

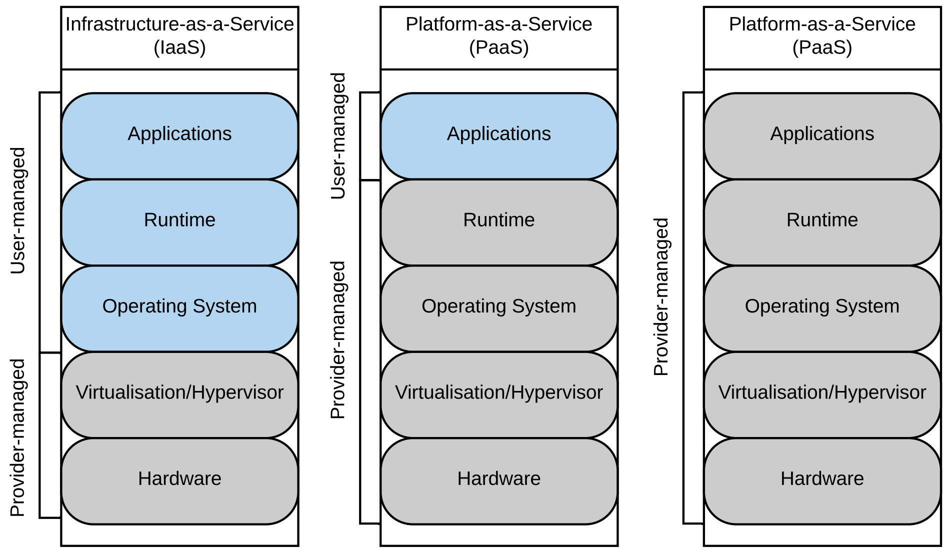
Infrastructure-as-a-Service (IaaS)
Platform-as-a-Service (PaaS)
Platform-as-a-Service (PaaS)
User-managed
Provider-managed
Applications
Runtime
Operating System
Virtualisation/Hypervisor
Hardware

Figure 1.1: NIST Service Models

1.1.4 Cloud Deployment Models

The most important cloud deployment models are public, where a top-level company/organisation (e.g., Google, Amazon, Microsoft) manages a world-wide gigantic infrastructure (100s of thousand of nodes) that is publicly available for anybody, individual or company. On the other side of the spectrum, a private cloud is (usually) managed by an organisation that offers the cloud resources to its own consumers. In between, a hybrid cloud is one where a private cloud, to be able to service infrequent peaks, consumes resources from a public cloud as if they were "local" resources.

1.2 Case Study

The case being studied is that of a research and learning organisation, with its own infrastructure, and a myriad of different resource requirements from researchers, as well as limited budget for maintenance personnel. Given these requirements, the most adequate model for a cloud deployment would be of a private nature, which would minimise maintenance, due to stricter access policies, along with requiring a smaller support burden from any maintainers to the researchers themselves. As for service models, IaaS is the most flexible, however, there may be instances where executing untrusted, student-submitted code — in an environment similar to AWS Lambda or Azure Functions — might be preferable, allowing for greater parameterisation and scheduling capability, at the cost of student freedom.

It is, then, of paramount importance that the selected ecosystem for a private cloud be low-maintenance, flexible, easily serviced, free, and with limited need for support.

The most popular solution in this area is OpenStack [39], a modular, Python-based open-source solution, with support and maintenance by several enterprise-level vendors, as well as healthy development and deployment bases. However, other solutions exist, and so it is worthwhile to study and present them, comparing their features, and evaluating their fitness to our requirements, which is done in Chapter 2.

1.3 Organisation of the Document

In this first chapter, a small introduction to Cloud computing, along with why it is important and in which scenarios it can be relevant in, is laid out. This lays the ground for presenting our case study, that of designing, deploying, and documenting a private cloud at DI-FCT-NOVA, along with a few other expected contributions in the course of fulfilling the goals of this work.

In the second chapter, a very global view of a Cloud solution is given, with its main components, their usage within a cloud context, and their importance to construct a useful resource for users. Along with this, a critical evaluation of three popular cloud management frameworks is made, followed by our arguments for the chosen cloud framework, with emphasis on the most important areas of a cloud: virtualisation, networking, and storage.

In the third chapter, there is an in-depth look at the selected cloud management framework, including architecture, deployment plans, and other relevant information.

In the fourth chapter, a concrete plan for an OpenStack-based private cloud architecture is detailed, in accordance to established requirements, including hardware clusters, networking, and services.

The fifth chapter is dedicated to elaborating on the specific deployment tooling and methodologies used, its details, configuration parameters, as well as justification for some of those configuration options.

Finally, a conclusion is presented, noting some future work that may still be done.

Chapter 2

Related Work

In this chapter, a brief overview of the essential services for a functional cloud infrastructure are presented, as well as an analysis of available solutions that could be deployed.

2.1 Overview of Cloud Infrastructure Services

Any Cloud-based solution will have some critical components in common, some of which will be detailed in an implementation-agnostic manner in this section. These components may be split into AAA (Authentication, Authorisation, and Accounting); Storage, covering all data storage capabilities; Compute, covering virtualisation or containerisation technologies that enable the compartmentalisation of a multi-tenant cloud infrastructure; and Networking, which further isolates different tenants and allows for the potential of defining complex network topologies for a tenant's compute resources.

2.1.1 Authentication, Authorisation and Accounting (AAA) Service

In a cloud infrastructure the AAA subsystem ensures that users are authenticated (who) and, before being granted access to resources (what), they provide the necessary authorisation for the operations they wish to perform (how). The details of those operations are then recorded to be later accounted for. The AAA components can use local or externally provided services (such as databases) that store and retrieve user and group credentials, membership, define roles, policies, privileges, etc., using standard APIs.

The AAA is the basis of tenant isolation in the cloud infrastructure, as it enables resource (compute, storage and networking) segregation among distinct tenants.

2.1.2 Compute Service

The most basic function of a cloud service is to provide some level of compute power to its consumers, whether via serverless functions, which are often limited-purpose procedures to be executed on cloud-managed infrastructure, scaling automatically on a per-demand basis; containers, which serve as a way to package software that's meant to be easy to migrate, but without dealing with operating system and lower concerns; virtual machines, which virtualise a complete computing device, allowing it to be treated like a physical server; and bare metal, which are physical machines that the cloud allocates tenants when the previous options are insufficient; a cloud may also provide some combination of these services, being that each instance of one of these types of compute constitutes a resource that must integrate seamlessly with the AAA service.

2.1.3 Storage Service

The most basic function of the storage system is to provide storage for its consumers, the compute (nodes), in their multiple forms: VMs, containers, and bare-metal hosts. However, a good storage system must satisfy the needs of its consumers and acknowledge that these can be quite diverse: template images (of VMs) are read-only storage; VM instances consume read/write storage that may be provided to the VM host as files or block-based storage; bare-metal hosts require block-based storage (at least for temporary local storage); and user applications may need to share data, and therefore the need for a distributed file system storage service (e.g., NFS, CIFS) is also a common requirement.

These diverse needs may be fulfilled by a one-size-fits-all storage solution or, at the other extreme, by distinct storage providers. Currently, the most popular solution is to have a distributed object-based storage provider (e.g., Ceph [10], ScaleIO [52]) at the backend, offering both object-based (e.g., Amazon's S3) and block-based (e.g., iSCSI [4]) storage protocols, and a middle-tier of file server nodes that offer distributed file-based storage (e.g., NFS, CIFS).

The popularity of this solution stems from the fault-tolerance and scalability of these architectures, as one may independently add more nodes (horizontal scalability) or add resources (disks, RAM, CPU, NICs) to existing nodes (vertical scalability), while fault tolerance is provided by scattering the data (entire "objects", or "segments") across nodes (something that may, or may not, contribute to scalability). Naturally, the storage system components must be fully integrated with the AAA, in order to provide the required separation of tenants, and overall security.

2.1.4 Network Service

In a cloud infrastructure, the network service can be better described if split into two parts: provider networks and consumer networks. Provider networks carry infrastructure management and storage traffic, and enable communication between consumer deployments within geographically dispersed datacentres and the Internet; consumer networks allow tenants to create their own network topologies and addressing plans, define their own routes and firewalling schemes, etc. Flexible cloud infrastructures further allow consumers to specify quality of service (QoS) parameters such as bandwidth, latency, etc. for their "private" networks as well as high availability topologies and VPN services that enable consumers to extend their on-site/on-premises "legacy" datacentres into "the cloud".

Also, in line with the other resource types, network system components must be fully integrated with the AAA, in order to provide the required separation of tenants, as well as overall security.

2.2 Cloud Management Frameworks

From the solutions available in this field, three were chosen for further evaluation due to their open-source nature and wide deployment bases: The OpenStack [39], CloudStack [3], and OpenNebula [55].

2.2.1 OpenStack

OpenStack [39] is a free, modular cloud management framework, which started as a joint project between RackSpace and NASA, later evolving into the OpenStack Foundation [65], but counts on many contributions by large enterprise-grade vendors. As of the latest release, Rocky, we can see the involvement of Red Hat, Huawei, SUSE, Rackspace, Fujitsu, among others [54], which indicates an investment on the part of these companies into the OpenStack framework.

The highly modular approach of OpenStack means that it can be configured and adjusted to virtually any cloud role, including modules to handle networking, block storage, object storage, computing, databases-as-a-service, functions-as-a-service, among others, along with a whole suite of management, monitoring, and metering services.

2.2.2 OpenNebula

Unlike OpenStack, which is led by a committee of corporations, OpenNebula [55] originated in the academic world [34] and is led by the researchers who started the project, under OpenNebula Systems. It aims at providing a management framework that is community-driven, with goals that are set independently of funding sources.

Instead of a modular approach, OpenNebula boasts a very tightly-knit and well-integrated set of tools, with minimal configuration and customization needs, at the cost of the more widespread feature-set of OpenStack.

2.2.3 CloudStack

CloudStack [3] was developed by Cloud.com, a startup funded by venture capital, as a proprietary and closed-source solution. However, it was later released as an open-source product and acquired by Citrix. Nowadays, it is owned and administered by the Apache Software Foundation.

CloudStack seeks to provide a fully-integrated top-down solution, with all the necessary tools to provision multi-tenancy-capable computing nodes, along with commonly-used networking features such as DNS, DHCP, VLANs, among others, boasting high compatibility with all mainstream-usage hypervisors. Unlike both OpenNebula and OpenStack, however, CloudStack is Java-based.

2.3 Overview of Core Cloud Features

Outside a very small core set of common features, these frameworks' feature-sets can be widely divergent, and prone to implementing things their own way, rendering direct comparisons of features difficult or even outright impossible. This section is intended to provide a comparison, where possible, between the three frameworks, in areas deemed important and relevant for our purposes.

