FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Web System For Creating And Managing Virtual High Performance Computing Environments

Pedro Adriano Pessoa Teixeira

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Jorge Manuel Gomes Barbosa (PhD)

Co-Supervisor: Tito Carlos Soares Vieira (MSc)

June 21, 2012

Web System For Creating And Managing Virtual High Performance Computing Environments

Pedro Adriano Pessoa Teixeira

Mestrado Integrado em Engenharia Informática e Computação

June 21, 2012

Abstract

Current Grid computing infrastructures are generally not very flexible when it comes to the users' needs. As such, whenever it is required, the user must adapt its code to the infrastructures specifications.

On the other hand, Cloud Computing is associated with an extreme flexibility allowing the infrastructure to adapt itself to the users' requirements. Another aspect present in Cloud Computing but non-existent in Grid Computing is the Quality of Service factor, where a user can submit a job according to a certain cost or deadline.

FEUP — Faculty of Engineering of Port University — has started developing a private cloud project at its Informatics center (CICA — Informatics Center Prof. Correia de Araújo) — in which a user can create custom Virtual Machine images on-the-fly and have the system automatically provision the required resources to run the submitted job.

OpenStack and *OpenNebula* are competing cloud management platforms, both with their own methods of dealing with Virtual Machine images.

In this report both cloud management platforms are reviewed in great detail and a choice is made as to which one to use in FEUP's private cloud project. The technologies that support these platforms are also discussed so that the reader can be properly contextualized.

The work and research involved in creating a web system based on *Python* and *Django* that is capable of creating Virtual Machine images according to the users requisites, as well as capable of registering those VM images with a cloud management platform, namely *OpenStack*, is also documented.

Resumo

As infraestruturas actuais de computação em Grelha geralmente não são muito flexíveis no que diz respeito às necessidades dos utilizadores. Assim sendo e sempre que é necessário, é o utilizador que tem de adaptar o código às especificações das infraestruturas.

Por outro lado, a computação em Nuvem é associada a uma flexibilidade extrema, permitindo assim que seja a estrutura a adaptar-se aos requisitos do utilizador. Outro aspecto presente neste tipo de computação, mas que é totalmente ausente na computação em Grelha, é o factor Qualidade de Serviço, em que um utilizador pode submeter um trabalho de acordo com um determinado custo ou um determinado prazo.

A FEUP — Faculdade de Engenharia da Universidade do Porto — começou a desenvolver o seu próprio projecto de nuvem privada no seu centro de informática (CICA — Centro de Informática Prof. Correia de Araújo) em que um utilizador pode criar as suas imagens de máquinas virtuais *on-the-fly* e ser o sistema a provisionar os recursos necessários para correr o trabalho de computação desejado.

OpenStack e *OpenNebula* são duas plataformas de gestão de nuvens, competidoras no mercado, sendo que ambas possuem os seus próprios meios de lidar com imagens de máquinas virtuais.

Neste documento as duas plataformas de gestão de nuvens são revistas e uma delas é escolhida para ser usada no projecto de nuvem privada da FEUP. As tecnologias que servem de suporte a estas plataformas são também abordadas, para que o leitor consiga sentir-se contextualizado.

O trabalho e pesquisa envolvido na criação de um sistema web baseado em *Python* e *Django* que é capaz de criar imagens de máquinas virtuais de acordo com os requisitos do utilizador, bem como capaz de registar essas imagens na plataforma de gestão de nuvens, nomeadamente *OpenStack*, é também documentado.

Acknowledgements

To the people at *CICA*, namely Miguel Costa and Jorge Ruão, for going out of their way to help me.

To Prof. Jorge Barbosa and Prof. Tito Vieira, for all the support, guidance and most of all, patience.

To everyone who shared these last seven years with me, days and nights. I could not have made it without your help and support.

To the members of *NiFEUP*, past and present, for giving me a sense of belonging and helping me when I most needed.

To my family, for the immeasurable support and gargantuan ammounts of patience needed to put up with my antics.

Pedro Teixeira

"It is sometimes an appropriate response to reality to go insane."

Philip K. Dick, *VALIS*

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Abbreviations

ACL	Access Control List
AMI	Amazon Machine Image
API	Application Programming Interface
CICA	Centro de Informática Prof. Correia de Araújo
CLI	Command Line Interface
CPU	Central Processing Unit
DNS	Domain Name Server
FEUP	Faculdade de Engenharia da Universidade do Porto
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IRC	Internet Relay Chat
MIEIC	Mestrado Integrado em Engenharia Informática e Computação (Integrated Master in Informatics and Computer Engineering)
MVC	Model-View-Controller
OS	Operating System
PHP	PHP: Hypertext Processor
QOS	Quality of Service
VD	Virtual Disk
VM	Virtual Machine
VO	Virtual Organization
VW	Virtual Workspace
WWW	*World Wide Web*

Chapter 1

Introduction

High performance computing describes the ability of using parallel processing in order to perform advanced application programs with a great deal of efficiency, reliability and quickness. [Tec]

Current Grid Computing infrastructures are generally not very flexible when it comes to the users' needs. As such, whenever it is required, the user must adapt its code to the infrastructures specifications.

On the other hand, Cloud Computing is associated with an extreme flexibility allowing the infrastructure to adapt itself to the users' requirements. Another aspect present in Cloud Computing but non-existent in Grid Computing is the Quality of Service factor, where a user can submit a job according to a certain cost or deadline.

Furthermore, there is also the elasticity component[1], something that is not available in Grid Computing technologies but is inherent to Cloud Computing, and is one of its flagships that may be able to cross over to grid infrastructures.

In this document is presented a study that analyzes a community project for creating and managing private Cloud infrastructures (along with some of the technologies that supports this project) and how it can be implemented in FEUP - Faculty of Engineering of the University of Porto.

This first chapter introduces a brief technological and situational context, as well as the motivation behind the choice of this subject and the objectives set. The document's structure is also shown.

1.1 Context

Leonard Kleinrock (part of the team that developed Arpanet, an early seed for the Internet) said in 1969:

[1] An application can exapand and contract on demand, across all its tiers (presentation layer, services, database, security and more). Its components can expand independently from each other, without affecting, reconfiguring or changing the other components.

> "As [...] computer networks [...] grow and become sophisticated, we will probably see the spread of 'computer utilities' which, like present electric and telephone utilities, will service individual homes and offices around the country." [BYV+09]

Confirming Kleinrock's prediction, computing is migrating in a direction where people develop software for an incredible amount of people so it can be used as a service, instead of running said software on their personal computers. Different providers such as Amazon, Google, IBM and Sun Microsystems are now establishing data centers dedicated to hosting Cloud Computing[2] applications spread around the world in order to ensure redundancy and reliability in case one of the datacenters fails.

User requirements for Cloud services are complex and varied, so service providers need to know they can be flexible when delivering those services at the same time they keep the users clear from the infrastructure on which those services stand.

Computing services are available instantly when anyone needs them and the consumers only required to pay the providers when they actually access and use those resources. Consumers no longer have the need to invest in and maintain complex IT infrastructures and software developers are facing new challenges. They must create custom made software that will be used as a service, instead of the traditional practice of installing the software in the users' machines. Some people state this is the era of pervasive computing, where computation and information are available all the time. [McF]

Having this in mind, FEUP has started developing a private cloud project at its Informatics center (CICA - Centro de Informática Prof. Correia de Araújo). As it will be discussed in greater detail in Chapter 3 - Problem Statement - this document reports the work realized on the front-end of the private cloud project.

1.2 Motivation and Objectives

Some computing infra-structures, namely Grids[3] and Clusters[4], can be rather inflexible when compared to Clouds, as the latter are supposed to allow the user to take advantage of a myriad of services, and not just computing power. [Sun]

However, as powerful as these infra-structures can be, they can be deemed useless if people who need to work with them, cannot do it because they have no knowledge of the technologies. As such, this type of issue causes a lack of growth in the use of FEUP's computing system.

This project aims at increasing the usability of the current computing system that exists at FEUP and with this, increase its usage and stop the lack of growth. In order to achieve this goal, it was proposed that a web portal would be developed which would simplify the access to the system. This portal would have a list of common software packets and Linux distributions that

[2] Using multiple server computers via a digital network as if they were a single computer.

[3] Distributed systems that are loosely coupled, heterogeneous and geographically dispersed and act together to perform very large tasks.