2.3.1 Virtualisation

In order to better understand the different virtualisation capabilities of these frameworks, the concepts of hypervisors and containers need to be introduced, and distinguished from each other. Hypervisors and containers can be more easily viewed as virtualisation solutions for different levels of the computing stack — hardware, and operating-system, respectively.

Hardware-level virtualisation seeks to create a virtual machine (VM). This VM seeks to present a predefined set of virtual resources (such as CPU and RAM) to any software executing within. The software running inside the VM has no access nor knowledge about the environment outside the VM itself, such as whether the VM is executing in a shared environment, or what the real hardware is. Management of these virtual machines is done by a virtual machine manager (VMM) [47], also called a hypervisor. Private clouds have made extensive use of this technology, with some examples of common hypervisors being KVM [28], Xen [57], and Hyper-V [23].

Operating-system level virtualisation creates containers, which are similar to virtual machines in that they attempt to present a full computer to software running within, however, unlike hardware-level virtualisation, this hardware is not virtualised and corresponds to a configured subset of the real hardware. This means that there is little to no compartmentalisation and isolation between containers at the hardware level, because they rely on process-level isolation, with support from the operating system. Modern private clouds have been increasingly supportive of containerisation technologies, since, where lack of proper isolation and compartmentalisation is not a priority, containers are much faster to orchestrate, consume less resources overall (since they share the operating system with the host), and are easy to deploy and destroy. Examples of containerisation technologies are LXC [29], OpenVZ [44], Windows Containers [64], and Docker [14].

As such, a wide support for virtualisation technologies is healthy in a cloud management framework, giving the deployers a varied array of choice among virtualisation styles and feature sets, as well as the possibility of having clusters with different virtualisation technologies available. In Table 2.1, it is indicated whether a cloud management framework supports a given virtualisation technology. It is important to remark that OpenNebula's support for LXC is provided by a third-party add-on, and that Bare Metal, while not a virtualisation technology per se, is included, since the provisioning of bare machines, sans virtualisation, is desirable in some scenarios.

Table 2.1: Virtualisation Support

	OpenStack	**OpenNebula**	**CloudStack**
KVM	✓	✓	✓
Citrix XenServer	✓	—	✓
Xen	✓	✓	✓
VMware	✓	✓	✓
Oracle VM	—	—	✓
Hyper-V	✓	—	✓
LXC	✓	3rd party	✓
Bare Metal	✓	—	✓

To demonstrate the importance of virtualisation technology support, picture a scenario with a pre-existing cluster with a VMware vSphere deployment. An administrator that wishes to integrate this existing cluster with a newly deployed cloud management framework, may only do so if that new framework supports VMware. If so, integration is straightforward, and the cloud management framework will utilise the old cluster as-is, or with some minor configuration changes. Whereas, if VMware is not supported, then either the administrator or the cloud management framework will have to provision this cluster with a supported virtualisation technology, which will overwrite any VMware software present. Integrating existing clusters with supported virtualisation technologies is supported across all three cloud management frameworks presented, as long as the same virtualisation technology is used in the existing cluster.

As can be observed in Table 2.1, OpenNebula supports the smallest number of virtualisation technologies, and CloudStack the most, while OpenStack lacks support for Oracle VM. Oracle VM is a proprietary branch of the open-source Xen hypervisor, also known as Oracle VM Server, published by Oracle. If a varied virtualisation support is required, OpenNebula would be a poor choice, especially for its inability to provision bare metal machines, an important use-case for many clouds, especially in the realm of research and High-Performance Computing.

2.3.2 Networking

Our next comparison criteria is the supported networking features, which are particularly important in scenarios where existing networking equipment needs to be utilised and integrated, and the acquisition of new enterprise-grade equipment is not an option.

Within the realm of cloud networking, there are a few technologies and concepts which are relevant to introduce. Firstly, as referenced in Sect. 1.1.1, tenants necessitate isolation, not just in the domain of computing (as provided by virtualisation technologies), but also while communicating between distinct virtualisation units (VUs), such as virtual machines, and containers, belonging to that tenant. For the tenant, it is irrelevant whether all their VUs are contained in a single discrete host, or spread across a whole datacentre, as long as their communications are private, and inviolable. In order to facilitate these privacy requirements, a few technologies may be used.

The most basic method to isolate traffic in the same network are virtual LANs, or VLANs, as defined in IEEE 802.1Q [1], where packets are identified by a special, 12-bit field in the Ethernet frame, which allows virtual switches within each host to uniquely identify which tenant a given packet belongs to. Although limited to a theoretical 4095 tenants, with one tenant per VLAN, this solution is practical for smaller private clouds, given that it is unlikely such a cloud will hit the tenant limit. Beyond the restricted number of simultaneous tenants, another disadvantage of this method is that VLAN identifiers do not carry over the public Internet, which would force multi-region private clouds to employ traffic tunneling technologies to communicate.

Another method that is also commonly used, and was referenced in the last paragraph, is tunneling protocols. The supported tunneling protocols are normally Virtual eXtensible LANs (VXLANs) [31], and Generic Routing Encapsulation (GREs)[15]. VXLAN encapsulates Ethernet traffic in UDP, referred to as MAC-in-UDP, while GRE has its own protocol defined in IP (type 47), and it encapsulates IP-in-IP. These tunneling protocols, as the name implies, permit encapsulating (and transporting) types of traffic meant for one network within a different network. Examples of usage are Virtual Private Networks (VPNs), which allow traffic meant for a private network to be transported, albeit encrypted, over the Internet. In a cloud setting, however, the private network would be a tenant's sectioned subnetwork, whereas the Internet could be any outer network, from the datacentre's private network, to the Internet itself, in the case of multi-regional cloud deployments.

It is also important to note that, since automated instrumentation and orchestration is extremely desirable in the modern cloud-based datacentre, the use of scriptable and automatable hardware technology is a great boon. However, since the costs of such technologies are commonly quite high, frameworks that can exploit them to their maximum potential have a natural advantage over those that do not. Normally, this is done via cooperation of third-party vendors, since these technologies are highly proprietary, and so a large deployment-base is desirable to attract third-party extension development and integration.

Table 2.2: Networking Support

	OpenStack	**OpenNebula**	**CloudStack**
Flat Network	✓	✓	✓
VLAN	✓	✓	✓
VXLAN	✓	✓	✓
GRE	✓	Via OpenVSwitch	✓
SR-IOV	✓	3rd party	✓
Switch Management	✓	—	—

Each of the presented frameworks implements, at the very least, a flat networking model, a VLAN-segregated model, and a VXLAN tunneling model. The flat model is not recommended for production environments, since all traffic coexists on the same subnet, which means that any tenant can monitor any other tenant's traffic, as well as management/administration traffic. This means that the flat networking model's usage scenario is for testing purposes, or for cloud networks where tenant isolation is not required, or wanted. In the VLAN-segregated model, an allocation of VLAN identifiers is made on a per-tenant basis, as well as allocating a range of IDs for managerial traffic.

OpenStack's Neutron supports the widest amount of network configurations, with its concept of drivers, it can also control external network hardware by itself, meaning that maintainers do not need to configure any switches or routers themselves. Neutron employs two concepts; type drivers and mechanism drivers. Type drivers indicate the type of the underlying networking layer, and may be flat, VLAN, VXLAN, or GRE tunnels. As for the mechanism drivers, they indicate what kind of mechanisms are used to implement the networking facilities indicated by the selected type driver. These may be Linux bridges, Open vSwitch, Single-root input/output virtualisation (SR-IOV), MacVTap, as well as external Software-Defined Networking (SDN) frameworks, such as OpenDaylight, and OpenContrail. In addition to these mechanisms, an extra one exists, L2 Population, which is meant to be used in combination with VXLAN or GRE tunnels, as well as either Linux Bridging, or Open vSwitch. This extra mechanism seeks to locally populate the forwarding tables, so that ARP broadcasts do not need to be "emulated" via unicast across the whole topology, in cases where it is not necessary.

In OpenNebula and CloudStack, less features and customisation are supported natively, with OpenNebula lagging behind due to CloudStack's support of Nicira's Network Virtualization Platform (NVP), and Nuage's Virtualized Services Platform (VSP) through plugins, although this is limited to a restricted set of virtualisation technologies.

2.3.3 Storage

In a cloud-based system, storage is an extremely important topic because data is often difficult to replace when failure occurs, and inefficiencies in data locality and access may lead to a slowdown in computational tasks. Another very important characteristic of clouds is that storage is *shared* — that is, a “single” storage repository is shared with a large number of clients (hosts and/or VMs).

Cloud deployments should consider two distinct categories of storage: ephemeral and permanent. Ephemeral storage is, by its very design, perishable, which means that it will not be persisted, and will thus be destroyed at some point in the future. A common use-case for this type of storage is the usage of virtual machines’ storage drives as temporary data stores, when the consumer is only interested in the end-result of the computation being run. Permanent storage, on the other hand, is intended for data that should be available up to the moment when it is explicitly destroyed, and perhaps the most common case for cloud-based storage. Storage may be implemented in three distinct ways [22]; object [33], block, and file-based storage.

Object storage (OBS), as the name implies, represents data as objects with a unique identifier; these objects contain the data itself, and some metadata, which has further information about the object that may be relevant to either the storage system (e.g., replication data), or to the data consumer itself (e.g., how to decode the stored data). Object stores are often used for large volumes of unstructured data which need to be replicated, and available with some form of consistency; they contain, e.g., user and virtual machines’ data such as VM templates and images.

Block storage is a more traditional approach to computing storage, where a discrete number of storage blocks is allocated, as a volume, to the consumer, which may then format it as a filesystem. If back-ends such as Gluster [20] and Ceph are used, then block storage (which is provided by objects created on top of these OBS systems — e.g., through Ceph’s reliable autonomic distributed object store, RADOS [62]) may be replicated and distributed. In cloud and environments the main goal of block storage is to be used as volumes available to the VMs — for persistent user data inside or as system drives (persistent or ephemeral).

Finally, there is the file-based storage paradigm which, in these environments, is usually provided by distributed, client-server filesystems such as NFS and CIFS. This type of storage is commonly used either as primary storage, or to supplement block- or object-based storage in virtual machine environments.

All these technologies have a common characteristic: the storage provider is not internal to the compute nodes — an exception is when, e.g., ephemeral storage is provided on local disks (usually SSDs) to run images that are destroyed as soon as the VM is halted. However, these technologies still aim at having low latency and ease of use while providing the consumer with their data. It is then important to pick the appropriate protocols and implementations underlying these storage architectures, as well as educating the consumer on the best use-cases for each of them, often by having appropriate default options in the provisioning UIs themselves.