[4] Group of linked computers working closely together as if they were a single machine.

the user could choose from and create an VM image which would be used to run the researcher's computing job.

Furthermore, this project covers an emerging technology in the cloud computing field (*OpenStack*), a technology that surpassed some of its direct competitors, even though it was started later.

1.3 Dissertation Structure

This Dissertation is structured as follows:

Chapter 1: "Introduction" — This chapter.

Chapter 2: "State of the Art" — Bibliographic review on some of the most relevant areas for this project and on the technologies that could (and some will) be used.

Chapter 3: "Problem Statement" — Exposes the "problem" that originated this dissertation, as well as the solution proposed and its relevance and contribution.

Chapter 4: "Approach and Results" — Reviews the current state of the implementation. FIXME

Chapter 5: "Conclusion" — Reviews the project, drawing conclusions on what was implemented and what remains to be done, with reference to Chapter 4. It provides a summary of the contributions and the future work and how it can be used for whoever wishes to work with these technologies in these environments.

Abbreviations are used throughout this document to improve readability, all of which can be found in . References and citations appear inside [square brackets] and in highlight color. Highlighted text will act as an hyperlink when visualizing this document in a computer.

Chapter 2

State of the Art

This chapter presents a bibliographic review on the subjects covered by this project. Firstly, the concepts of Virtualization, Grid and Cloud Computing are presented, as well as a comparison between these two areas. There is also an analysis on existing developments, applications and projects on the area of Cloud and Grid Computing, with greater focus on two community driven projects (OpenStack and OpenNebula), since they are currently in use at FEUP. Some of the computing technologies in use at FEUP are also presented and discussed.

2.1 Virtualization and Virtual Machines

Throughout this project, Virtual Machines (VMs) are mentioned in great amount and as such, they deserve a special section.

Certain problems arise when the requirements of different virtual organizations (VOs) that need to use the same resources are in conflict or are incompatible with site policies. The software available on clusters cannot guarantee isolation of different communities and maintain resource availability while ensuring good utilization of those said resources. This is where Virtual Machines (VMs) come into play. They are emulations of lower layers of computer abstractions on behalf of the higher layers and allow the isolation of the applications from the hardware and neighbour VMs and customizing the platform so it suits the user's needs. [FFK+06, ZKFF05]

Virtualization benefits include an improvement in fault isolation and independence from guest VMs, performance isolation and simplifying the migration of VMs across different physical machines. These benefits enable VMs to share pools of platform and data center resources. [NS07]

The ability to serialize and migrate the state of a VM paves the way for better load balancing and improved reliability that cannot be achieved with traditional resources. Deploying virtual clusters - set of VMs configured to behave as a cluster and intended to be scheduled on a physical resource at the same time [ZKFF05] - of diverse topologies requires the ability to deploy many VMs in a coordinated manner so that sharing of infrastructure, such as disks and networking, can be properly configured. This can become more costly than the deployment of single VMs.

In order to understand more, it is necessary to define virtual workspaces (VWs). These are an aggregation of an execution environment and the resources allocated to that specific environment. It is described by the workspace metadata, containing all the information needed for deployment. An atomic workspace, representing a single execution environment, specifies the data that must be obtained and the deployment information that must be configured on deployment. It is also needed to specify a requested resource allocation, something that describes how much of each resource should be allocated to the workspace.

These atomic workspaces can then be combined to formed what is called a virtual cluster. Foster et al propose an aggregate workspace that contains one or more workspace sets - atomic workspaces with the same configuration. Cluster descriptions can be defined in ways that atomic workspaces can be constructed flexibly into more complex structures, organizing at the same time the infrastructure sharing between the virtual nodes. This deployment enables the user to specify different resource allocations for different members of aggregates defined like this. Foster et al consider that the trade-off they obtained is acceptable, as the slowdown suffered was of about 5% and considering that virtual machines offer unprecedented flexibility in terms of matching clients to resources. [FFK+06]

Zhange et al also approached the virtual cluster theme in an article written with both Foster and Freeman, where they combine virtual clusters with Grid technology. The authors only considered two types of node within the cluster: head-nodes and worker nodes. They optimized the loading of the virtual images through image cloning (only transferring one image for all the worker nodes and one image for the head-node, therefore cloning all the worker node images at either staging or deployment time) and they considered that the cost of virtual cluster deployment and management is a good justification for expecting that they may be used for VOs for large groups of short jobs and single long-running jobs. They also found that the cost of running batch jobs in a a virtual cluster was very acceptable.[ZKFF05]