In OpenStack object storage is supported through Swift [41], which is an eventually consistent object storage. Swift may have its back-end replaced by Ceph, among others. OpenNebula, on the other hand, does not have its own native object storage, suggesting the usage of independent projects for that purpose (e.g., Ceph [10], Riak [51], Gluster[20], et al). Finally, CloudStack supports Swift and Amazon's S3 natively for its secondary storage capabilities, which are used to store VM templates, ISO images, and disk volume snapshots.

Block storage is supported in OpenStack by Cinder [38], which acts as a wrapper for technologies such as LVM [30] and Ceph, providing a unified API for block storage availability to its other modules. OpenNebula natively supports block storage as well, with drivers for Ceph and LVM available, as well as iSCSI [4] support. CloudStack, on the other hand, supports NFS, iSCSI, and RBD (RADOS Block Device), with CIFS being supported exclusively by the Hyper-V hypervisor.

In short, file-based storage seems to be supported in all frameworks, with OpenStack implementing it under the Manila project, acting as a front-end to Ceph, as well as other file-based storage technologies such as NFS and CIFS.

Table 2.3: Storage Support

	OpenStack	**OpenNebula**	**CloudStack**
NFS	✓	✓	✓
iSCSI	✓	✓	✓
Ceph/RBD	✓	✓	✓
Amazon S3	✓	✓	Only Secondary
CIFS	✓	—	Only Hyper-V

2.4 Choosing a Cloud Management Framework

All the frameworks presented in Sect. 2.2 have advantages and disadvantages when compared against each other. OpenStack, for all its feature-set and modularity, seems to suffer from fragmentation in its documentation and has a more complex deployment process. OpenNebula streamlines this process at the cost of integration with adjacent technologies, choosing instead to support restricted subsets of the overall cloud ecosystem, with the perceived notion that this will produce a more robust, albeit smaller, solution. CloudStack seems to severely lack documentation and practical examples, possibly due to the smaller user-base when compared to the other two solutions. Its administration by the Apache Software Foundation makes it attractive from a feature and stability point of view.

However, its low adoption and difficulty when traversing its documentation renders it, at first sight, a poor choice.

In the end, OpenStack was selected. OpenStack has wide adoption in the commercial world, as well as some in the academic world. Its modularity means that integration with existing systems, modification of existing modules, and even creation of new modules in the future, might be a great boon. In addition, its wide community and healthy proprietary distributions, from companies such as Canonical, Red Hat, and Mirantis, warrants support services in case of deployment difficulties or cloud upgrades in the future. Furthermore, if this solution proves practical and worthy of investment, there are many consultancy options available to work with.

Chapter 3

OpenStack-based Cloud Solution

OpenStack is a free, and open-source, modular cloud-computing platform that allows organisations to build and maintain private cloud infrastructures. Although the initial objective was to provide an IaaS platform, the highly modular approach of OpenStack means that it can grow and be configured and adjusted to virtually any cloud role; therefore, modules that were created to support IaaS such as computing, storage (both block and object-based) and networking, are now joined by others of a more PaaS flavour such as databases-as-a-service and functions-as-a-service, among others, along with a whole suite of management, monitoring, and metering services.

OpenStack releases new versions on a 6-month schedule, with each new version having a different codename. At this time, the current stable version is Train, and the next version, tentatively scheduled for a May 2020 release, is Ussuri. Releases have four phases: Development, Maintenance, Extended Maintenance, and End-of-Life. Maintenance lasts approximately 18 months from the release date, and Extended Maintenance lasts for as long as there are community members willing, and able, to maintain that release. End-of-Life releases are no longer maintained or supported at all, and should be upgraded as soon as possible.

3.1 OpenStack Modules

The great selling feature of OpenStack is its modularity. The first release, Austin, was launched in 2010 and had two modules: Nova and Swift. These are, respectively, the computing and object storage resource allocation modules. Since then, many more modules have been added to OpenStack.

Modules are also often called services or projects, in the OpenStack documentation, and can be divided in several categories.

3.1.1 Compute

Compute modules' role is to manage virtualised computing resources in an OpenStack cloud, dealing with the whole lifecycle of virtualised units. There are three modules in this category: Nova, Zun, and Quinling. Nova handles virtual machine infrastructures, resorting to compute drivers that interface Nova with supported hypervisors. Zun's drivers, on the other hand, deal with container infrastructures; the default driver interfaces with Docker. Finally, Quinling provides serverless functions (similar to AWS Lambda or Azure Functions) by resorting to a plugin-like system for runtime environments, each supporting a different programming language. These runtime environments may be created and subsequently added at any point.

3.1.2 Bare Metal

In contrast with Compute modules, Bare Metal modules deal with physical, instead of virtualised resources. This is important since, to provide virtualised resources, physical resources such as compute nodes must be provisioned and managed. There are two modules in this category: Ironic, and Cyborg. Ironic enables OpenStack to provision bare-metal nodes in the same way that compute modules allow the provisioning of virtualised units. Cyborg gives Compute modules the capability of provisioning virtualised environments with access to accelerator engines such as GPUs and FPGAs. Ultimately, however, Bare Metal modules provide compute capabilities, even if they are not virtual, and so are considered part of the OpenStack compute system.

3.1.3 Storage

Storage modules allow OpenStack to interface and provide storage to the infrastructure - (physical) compute nodes, virtual machine instances, management nodes, etc.; modules in this section are Swift, Cinder, and Manila. Swift provides an eventually-consistent object store, ideal for unstructured data, spread across multiple regions. Cinder is OpenStack's block storage module, capable of interfacing with multiple different backends, such as Ceph and ScaleIO [52]. Finally, Manila provides file-based storage, such as NFS and CIFS shares, which are consumable by provisioned nodes, whether virtual or physical.

3.1.4 Networking

OpenStack's Neutron, Octavia, and Designate networking modules provide users with scalable network capabilities. Neutron is a fundamental IaaS service, since it provides virtual network capabilities, managing virtual switches, vNICs, Quality of Service, among others, with the capability of interfacing with Software Defined Networking software such as OpenDaylight SDN controllers. Octavia and Designate, on the other hand, provide higher level networking services typical of PaaS: Octavia is a Load-Balancing-as-a-Service provider, similar in functionality to Amazon's Elastic Load-Balancing and Azure's Load

Balancer, whereas Designate is a DNS-as-a-Service, with multi-tenancy awareness and support for third-party DNS server software.

3.1.5 Shared Services

Shared services is an umbrella category, incorporating modules that integrate with other modules to provide additional functionality, or new functionality, but do not stand on their own. Modules in this category are Keystone, Barbican, Glance, Karbor, and Searchlight.

Keystone is OpenStack's identity service, providing authentication and authorization for multi-tenancy systems, with support for protocols/APIs such as LDAP, OAuth, SAML, and SQL; other OpenStack modules that require authentication services to carry on their tasks, integrate with Keystone.

Barbican is a Key/Secret storage service which may be used to store certificates, cryptographic keys, raw binary data, or any data that could be considered too sensitive and privileged to be stored elsewhere, e.g., in other modules' local storage. Barbican is integrated with Keystone.

Glance is an Image service, providing Virtual Machine images, or templates, integrating with Nova's provisioning capabilities and using Swift for storage (recall that Swift itself can use backend storage services such as Ceph).

Karbor is an Application Data Protection service, which assists in protecting OpenStack deployed applications against data-loss or damage, with a set of addons for each specific application type.

Finally, Searchlight is a search/indexing module, integrating with a myriad of other OpenStack modules, to provide a unified interface where users may search for resources, deployments, containers, storage units, et al., in a seamless way, across a whole OpenStack cloud, or even beyond — in AWS and Azure, for instance — provided that modules that integrate with those public clouds are deployed in the OpenStack infrastructure.

3.1.6 Orchestration

Orchestration modules allow a more automated and streamlined OpenStack cloud usage, whether by allowing the specification of resources to be allocated via configuration files, or providing tools to better orchestrate services through events and templating. The relevant modules that provide ochestration services are Heat, Senlin, Mistral, Zaqar, Blazar, and Aodh.

Heat interconnects with other OpenStack modules, allowing their orchestration and specification via a simplified text format.

Senlin enables resources managed by the Compute modules to be grouped and administered in a block instead of individually. Senlin also allows for automatic scaling-in and scaling-out upon event triggers, to prevent resource waste.

Mistral is a workflow-management module, where workflows are interdependent tasks described in YAML-based documents, to be handled and executed by the Cloud, exploiting all resources provided by OpenStack.

Zaqar is OpenStack's messaging service, similar to Amazon's Simple Notification Service (SNS) and Simple Queue Service (SQS), allowing decoupled components of a tenant's resource pool to communicate with each other, as well as with other OpenStack modules integrating with Zaqar.

Blazar permits consumers to schedule the reservation of resources, in a time-based fashion, to better utilise the Cloud's resource pool, and minimise waste.

Finally, Aodh is an alarming service, which triggers actions based on certain events or parameters, enabling the Cloud to react to ongoing events in an automated fashion, or even notify administrators when certain conditions are met.

3.1.7 Workload Provisioning

Workload provisioning modules facilitate the allocation of clusters of specific, specialised resource classes, such as container clusters and big-data frameworks. Modules that provide these services are Magnum, Sahara, and Trove.

Magnum provides container orchestration engines (such as Docker's Swarm, Kubernetes, and Apache's Mesos) as first-class citizens of OpenStack, allowing for the provisioning and management of these systems by consumers, much like other Computing resources.

Sahara deals with the provisioning of 'Big Data' clusters supporting technologies such as Hadoop, Spark, and Storm, with tenants being capable of defining framework versions, cluster topologies, and hardware details, among other specifications.

Trove is a Databases-as-a-Service provider, with support for both relational and non-relational database engines.

3.1.8 Application Lifecycle

These modules provide useful quality of life features, allowing for a more autonomous and complete management of cloud resources in specific scenarios; modules in this category are Masakari, Murano, Solum, and Freezer.

Masakari is OpenStack's high availability module, allowing for automatic recovery of failed provisioning processes as well as virtualised resources.

Murano exposes a pre-built application catalog, allowing for the publishing and utilisation of cloud-ready applications in a simple, streamlined process.

Solum aims to simplify the integration of a Cloud-ready development environment, allowing developers to link their application development cycle to the cloud, and the automation of cloud deployments. Freezer is OpenStack's data backup and disaster recovery module, allowing for job synchronisation, snapshots and incremental backups, among others.

3.1.9 OpenStack Horizon

Tying all of OpenStack's modules together is a frontend module, Horizon. Horizon features a web-based UI, and it seeks to ease the administration and use of an OpenStack cloud infrastructure. It is supported by most, if not all, previously mentioned modules.