Katarzyna Keahey and Tim Freeman introduce the term *contextualization* in order to describe the process of quickly deploying fully configured images and adapt them to their deployment context, for single VMs. The authors understand *contextualization* as the process of adapting an appliance to its deployment context (an appliance defining an environment as an abstraction independent of its deployment). They are deployed dynamically and are potentially associated with a different context. According to the authors, they can also fulfill three different roles:

1. Appliance providers - they configure environments, maintain them and guarantee their consistency;
2. Resource providers - they provide resources with limited configuration requirements that are designed to support appliances but no longer to provide end-user environments for multiple communities;
3. Appliance deployers - they coordinate the mapping of appliances onto available resource platforms and information exchange between groups of appliances to enable them to share information.[KF08]

There are different types of approaches, since providers can have the applications running inside VMs or provide access to the VMs as a service (Amazon Elastic Compute Cloud), enabling the users to install their own applications. With virtualization, companies are trying to save power by getting the most of what they consume. Running several operating systems inside one machine, they can run independently and CPU idle time is kept to a minimum.[Wei07]

Virtualized computing clusters offer the advantage of being able to transform themselves to the user's needs. However, as pointed out by Nishimura et al, previous work has shown that the system does not scale when increasing the number of VMs and their detailed configuration is not allowed. To counter this issue, Nishimura et al propose a new way of managing virtual clusters so that a flexible and fully-customizable system integration by creating VMs on-the-fly is achieved.

The authors also propose the creation of *virtual disk caches* (VD caches), in order to reduce software installation time. This VD cache is created when a user requests it and is automatically destroyed to keep the total cache size within the given space. What the authors did was that when an installation request is made by the user, the system selects physical resources to host a virtual cluster for the request, instantiates a set of VMs and installs the operating system and other requested software to them. The experiments conducted by the authors using a prototype implementation showed that installing a 190-node virtual cluster can be done in 40 seconds, indicating that the installation of a 1000-VM could be done in under two minutes.[NMM07]

Nathuji and Schwan addressed the issue of integrating power management mechanisms and policies with the virtualization techniques deployed in virtual environments. They propose the *VirtualPower* approach which aims to control and synchronize the effects of the power management policies applied by the VMs to the virtualized resources.

The authors propose this approach having in mind the current limitations in battery capacities and the power delivery and cooling limitations existent in data centers when they try to handle the constant demands of performance and scalability. The authors state that their approach can exploit the hardware power scaling and the methods that control the power consumption of the underlying platforms. It takes guest VMs' power management policies and coordinates them through the system in order to achieve the objectives.

The power management actions are encoded as a set of rules, these being based on a set of mechanisms which serve as a base to implement the power management methods. This approach aimed to present guest VMs with a set of power states and then use the state changes requested by the VMs as inputs to virtualization-level management policies, including those to use specific platforms and their power management capabilities, along with policies that take into consideration goals derived from the applications running through the whole system and from global constraints, such as rack-level limitations on maximum power consumption.

Their findings showed that it is possible to respond to specific power management goals and policies implemented in guest VMs without a need for application specificity to be established at the virtualization level. [NS07]

It is extremely important to discuss *OpenNebula*, as this project will interact with it and as such, it is approached in a later section. [Ope]

2.1.1 Hypervisors

An hypervisor is a piece of software that emulates the functioning of certain hardware, a process called *Hardware Virtualization*. KVM - Kernel-based Virtual Machine - an open-source virtualization software[1] is used on the back-end of the project..