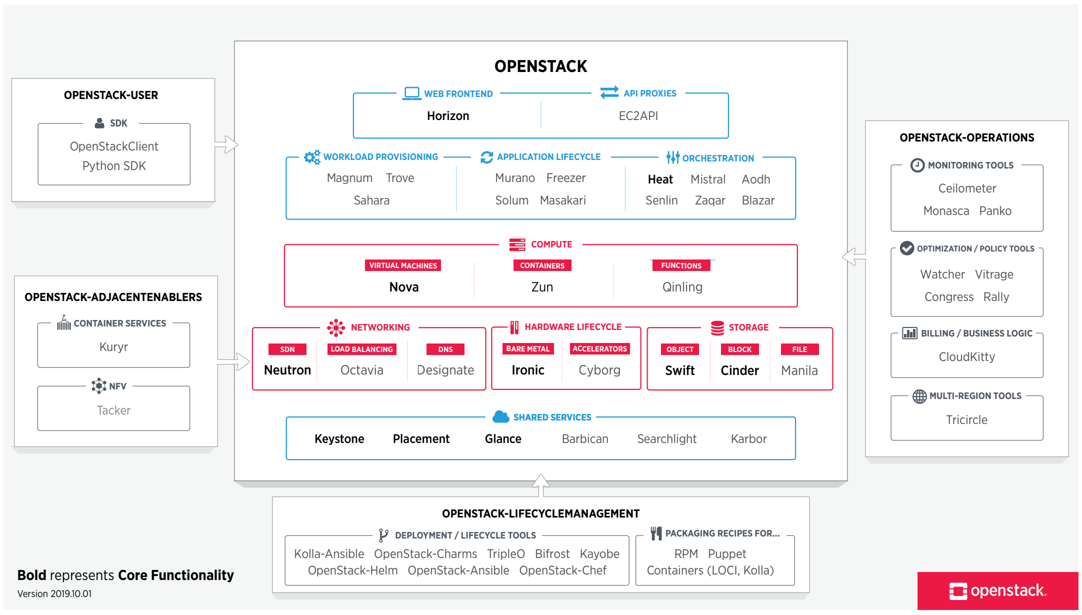

Figure 3.1: Overview of OpenStack Modules

3.2 Deployment

The deployment of a cloud management framework is a very complex task, and OpenStack is known for not being an easy framework to work with, at this level. Deployment involves a series of steps, normally defined in a chosen deployment method's documentation, which starts by picking a Linux distribution to work with, installing the required base packages, and performing the first configuration of the chosen modules, along with credential generation; further steps include identifying other hardware and/or services in the network, et al.

There are multiple possible avenues to deploy OpenStack, including proprietary branches of OpenStack itself, and some methods of making test deployments for evaluation purposes. Choosing one of these avenues requires meaningful consideration, especially given that the cloud needs to be easily maintainable. After all, the selected deployment method will also impact the lifecycle management of the cloud, such as updates and upgrades, recovery in case of catastrophic failures, among other concerns, and one that cannot be easily changed without a completely new deployment, from scratch.

Other avenues for deploying OpenStack use IaC (Infrastructure as Code) software such as Ansible [2], Puppet [48], or Juju [27]. We consider these options as backup plans, due to the amount of knowledge about the underlying paradigm and instrumentation software.

Another category are proprietary installations, from companies such as Canonical, Red Hat, SUSE, and Mirantis. While some of these offerings are not free, the ones that are can be quite attractive, and deserve consideration. Two of the largest offerings in this area are Canonical's Metal as a Service (MAAS) deployment [7], and Red Hat's RPM Distribution of OpenStack (RDO) deployment [49].

Finally, there are deployments of OpenStack as containers, and also OpenStack deploying itself, through a minimal, scripted install base. All of these paths are valid, but without hands-on experience and testing, it is difficult to make a choice regarding how easy are the deployment and the subsequent maintenance processes. Thus, we aim at evaluating some of these deployment methods, with special focus on OpenStack deploying itself, and proprietary installations. This will be done with resort to practical test deployments on virtualised environments, followed by some functional testing and performance evaluation.

Chapter 4

DI-FCT-NOVA Cloud Architecture

4.1 Requirements

In Chapter 1, we set forth a few arguments in favour of the use of a private cloud infrastructure to support highly dynamic environments: resources can be rapidly provisioned and released in self service portals, with minimal interaction with the infrastructure administrators.

Now, we discuss the requirements for our specific case: a computer science department within a university where students, teachers and researchers will be cloud consumers. Next, we propose solutions we found appropriate while satisfying two constraints: a) they must be compatible and, if possible, endorsed by the OpenStack developers/community; and, b) they must be lightweight, from the point of view of "system administration" tasks.

4.1.1 Authentication and User Management

An important requirement was to keep user management and authentication as simple as possible, to reduce managerial and record-keeping burdens. We evaluated both OAuth 2.0 [36] and OpenID Connect [37], both supported by OpenStack; however, these systems do not currently support whitelisting in a streamlined fashion, leading to an implementation burden that does not fulfill the lightweight requirement. Next, we looked at Keystone, which also supports SAML 2.0 [8], but this option was not pursued due to insufficient permissions granted by the faculty's Google Services accounts – what was available was not enough to set up that system. However, it is an option to explore in future cloud deployments.

In the end, we decided to implement "Authentication and User Management" via

LDAP [53], using external tools for the record-keeping of accounts and groups, but integrating LDAP with Keystone, OpenStack's module for authentication and authorisation. This allows for future integration with other systems that interact with LDAP backends. A further benefit of LDAP is that it allows the creation of ephemeral accounts with limited lifetimes, and self-renewing accounts which become locked after a period of inactivity.

4.1.2 VM template repository

Virtual Machine templates, or images, are managed through OpenStack's Glance. Cloud administrators create new images and upload them via OpenStack's dashboard or CLI; Glance, resorting to a storage provider such as Ceph to store those images, allows them to be retrieved by users when they wish to create new VM instances to run on Nova compute clusters.

4.1.3 Node Management

The process of adding/removing new nodes to the cloud should be simple, in line with our requirement of low maintenance costs. New servers may be acquired, or become available from other research projects, and the process of adding those servers to the cloud should be seamless and quick. On the other hand, sometimes servers may be reassigned to other projects or obsoleted, in which case removing nodes from the cloud should also be an easy process.

Satisfying this requirement is dependent on the deployment methodology that will be used, and therefore will be further discussed in Chapter 5. However, given that out of the three possible uses for servers in our cloud, one is virtual, only two possibilities remain for integrating new servers into the cloud infrastructure: either as part of the compute cluster, to add more resources to the compute pool for virtual machines, or as part of the storage cluster, to add more disk space, bandwidth, or resilience.

4.1.4 Private tenant networks

Isolation is an extremely important requisite, which is fully supported by OpenStack's networking module, Neutron. One of the most widely used methods to ensure tenant privacy in OpenStack deployments is VXLAN tunnels — this is our first choice when implementing private links between virtual machines. However, other methods are made available to the administrators, such as plain VLANs [1], GRE [15], or even Geneve [21] (however, its preliminary status may make it inadequate for production environments).

4.2 Cloud Components

While OpenStack may be deployed as a monolithic architecture on a small computer (even a laptop) for testing purposes, any production-grade deployment should feature a more distributed architecture with roles assigned to specific hardware according to what services they provide to the cloud.

As such, in the case of DI-FCT-NOVA's private Cloud, three distinct hardware categories were identified: storage cluster, compute cluster, and management nodes. Management nodes will host all services that constitute the OpenStack private cloud, as well as any ancillary software needed to support these services.

4.2.1 Storage Cluster

While the OpenStack project has developed its own storage software, it is widely known that other projects, such as Ceph are better suited for this task. Since hardware resource requirements of storage nodes is not very high, older hardware has been repurposed as hosts for the storage cluster, which will initially be composed of three Dell PowerEdge SC1435 [26] rack-mountable servers equipped with an adequate number of storage drives. This cluster will then host a Ceph deployment consisting of an Object Storage Daemon (OSD) per physical storage volume, and at least three Monitor (MON) instances in order to achieve a good degree of fault-tolerance.

The purpose of this Ceph deployment is to serve as a storage backend for OpenStack's Image Service (Glance), as well as its Block Storage Service (Cinder), which will store virtual machine images and storage volumes, respectively, as well as any other future OpenStack services that may need storage access.

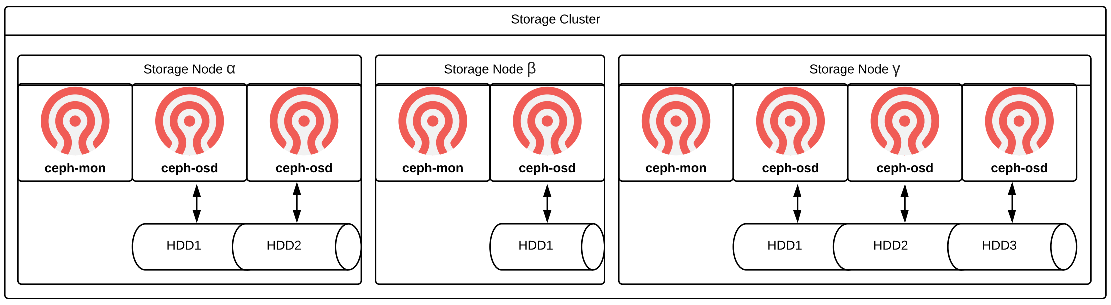

Figure 4.1: Example Storage Cluster with three nodes and heterogeneous volume numbers

4.2.2 Compute Cluster

The computing section of the private Cloud will be handled by five nodes, fully dedicated to hosting OpenStack's Compute Service (Nova), and configured to use the KVM hypervisor, the default option for Nova deployments.

Dedicating the most recent and powerful hardware to the computing cluster means that users will have more resources available in general, and that the deployment will be made to a homogeneous cluster, which may be beneficial when diagnosing possible misconfiguration issues.

4.2.3 Management Nodes

We have decided to host OpenStack's core services, from the relational database software to the Networking Service (Neutron), as virtual machines on an existing ESXi-based infrastructure. This brings in several advantages: it is very easy to dynamically scale resources according to the future evolution of the cloud infrastructure; it is possible, through the use of snapshots, to perform fast backups, safe upgrades (and rollbacks in case of problems), and easily restore the management nodes (VMs) in case of hardware failure.

The services to be provisioned on management nodes are Keystone (authentication/authorisation), Horizon (dashboard), Neutron (networking), Cinder (block storage), Glance (image/template storage), and Nova's (compute service) Controller, as well as the Ceph API gateway. In addition, other services needed by OpenStack, such as a relational database, a message brokering system, RabbitMQ [46], and an LDAP that will serve as a backend for Keystone's user data, are also virtualised and hosted on a management node.

4.3 Network Connectivity

Interlinking the three components — storage, compute and management, a network architecture of some breadth and complexity needed to be planned. To maintain some degree of compatibility with the existing department (and therefore the faculty) network infrastructure, which is based on VLANs [1], these were used separate traffic within the OpenStack cloud, where we identified five categories with regard to traffic segregation: Ceph networks, which carry storage traffic, both data and management; OpenStack management networks, the purpose of which is to separate management traffic from user traffic and to protect management APIs; OpenStack internal networks, for OpenStack's intra-service traffic; and external networks, for user-access and north/southbound VM traffic. Finally, it is important to mention the out-of-band network, which allows management access to the physical infrastructure via IPMI [12].

4.3.1 Ceph Networks

Ceph networks carry traffic from the Ceph storage cluster. The official Ceph documentation [50] recommends two networks for separation of concerns, naming them public, and cluster. In Figure 4.2, the traffic flow and access patterns of the two core Ceph services may be seen.

The cluster network will be used for synchronisation and replication of data and will be unavailable outside the Ceph cluster. The public network will be used for data movement between Ceph and its clients, i.e., external services that access the storage cluster; client examples include OpenStack's own Cinder and Glance, as well as Nova's virtual machines.

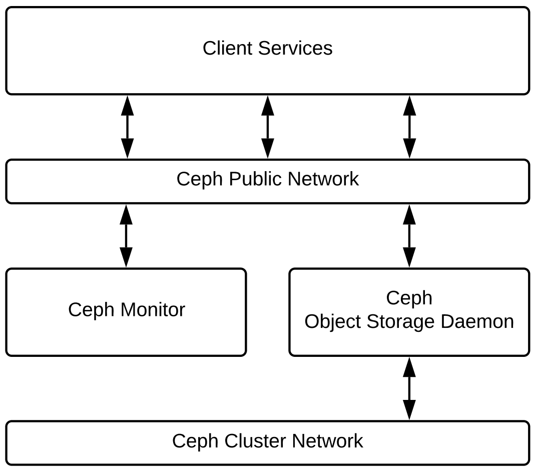

Figure 4.2: Ceph Networking

4.3.2 Administration Networks

Administrators need two networks, to fulfil two different duties: an OpenStack dedicated admin Network, where administerial APIs for its services reside, and a management network to support direct access to nodes, such as via SSH. Although exposing these networks to outside traffic does not outright give unauthenticated entities elevated access to the private cloud, they should nevertheless be protected from access by anyone that is not entitled to manage and administer the cloud: that should greatly reduce the available attack surface.

The previously mentioned out-of-band network should also be considered a part of the administration network set, though it is important to underline the importance that it remains absolutely inaccessible to anyone but trusted administrators, due to known security vulnerabilities in IPMI [35, 59] which would leave the physical machines vulnerable to unauthorised access. In essence, granting access to the out-of-band network should be seen as the equivalent of granting complete access to the whole cloud.

4.3.3 Internal Networks

OpenStack requires two networks for its exclusive use: an internal network, where OpenStack services may communicate with each other; and a Nova network for VM traffic, both east-west and north-south. Both networks should be completely isolated from the outside by design; the routing of north-south traffic from and to the Nova network will be carried out by the Neutron service in the management nodes.

4.3.4 External Networks

There is a single user-accessible public network that allows access to OpenStack services such as the dashboard (Horizon) or the APIs. In addition, there should be at least one network that will provide external access (north-south) to the private cloud's virtual machines through floating IP addresses that will be assigned (either by users or via DHCP) to those VM instances that should be externally accessible.

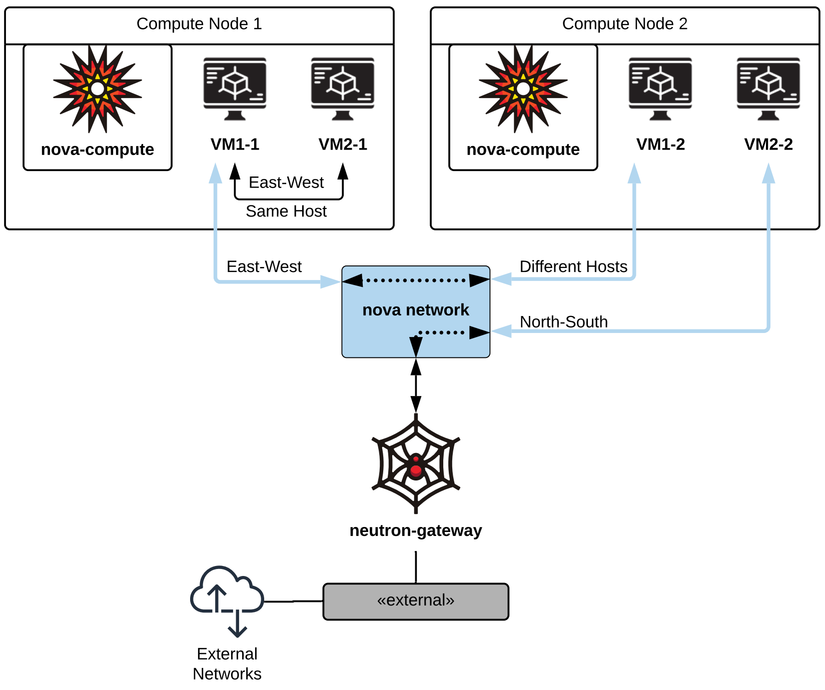

Figure 4.3: Virtual Machine Routing

Figure 4.3 shows possible traffic flows for virtual machines that are connected to the same virtual network, both on the same host and on different hosts (east-west traffic), as well as how these machines may connect to or be accessed from an external access network (north-south traffic).

4.4 Global View

With all the information presented thus far, it is possible to draw an abstract global view of the architecture, which is presented as Figure 4.4.

We see a management node, running the previously mentioned services, as well as a storage cluster and a compute cluster, each represented by individual nodes, to better illustrate the networking requirements of each node.

In this diagram, lower-level details such as deployment services were purposefully excluded, since they will be elaborated and built upon in the next Chapter, where a more complete view of the full deployment process may be seen.

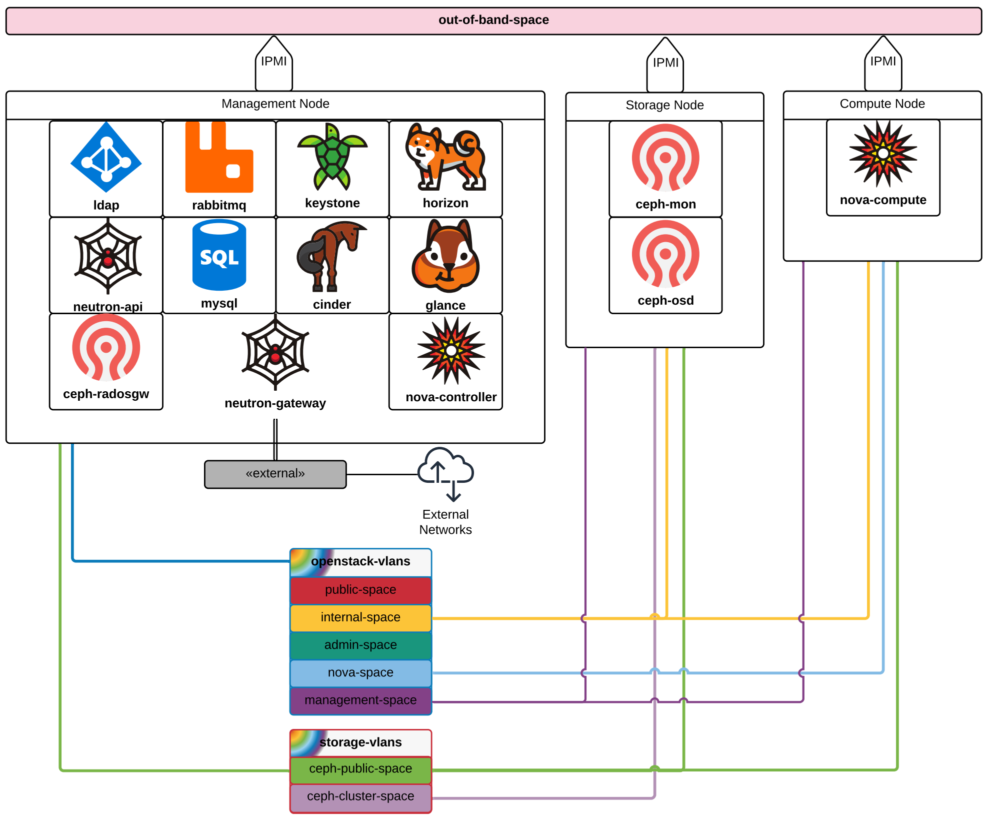

Figure 4.4: Cloud Architecture

CHAPTER 5

Implementation of the Private Cloud

In the previous Chapter, a high-level view of the private cloud was presented, adapted to the unique needs and constraints presented by the users and administrators of the future DI-FCT-NOVA cloud. This Chapter seeks to build on that work by augmenting it with the specific deployment methodologies and configuration parameters to achieve a functional and maintainable private cloud.

As such, this Chapter will cover the deployment tools used, along with their purpose and functionality.

5.1 Deployment Tools

While deploying OpenStack manually is possible, it would entail setting up each node with a suitable Linux distribution, installing baseline software and performing its subsequent configuration, as well as deploying each OpenStack service manually, along with the maintenance load of keeping all the configuration details and logging data accessible. As a result of that, OpenStack has a few deployment toolsets available, which not only provision the physical nodes, but also facilitate maintenance work by providing a centralised location to access and configure the Cloud deployment.

In the course of the work developed on this thesis, Canonical's deployment stack was the simplest to work with, requiring less time investment in learning peripheral technologies, and sufficiently open and simple for small adjustments to be done to the tooling, if necessary.

This section seeks to provide details on the strengths behind the tools used, along with their possible weaknesses, pitfalls, or points of improvement.

5.1.1 MAAS — Metal as a Service

Canonical's MAAS [6], which stands for Metal-as-a-Service, allows users to commission and deploy physical machines in a similar fashion to virtual machines in a cloud, either via its Web UI, or a more complete REST API, with a command-line client available.

MAAS' strengths lie in its capability to categorise, configure, boot up, and install a pre-selected OS image on a physical machine in accordance to the requirements given, whether they be RAM capacity, disk storage, or a custom categorised machine, via MAAS' tag-based system. It further provides options for IP management, including IPv6 support, DHCP, DNS, and NTP services for the entire physical infrastructure. In addition to its capabilities with physical machines, MAAS also has the capability of integrating with existing VMWare-based infrastructure to provision virtual machines as physical machines, treating them as first-class citizens, having the same feature set as when provisioning a real, physical machine. A very important feature of MAAS is that it allows for the definition of network spaces, which may contain subnets and VLAN tags, and allows for any machine to be commissioned within these network spaces.