The following features for KVM were identified:

- Virtualization using hardware virtualization extensions, such as Inter-VT and AMD-V, thus enabling faster virtualization;
- Symmetric Multi Processor emulation - enables multiprocessor hardware emulation;
- Live migration of VMs between hosts, allowing VM relocation without downtime;
- Paravirtualized networking and block devices, which enables faster emulation of those devices. [Car11]

2.2 Grid Computing

Buyya et al believe that Grid computing facilitates the sharing, selection and aggregation of geographically dispersed resources, be it supercomputers, storage systems, data sources or even special assets owned by organizations for solving large-scale resource-intensive problems in different areas of expertise, and that was Grid computing's motivation. Buyya also created a definition for "Grid" at the 2002 Grid Planet conference held in San Jose, United States:

> "A Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed 'autonomous' resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements."[BYV+09]

Ian Foster, one of the most revisited authors regarding Grid computing, states that the "Grid" must be looked upon in respect of the applications it contains, the business value it generates and the scientific results it is capable of returning, instead of its architecture. Carl Kesselman and Ian Foster wrote the following definition in their book "The Grid: Blueprint for a New Computing Infrastructure":

> "A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities."[KF98]

Foster and Steve Tuecke redefined the definition, this time referring social and policy issues, affirming that Grid computing is related to resource sharing and problem solving in a coordinated

[1] Nuno Cardoso compared XEN and KVM, but came to the conclusion that neither offered an advantage over the other, so KVM was chosen due to its simplistic installation process.

manner and that these occur in dynamic, multi-institutional virtual organizations, the aspect to remember being the power to do something with the result. The authors also stated that they are preoccupied with the "direct access to computers, software, data and other resources."

As such, Foster proposes (as pointed out by the title of his article) a three point checklist that defines what a Grid system should be:

- The Grid should coordinate resources that are not subject to centralized control – Integration and coordination of both users and resources that live within different domains;
- The Grid should use standard, open, general-purpose protocols and interfaces, as this will allow the establishment of dynamic resource-sharing arrangements and the creation of something more than an agglomerate of incompatible and non-interoperable distributed systems;
- The Grid should deliver nontrivial qualities of service, such as response time, throughput, availability, security, co-allocation of multiple resource types to meet complex user demands, resulting in the utility of the combined system to be greater than just the sum of its parts.

Foster also states that the Web is not a Grid, as though its general-purpose protocols support the access to distributed resources; they do not coordinate their use to deliver qualities of service.

Some large-scale Grid deployments inside the scientific community abide by the three points described by Foster, such as NASA's Information Power Grid and the TeraGrid, which will link major U.S. academic sites, as they integrate resources from several institutions, use open and general-purpose protocols (Globus Toolkit, which will be discussed in further details later on this report) to negotiate and manage sharing and they address multiple dimensions of the quality of service, such as security, reliability and performance.[Fos02]

Stockinger started a survey where he contacted over 170 Grid researchers globally spread in order to obtain a general feel on how the Grid was being defined. The results showed that the Grid infrastructure should provide a set of capabilities, such as:

- Description of available resources, what they are capable of doing and how they are connected;
- Visibility into the state of resources, including notifications and logging of significant events and state transitions;
- Assurance of the quality of service across an entire set of resources for the lifetime of their use by an application;
- Provision, life-cycle management and decommissioning of allocated resources;
- Accounting and auditing of the service;
- Security.

The results also showed that a Grid should have a set of characteristics, including:

- Collaboration - sharing resources in a distributed manner;
- Aggregation - the Grid is more than just the sum of all parts;
- Virtualization - Services are provided in a way that the complexity of the infrastructures is hidden from the end-user through the creation of an abstract "layer" between clients and resources;
- Heterogeneity;
- Decentralized control, Standardization and Interoperability - supporting Ian Foster's definition;
- Access transparency - users should be able to access the infrastructure without having to preoccupy themselves how they are doing it;
- Scalability;
- Reconfigurability;
- Security - specially since the systems are often spread through multiple administrative domains. [Sto07]

The members of the EGEE (Enabling Grids for E-sciencE Project) also state that their Grid abides by some of the characteristics mentioned above, namely "decentralized control", "heterogeneity" and "collaboration" [B08́]. Their Grid is described in greater detail in the "Grids VS Clouds" section below.

Bote-Lorenzo et al also identified some core Grid characteristics that coincide with Stockinger and Ian Foster's definitions. These include scalability, heterogeneity, resource coordination and dependable, consistent and pervasive access. The propose the following definition for a Grid:

> "... large scale geographically distributed hardware and software infra-structure composed of heterogeneous networked resources owned and shared by multiple administrative organizations which are coordinated to provide transparent, dependable, pervasive and consistent computing support to a wide range of applications. these applications can perform either distributed computing, hight throughput computing, on-demand computing, data-intensive computing, [...]"[BLDGS04]

Baker et al say that the Grid has evolved from something static and carefully configured, to what has been witnessed in the past years, where it became a seamless and dynamic virtual environment, capturing the attention from the industry and thus making an impact on the Grid's architecture and protocols and standards.