Since these systems need to be reliable, high availability (HA) mechanisms are also available, though they were not used in the context of this thesis. These mechanisms rely on a division of MAAS servers between two distinct roles: region controllers (regiond), and rack controllers (rackd). Region controllers administer rack controllers, and rack controllers administer physical machine racks, which may (or may not) correspond to physical server racks. In a deployment of MAAS without HA, a single instance is both a rack controller, as well as its own region controller.

The region controller manages DNS, NTP, and the API (coupled with the web UI), whereas the rack controllers manage DHCP and the TFTP service for booting the physical machines it administers via PXE [56].

This system has proven to be very adequate at managing IPMI [12]-capable machines during test scenarios, as well as providing a good inventory and UI to perform tests from. Although it is meant to be used in conjunction with orchestration software (such as Puppet [48] and Juju [27]), it is very useful on its own, and for small, uncomplicated physical hardware provisioning, without the need for tenant isolation, might be a good solution.

5.1.2 Juju

Canonical's Juju [27] is an orchestration and modelling software meant to assist administrators in the task of defining applications and their configuration as well as how they're meant to run, and where. Its main target is cloud infrastructure, whether public clouds, such as Amazon's AWS, Google's Cloud Engine, or Microsoft's Azure, or private clouds such as those running on MAAS, OpenStack, or vSphere.

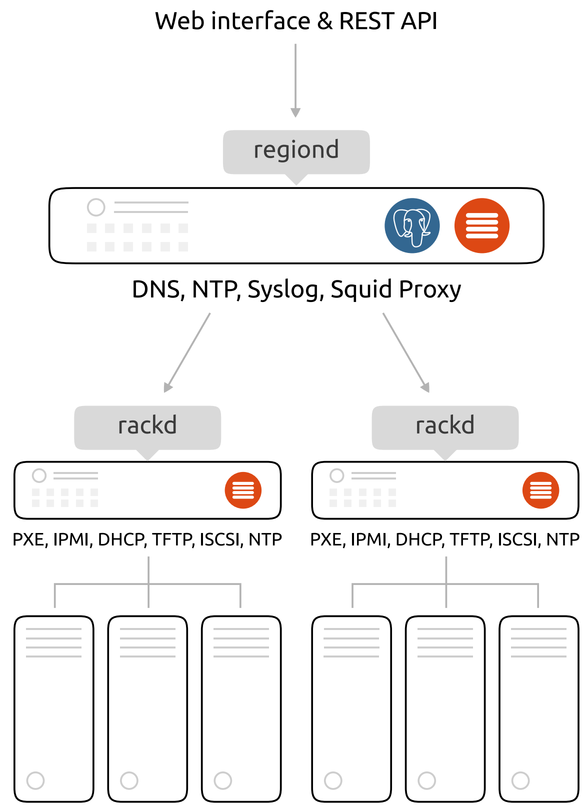
Web interface & REST API
regiond
DNS, NTP, Syslog, Squid Proxy
rackd
rackd
PXE, IPMI, DHCP, TFTP, ISCSI, NTP
PXE, IPMI, DHCP, TFTP, ISCSI, NTP

Figure 5.1: MAAS Architecture [5]

Juju works via Charms, which are collections of scripts containing the necessary instructions to deploy, configure, scale, maintain, upgrade, and monitor applications. Usually, each application has one charm that deploys it, and this charm may have relations with other applications, deployed by different charms. In this way, it is possible to deploy complex bundles of applications, such as OpenStack or Hadoop, where each individual application is one charm, having relations to other charms, in order to deploy a functional instance.

Both charms and bundles of charms may be configured via a YAML-based configuration file, which defines parameters such as what interface or network to bind the application to, or other options made available by the charm creators. An example of a bundle configuration file may be seen in Figure 5.2, which would deploy a MediaWiki [19] instance, with a MySQL backend. This is done on an Ubuntu Trusty distribution, on a machine with a minimum of 4 CPU cores and 8 GB of RAM, where each service (MediaWiki and MySQL) are deployed on separate containers. The relations section indicates that mediawiki requires an association to a database (db) type service, and that mysql fulfils the database requirement.

In order to function, Juju needs to be associated with a machine provider, MAAS in our case, and undergo a process called bootstrapping. This involves commissioning a machine from the provider and installing Juju's Controller, a service dedicated to managing Juju-commissioned machines, responding to triggered events, and seamless configuration between same services running on different machines. Once the bootstrapping process is complete, Juju is ready to deploy charms or bundles with any necessary services.

```
series: trusty
machines:
  '0':
    constraints: cores=4 mem=8G
    series: trusty
services:
  mediawiki:
    charm: cs:trusty/mediawiki-3
    num_units: 1
    options:
      debug: false
      name: ExampleWiki
    to:
      - lxd:0
  mysql:
    charm: cs:trusty/mysql-29
    num_units: 1
    options:
      flavor: distro
      ha-bindiface: eth0
      ha-mcastport: 5411
      vip_cidr: 24
      vip_iface: eth0
    to:
      - lxd:0
relations:
- - mediawiki:db
  - mysql:db
```

Figure 5.2: Simplified configuration for a MediaWiki deployment

5.1.3 Architecture Impact

In the previous Chapter, a generic architecture was shown without considerations for deployment tooling; in order to fulfil that plan with the Canonical suite of deployment tools, two additional services need to be added to the management nodes: MAAS itself and Juju's Controller. This Deployment Node shall be represented in a separate node to the existing Management Node due to clarity, but will also be implemented within the ESXi infrastructure, and is thus part of the Management Nodes. Figure 5.3 shows the complete view of the architecture, with this additional node added.

5.2 Deployment Process

The deployment process using the previously described tools is a straightforward process, once an architecture is planned out, and may be divided into baremetal provisioning, and then service deployment.

5.2.1 Baremetal Provisioning

The first steps towards a successful deployment and to create a MAAS installation on a management node, and then registering both the ESXi infrastructure for management nodes and the physical machines that will be part of our OpenStack cloud deployment into the MAAS database.

After registry, the device is usually booted via PXE with an image by MAAS that will perform introspection in order to ascertain the hardware capabilities, upon which

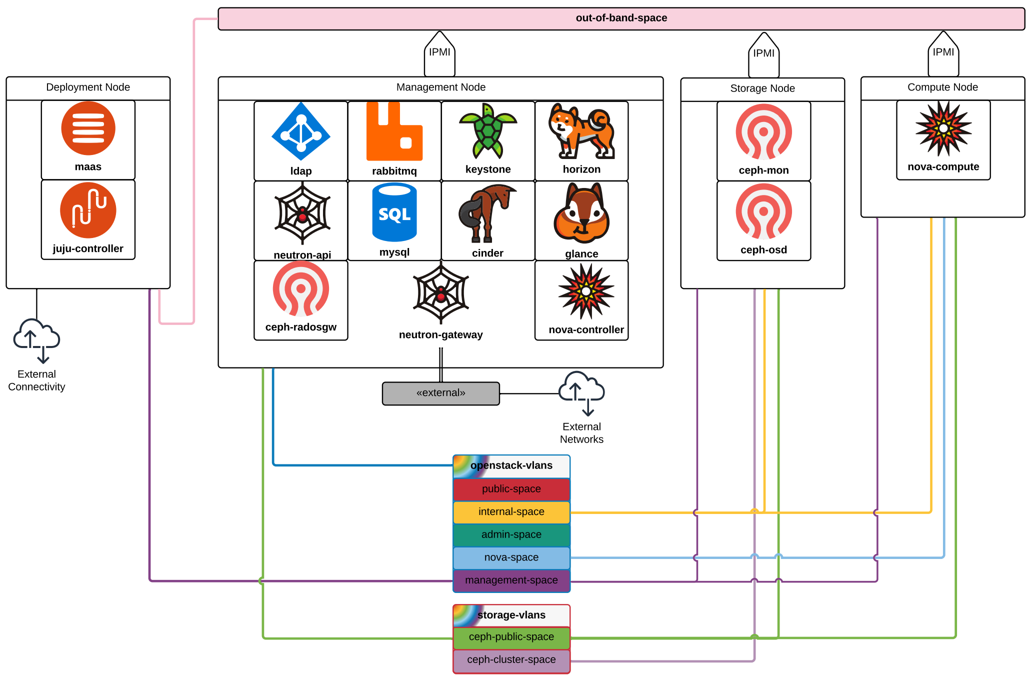

Figure 5.3: OpenStack Deployment Architecture

the machine will be available for commissioning. Machines should be tagged with their purpose as well, as an example, machines that will be used in the compute cluster should have the 'compute' tag.

In parallel, the different network configurations mentioned in the previous Chapter may be configured into MAAS. This includes assignment of VLAN IDs for the separate networks, as well as subnet ranges and nomenclature concerns. The VLANs that are of exclusive OpenStack use may be assigned any subnet range, since it will not interfere with any existing systems, but the external-facing networks will need coordination with datacentre personnel in order to prevent clashes in configuration.

At the end of these steps it is expected that the MAAS system may commission any of our machine resources into their possible network placements without any errors.

5.2.2 Service Deployment

After the MAAS configuration is complete, an installation of the Juju client should be performed on a client machine with access to the MAAS node. The MAAS node itself may also be used as a Juju client.

The Juju client will need to be configured with the MAAS access credentials, and then the Juju controller will need to be deployed via the `juju bootstrap` command. This will allocate a machine registered on MAAS, which should be a virtual machine inside the ESXi infrastructure tagged for the purpose, belonging to our management node set.

After this, the infrastructure is ready for the private cloud deployment, which may be done via the `juju deploy bundle.yaml` command, where bundle.yaml is the aforementioned YAML-based configuration file for our constituent services. During the several test deployments, the time between executing the deploy command and having a fully deployed cloud was close to one hour, with little deviation.

5.3 Post-Deployment

After deployment, the private cloud is not yet ready to be used, needing some configuration steps by the administration, namely creating virtual machine flavours, uploading at least one image to boot virtual machine instances with, and adding at least an external network to provide north-southbound traffic routing to tenant networks. Tenants will need to add one or more tenant networks so instances can be connected to, and may be routed to external networks.

5.3.1 Virtual Machine Flavours

A flavour of virtual machine defines what resources will be made available to a virtual machine instance, such as RAM, number of vCPUs and storage capacity. Since there are no strict requirements for these categories, configuring a set of generic capabilities initially, and allowing administrators to extend these with new categories at a later date, if necessary. The initial flavours are laid out on Table 5.1, and are drawn from pre-Mitaka versions of OpenStack, which came with these flavours configured by default [18].

Flavour	*vCPUs*	*Storage (GB)*	*RAM (MB)*
Tiny	1	1	512
Small	1	20	2048
Medium	2	40	4096
Large	4	80	8192
XLarge	8	160	16384

Table 5.1: OpenStack's Default Virtual Machine Flavours

5.3.2 Virtual Machine Images

In order to boot a virtual machine instance into an useful state, an image containing an operating system needs to be uploaded to the private cloud so that it may be used as a base for that virtual machine. In some cases, Linux distributions make these available in public repositories: CentOS [9], CirrOS [11], Debian [13], Fedora [16], Ubuntu [58], and OpenSUSE [43]. There are also evaluation versions of Microsoft Windows made available by CloudBase [63].

These images may also be custom-made using appropriate tooling, such as diskimage-builder [17], Oz [45] and virt-builder [60], but also OpenStack-ready tools like image-bootstrap [24], and the newer imagefactory [25]. There is also specific tooling for creating custom Microsoft Windows cloud images by CloudBase [42].

One or more images will need to be uploaded to OpenStack, which may be done via the dashboard or the command-line via the `openstack image create` command.

5.3.3 Network Creation

As a final step, as many provider networks as need to be made available to the users should be configured. At least one is necessary if inbound and outbound access for virtual machines is needed, but as many as desired can be set up and in use.

This configuration may be done either on the dashboard or via the command-line interface, through the `openstack network create -external` command, and then the `openstack subnet create` to configure a subnetwork for the external network.

The other networks to be created are the remit of tenants, and not the administration, and are created in a similar manner to the provider networks.

5.4 Authorisation and Authentication

OpenStack uses a policy-based authorisation system, relying on `policy.json` files for each project the user may interact with either directly or via APIs [40]. These files lay out rules for acquiring each policy on a per-policy basis, and policies range between purely informative tasks, such as listing users or domains, to more wide-ranging, such as altering provider networks and wholesale removal of domains.

In this deployment, the defaults were used, which adequately fulfil the necessary requirements. Since a production-grade OpenStack deployment has many interacting projects, and documentation on their myriad policies and interrelations is scarce, altering these policy files, even minimally, should not be a task that is undertaken lightly, since a small mistake may cascade into denying users access rights they should have, or less benignly, allow users to obtain features they should not have access to.

On the Authentication front, three types of accounts were required: permanent, self-renewing, and ephemeral. Permanent accounts should belong to administrators and/or full-time personnel that need access to the private cloud; self-renewing accounts should belong to higher-level personnel that, although their access to the cloud should be temporary, are capable of self-limiting their usage upon conclusion of their usage period, allowing their accounts to expire without renewal; and finally, ephemeral accounts would be limited to students and other very limited-term personnel who need access to the private cloud on a very temporary basis, with a hard temporal limit on that usage.

5.5 User Stories

OpenStack is an incredibly complex and customisable system and, as such, the process of condensing down that same complexity into a simplified document without losing sight of the whole is a non-trivial process. Because of this, some user stories shall be laid out as examples of possible uses of the DI-FCT-NOVA private cloud, and in the process, the hope is that a tractable slice of the OpenStack functionality is adequately given.

5.5.1 Context

OpenStack uses its own terminology which has some connection to the generic terminology introduced in Chapter 1: Tenants are referred to as Projects, which contain Users. Both Projects and Users are organised under Domains. In this way, several Users may work under the same Project, and we may enforce isolation between Users and their Projects using Domains. A User's relationship with a Project will always be associated with a Role, which defines the access scope of the User. Figure 5.4 presents an example of how these concepts fit together, and will be built upon further in the user stories.

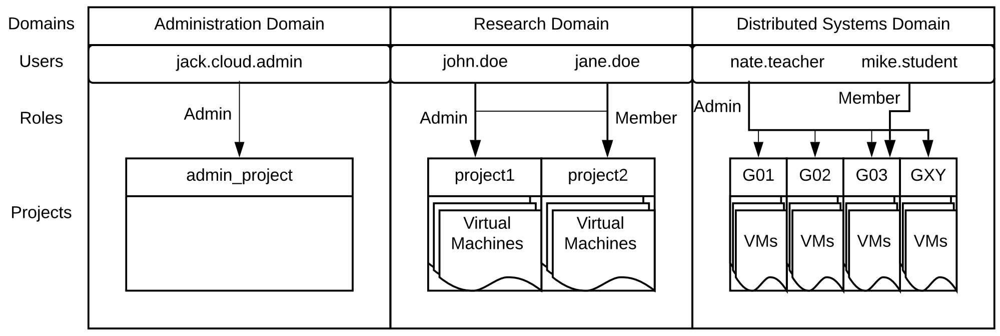

Figure 5.4: OpenStack Identity Concepts

By default, an OpenStack cloud defines a Cloud Administrator role, with full access to the whole cloud and the capability of effecting configuration changes at the deepest levels within multiple domains. This user, much like the `root` Linux user, should be used with care, and just for administrative duties that cannot be performed with smaller-scoped accounts.

The way OpenStack implements the Cloud and Domain Administrator roles is not via a standard role, but by having special Domains and Projects that define the users therein as having special privileges. In the case of the Domain Administrator, a special Administrator Project is defined within each Domain, and any users with the Admin role access to this project are considered Domain Administrators by OpenStack, which entitles them to manage users, projects and resources scoped within their Domain. The Cloud Administrator also has an Administrator Project, but contained within a special

Administrator Domain, which entitles him to the Domain Administrator role over all Domains in the private cloud.

5.5.2 Research Domain

The Research Domain story starts with a necessity on the part of researchers at DI-FCT-NOVA to use compute resources in the context of their research projects, while being isolated from other unaffiliated projects and researchers.

Given this context, a Research Domain would be created by the Cloud Administrator, who would perform the configuration of that domain to be backed by LDAP. Within this domain, accounts for each individual researcher to be given access to the private cloud would be added. This Research Domain should have one (or more) Domain Administrators, who will be capable of managing projects and administering user access controls.

Once the creation work is done, a Domain Administrator should be capable of creating Projects for each researcher that needs an isolated workspace, defining its maximum resource allocation pool, and giving the Admin role to research users responsible for that project.

Ultimately, it should be the remit of the Project Administrator to administer the resources of their project responsibly, and make sure the users in his project are well-behaved, only resorting to the Domain Administrator in cases where his access level or know-how is insufficient to rectify possible eventualities.

5.5.3 Class Domains

Class Domains seek to facilitate the roles of teachers and students of DI-FCT-NOVA, by providing student groups with isolated compute resources to perform their class projects in, at the behest and control of their respective teachers.

When a lecturer assigned to a class needs to allocate resources for his students within the private cloud, he would ask the Cloud Administrator to create a Class Domain (in Figure 5.4, the Distributed Systems class is used as an example, with G01 to GXY as workgroups), which would be backed by LDAP, having the accounts of all students signed on to that class.

The lecturer or lecturers responsible for this class would gain the role of Domain Administrator within their class domain, permitting them to create new projects for each student group, and assign student accounts to those projects, limiting their resource allocation pool according to what will be necessary in each project.

The final objective is that students will have full access to an isolated environment to create and experiment with virtualised computing resources, facilitating learning without needing dedicated resources. This environment will be administered by their lecturers, with whom they will have a direct link to, both in case of needing assistance with the system as well as in cases where edge cases appear and may be resolved by the lecturers.

5.6 Cloud Validation

The validation of a private cloud for the purposes of DI-FCT-NOVA beyond the basics on whether its most basic features work, such as creation of virtual machines and networks, is an involved process that requires multiple users interacting with the system and reporting any problems encountered. Unfortunately, there was no time to perform that level of scrutiny on the private cloud due to time constraints. Attempting to perform the previously mentioned use cases in the private cloud would be a good start, since they encompass the two major categories of usage, and should test all the necessary functionality of the private cloud.

Chapter 6

Conclusion

During the course of this work, it was possible to evaluate at least three private cloud frameworks, and realise that within the private cloud framework space, OpenStack is definitely a giant, not just in usage and features, but also investment, volume of contributions, and overall third-party documentation. However, these advantages do not come without a dark side: outdated or missing documentation for some systems, highly fragmented deployment base, with new versions every six months potentially having a heavy contribution to both the fragmentation and the documentation issues, since there is little time to settle within that time period. These issues mean that using an established commercial solution/distribution of OpenStack, with its associated support lines, might be worth the cost in maintenance and support, even for institutions with limited funding opportunities.

That said, the advantages of having a functioning private cloud at DI-FCT-NOVA cannot be understated: with its separation between Cloud, Domain, and Project administration, work can be efficiently distributed between researchers that have the capabilities and know-how to administer the system, without giving them too much access to the underlying systems involved. These advantages do not manifest just in the research arena, however, since the isolation capabilities of a private cloud allow less trusted users, such as students, to have their own allocated space within this private cloud in order to further augment their studies, without having a perceivable impact on important work running in parallel, or even on other students' workflows. Within the context of cloud computing classes, especially, having a functional private cloud backed by OpenStack can be a curriculum enhancement for students wishing to pursue a career in cloud computing, even if that career does not directly involve OpenStack.

At a personal level, although the work developed during the course of this thesis was at times highly frustrating and fraught with delays caused by difficult to debug problems, often due to the interactions between virtualisation, networking, and systems administration, it was also incredibly rewarding, allowing the acquisition of knowledge and experience with enterprise-grade systems administration, cloud technology, network planning, among many others. Some of these areas are often difficult to acquire practical experience in without the acquisition of expensive hardware or a job in the field, and were thus highly gratifying and very useful in a future professional setting. Furthermore, frequent meetings and close coordination with both advisers allowed for a fruitful environment that is wholly different in both personal responsibility as well as independence from a classroom environment, which provides a much needed bridge between student life and a professional career.

Unfortunately, some objectives laid out were unaccomplished, such as integrating Keystone with Google, either via OAuth or SAML, using a whitelist-based access, which is unsupported by Keystone; testing OpenStack's Quinling project, which would potentially allow students to submit their classwork and receive the output of their work without having to interact with a server directly, in an isolated and secure fashion, and exploring OpenStack's Ironic, allowing for the provisioning of whole physical machines for projects that should not be virtualised, although a solution outside OpenStack, MAAS, could be adapted to serve a similar purpose.

6.1 Future Work

Beyond a functional private cloud deployment, and following from the mentioned unaccomplished goals, keeping an eye on the viability of implementing a whitelist-based Google-integrated Keystone in the future, either by requesting that capability via official channels, or contributing to its development and future integration in the private cloud would allow for the discontinuation of locally-based LDAP servers, which would lead to less maintenance requirements on administration.

An OpenStack version is meant to be under active maintenace for approximately 18 months, so a future upgrade should be a necessity. With the way the deployment was planned, and using Juju, upgrades should not be extremely complex, especially given that there is no service-level agreement to uphold, but it is still a process that needs to be closely monitored for possible faults.

In addition, OpenStack contains many projects that, while potentially interesting for the DI-FCT-NOVA private cloud, were not evaluated or researched in-depth, either due to lack of time, particular complexity, or due to their recent release and possible instability. Further investigation of these projects, eyeing a future upgrade of the private cloud with one or more of them might be desirable, depending on the features necessary.

At a surface-level, we may divide these modules into three categories: high-availability and disaster recovery; baremetal and high performance computing; and containerisation.

Within the realm of high-availability and disaster recovery, due to the management part of the private cloud running on an ESXi infrastructure, and high-availability not being a primary concern, these modules weren't analysed. However, the Freezer module, which allows for point-in-time snapshots, differential and incremental backup policies of files or filesystems, among others, both for virtual machines running inside the private cloud, but also for external devices running outside the cloud, might be a worthwhile addition. Another module in this category is Masakari, which allows for automatic recovery of failed virtual machine instances; from errors within the Nova instance itself, all the way to full Nova node failure.

The baremetal and high-performance computing fields have two modules that are worth analysing: Ironic and Cyborg. Ironic, as mentioned before, allows for bare metal provisioning, and was an unaccomplished secondary objective of this private cloud development, whereas Cyborg permits usage of accelerator devices meant for high-performance computing, such as FPGAs, GPUs, NVMe SSDs, among others, within virtual machine instances. The usage of these modules may be analysed and implemented in the future, with the caveat that, at a surface level, they both seem fairly complex to deploy successfully.

As for containerisation, there is both Zun and Magnum, serving different use-cases. Zun treats containers as first-class resources of OpenStack, allowing users to instantiate containers in the same manner as virtual machines, additionally requiring project Kuryr to integrate containers' networks with Neutron. Magnum, on the other hand, is a lower-level project, seeking to help users implement their own Container Orchestration Engine (COE), such as Kubernetes, Docker, etc., allowing the user to manage those engines at a much lower-level of granularity, with the least amount of integration with OpenStack. While Zun's usefulness is apparent, bringing all the advantages of containerisation to the private cloud, Magnum would only be useful for cases where users absolutely need to have control over all aspects of a COE.

Two extra projects that are worth mentioning but do not belong to a definite category are Quinling, which has been previously mentioned as the Functions as a Service project in OpenStack, and is potentially useful if it can be configured to perform the duty of allowing students to execute coursework payloads in an isolated manner, without direct access to a virtual machine; and Blazar, which is a resource reservation service, allowing users to reserve virtual machine instances, storage volumes, and even whole hosts for a single purpose. Blazar may indeed be useful especially given the private cloud's future growth, both in users and hardware, since it may be useful for researchers to reserve whole physical nodes so that their virtual machine instances have complete physical isolation from all other instances running on the private cloud.

Bibliography

[1] *802.1Q IEEE Standards for Local and metropolitan area networks Virtual Bridged Local Area Networks*. LAN/MAN Standards Committee of the IEEE Computer Society. 2003.

[2] *Ansible*. URL: https://www.ansible.com/.

[3] Apache. *Apache CloudStack*. URL: https://cloudstack.apache.org/.

[4] M. Bakke, J. Hafner, J. Hufferd, K. Voruganti, and M. Krueger. *Internet Small Computer Systems Interface (iSCSI) Naming and Discovery*. RFC 3721. RFC Editor, Apr. 2004.

[5] Canonical. *MAAS: How it Works*. URL: https://maas.io/how-it-works.

[6] *Canonical MAAS*. URL: https://maas.io/.

[7] *Canonical OpenStack*. URL: https://www.ubuntu.com/openstack/install.

[8] J. P.R.M.E.O.I.N.R.S.S. M. Cantor S.; Kemp. *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*. 2004. URL: http://xml.coverpages.org/SAML-core-20-CD-01.pdf.

[9] *CentOS Cloud Images*. URL: http://cloud.centos.org/.

[10] *Ceph*. URL: https://ceph.com/.

[11] *CirrOS Cloud Images*. URL: http://download.cirros-cloud.net/.

[12] I. Corporation. *Intelligent Platform Management Interface*. URL: https://www.intel.com/content/www/us/en/servers/ipmi/ipmi-home.html.

[13] *Debian OpenStack Images*. URL: http://cdimage.debian.org/cdimage/openstack/.

[14] Docker. *Docker*. URL: https://www.docker.com/.

[15] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. *Generic Routing Encapsulation (GRE)*. RFC 2784. http://www.rfc-editor.org/rfc/rfc2784.txt. RFC Editor, Mar. 2000. URL: http://www.rfc-editor.org/rfc/rfc2784.txt.

[16] *Fedora Cloud Images*. URL: https://alt.fedoraproject.org/cloud/.

[17] O. Foundation. *OpenStack Diskimage-Builder*. URL: https://docs.openstack.org/diskimage-builder/latest/.

[18] O. Foundation. *OpenStack Flavours*. URL: https://docs.openstack.org/nova/stein/admin/flavors.html.

[19] W. Foundation. *MediaWiki*. URL: https://www.mediawiki.org/wiki/MediaWiki.

[20] *Gluster*. URL: https://docs.gluster.org/en/latest/.

[21] J. Gross, I. Ganga, and T. Sridhar. *Geneve: Generic Network Virtualization Encapsulation*. Internet-Draft draft-ietf-nvo3-geneve-13. Work in Progress. Internet Engineering Task Force, Mar. 2019. 35 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-nvo3-geneve-13.

[22] R. Hat. *File storage, block storage, or object storage?* URL: https://www.redhat.com/en/topics/data-storage/file-block-object-storage.

[23] *Hyper-V Architecture*. URL: https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture.

[24] *Image-bootstrap*. URL: https://github.com/hartwork/image-bootstrap.

[25] *Imagefactory*. URL: http://imgfac.org/.

[26] D. Inc. *Dell PowerEdge SC1435*. URL: https://www.dell.com/downloads/emea/products/pedge/en/SC1435_Spec_Sheet_Quad.pdf.

[27] *Juju Charms*. URL: https://jaas.ai/.

[28] *Kernel-based Virtual Machine*. URL: https://www.linux-kvm.org/page/Main_Page.

[29] *Linux Containers*. URL: https://linuxcontainers.org/.

[30] *Logical Volume Manager*. URL: https://www.sourceware.org/lvm2/.

[31] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. RFC 7348. http://www.rfc-editor.org/rfc/rfc7348.txt. RFC Editor, Aug. 2014. URL: http://www.rfc-editor.org/rfc/rfc7348.txt.

[32] P. Mell and T. Grance. *The NIST Definition of Cloud Computing*. Tech. rep. 800-145. National Institute of Standards and Technology (NIST), 2011. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[33] M. Mesnier, G. Ganger, and E. Riedel. "Storage area networking - Object-based storage." In: *IEEE Communications Magazine* 41.8 (2003), pp. 84–90. DOI: 10.1109/mcom.2003.1222722. URL: https://doi.org/10.1109/mcom.2003.1222722.

[34] R. Moreno-Vozmediano, Montero, and I. M. Llorente. "IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures." In: *Computer* 45.12 (Dec. 2012), pp. 65–72. DOI: 10.1109/mc.2012.76. URL: https://doi.org/10.1109/mc.2012.76.

[35] NIST. *CVE-2013-4786*. URL: https://nvd.nist.gov/vuln/detail/CVE-2013-4786/.

[36] *OAuth 2.0*. URL: https://oauth.net/2/.

[37] *OpenID Connect*. URL: https://openid.net/connect/.

[38] *OpenStack Cinder*. URL: https://docs.openstack.org/cinder/rocky/.

[39] OpenStack Foundation. *OpenStack*. URL: https://www.openstack.org/.

[40] *OpenStack Policy.json*. URL: https://docs.openstack.org/security-guide/identity/policies.html.

[41] *OpenStack Swift*. URL: https://docs.openstack.org/swift/rocky/.

[42] *OpenStack Windows Image Creation*. URL: https://github.com/cloudbase/windows-openstack-imaging-tools/.

[43] *OpenSUSE Cloud Images*. URL: https://software.opensuse.org/distributions/leap#jeos-ports.

[44] *OpenVZ*. URL: https://openvz.org/.

[45] *Oz*. URL: https://github.com/clalancette/oz/wiki.

[46] I. Pivotal Software. *RabbitMQ*. URL: https://www.rabbitmq.com/.

[47] G. J. Popek and R. P. Goldberg. “Formal requirements for virtualizable third generation architectures.” In: *Communications of the ACM* 17.7 (1974), pp. 412–421. DOI: 10.1145/361011.361073. URL: https://doi.org/10.1145/361011.361073.

[48] *Puppet*. URL: https://puppet.com/.

[49] *RDO*. URL: https://www.rdoproject.org/.

[50] I. Red Hat. *Ceph Network Configuration Reference*. URL: https://docs.ceph.com/docs/master/rados/configuration/network-config-ref/.

[51] *Riak S2*. URL: http://basho.com/products/riak-s2/.

[52] *ScaleIO*. URL: https://searchconvergedinfrastructure.techtarget.com/definition/ScaleIO.

[53] J. Sermersheim. *Lightweight Directory Access Protocol (LDAP): The Protocol*. RFC 4511. June 2006. DOI: 10.17487/RFC4511. URL: https://rfc-editor.org/rfc/rfc4511.txt.

[54] Stackalytics. *Stackalytics*. URL: https://www.stackalytics.com/?release=rocky&metric=commits.

[55] O. Systems. *OpenNebula*. URL: https://opennebula.org/.

[56] I. C. SystemSoft. *Preboot Execution Environment (PXE) Specification*. URL: https://web.archive.org/web/20110524083740/http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf.

[57] *The Xen Project*. URL: https://www.xenproject.org/.

[58] *Ubuntu Cloud Images*. URL: http://cloud-images.ubuntu.com/.

[59] U. S.D.o.H. S. US-CERT. *Risks of Using the Intelligent Platform Management Interface (IPMI)*. URL: https://www.us-cert.gov/ncas/alerts/TA13-207A.

[60] *virt-builder*. URL: http://libguestfs.org/virt-builder.1.html.

[61] *VMware ESXi*. URL: https://www.vmware.com/products/esxi-and-esx.html.

[62] S. A. Weil, A. W. Leung, S. A. Brandt, and C. Maltzahn. "RADOS." In: *Proceedings of the 2nd international workshop on Petascale data storage held in conjunction with Supercomputing '07 - PDSW '07*. ACM Press, 2007. DOI: 10.1145/1374596.1374606. URL: https://doi.org/10.1145/1374596.1374606.

[63] *Windows Cloud Images*. URL: https://cloudbase.it/windows-cloud-images/.

[64] *Windows Containers*. URL: https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/.

[65] B. Wire. *OpenStack Launches as Independent Foundation, Begins Work Protecting, Empowering and Promoting OpenStack*. URL: https://www.businesswire.com/news/home/20120919005997/en/OpenStack-Launches-Independent-Foundation-Begins-Work-Protecting.