The authors also describe a few standards and organizations that have been actively present in the Grid's environment over the past years. These include the Global Grid Forum (GGF), a community-driven set of groups which goal is to develop standards and best practices for wide-area distributed computing. The GGF creates a group of documents that provide some information to the Grid community, dividing its efforts into several categories, including architecture, data and security.

The authors also approach the World Wide Web Consortium (W3C), an international organization created to promote common and interoperable protocols. This organization was responsible for creating the first Web Services specifications in 2003, such as SOAP and the Web Services Description Language (WSDL). According to the authors, the most important Grid standard to appear recently is the Open Grid Services Architecture (OGSA), which goal is to define a common, standard, and open architecture for Grid-based applications. It was announced by the GGF at the Global Grid Forum in 2002 and in March 2004 it was declared by the GGF to be the flagship architecture.[BAFB05]

Iosup, Dumitresco and Epema analyzed four Grid implementations and the differences on their workload:

- Firstly, they covered the LHC Computing Grid, which testbed has 25,000 (twenty five thousand) CPUs and 3 PetaBytes of storage. Jobs are managed and routed to resources via a Resource Broker, which tries to conduct the job matchmaking and balance the workloads at the global level. The site used by the authors had around 880 CPUs;

- Secondly, they looked at the Grid3 testbed, representing a multivirtual organization environment that sustains production level services required by various physics experiments. It is composed by more than 30 sites with 4500 (four thousand five hundred) CPUs;

- Thirdly, they analysed the TeraGrid system - used for scientific research - which has over 13,6 TeraFLOPS of computing power and can store 450 TeraBytes of data;

- Finally, they reviewed the DAS-2 environment, which has 400 CPUs spread over five Dutch Universities and its workload ranges from single CPU jobs to very complex ones. These can be submitted either via the local resource managers or to Grid interfaces that communicate with them.

They discovered that while Grid research focuses on complex application types, most of the applications encountered were extremely easy to run in parallel (embarrassingly parallel applications).

The authors identified two large problems, a scale (origin and size of the data that must be collected) and a methodological (missing components of the information) problem. In order to address the first problem, the information should ideally come from three different sources:

- Local and Grid scheduler - without these logs, job arrival and dependency information can be lost and an analysis of site-related performance metrics cannot be done;

- Grid AAA (authentication, authorization and accounting) modules - these modules provide the information regarding the link between jobs and their owners;

- Monitoring systems - without the information these systems provide, it is impossible to understand how the applications are running within the Grid and to quantify the system utilization.

The authors concluded that a small number of VOs and users control the workload in terms of submitted jobs and consumed resources, system evolution can appear at the system, VO and user level and should be considered when provisioning resources.[IDE+06]

Malcolm Atkinson from the National e-Science Center in the United Kingdom, says the following:

> "With Web Services we allow a thousand flowers to bloom. With a Grid we organize the planting and growth of a crop of plants to make harvesting easier."[Sto07]

Iosup et al end their article with the following quote:

> "[...] conclude that Grids are not yet utilized at their full capacity."[IDE+06]

which serves as a conclusion for this section.

2.3 Cloud Computing

Similarly to the Grid, many definitions arise when one talks about the Cloud. Presently, it is considered normal to obtain access to content spread over the Internet without a reference to the hosting infrastructure that lies underneath it. This infrastructure is made of data centers that are being monitored by service providers.

Buyaa et al state that Cloud computing extends this paradigm in where the capabilities of the applications are viewed as complex services that can be accessed over a network. The authors also believe that the Cloud is an infrastructure from where businesses and users can access applications from anywhere in the world anytime they want. Cloud computing's services need to be reliable, scalable and sufficiently autonomic to support omnipresent access, dynamic discovery and they need to support composability, as they must permit to be reassembled and selected in any order to comply to the user's requirements.

The authors have a definition of their own:

> "A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers." [BYV+09]

They believe that Clouds are the new datacenters with hypervisor technologies such as VMs, with services provided on-demand as a personalized resource collection in order to meet the

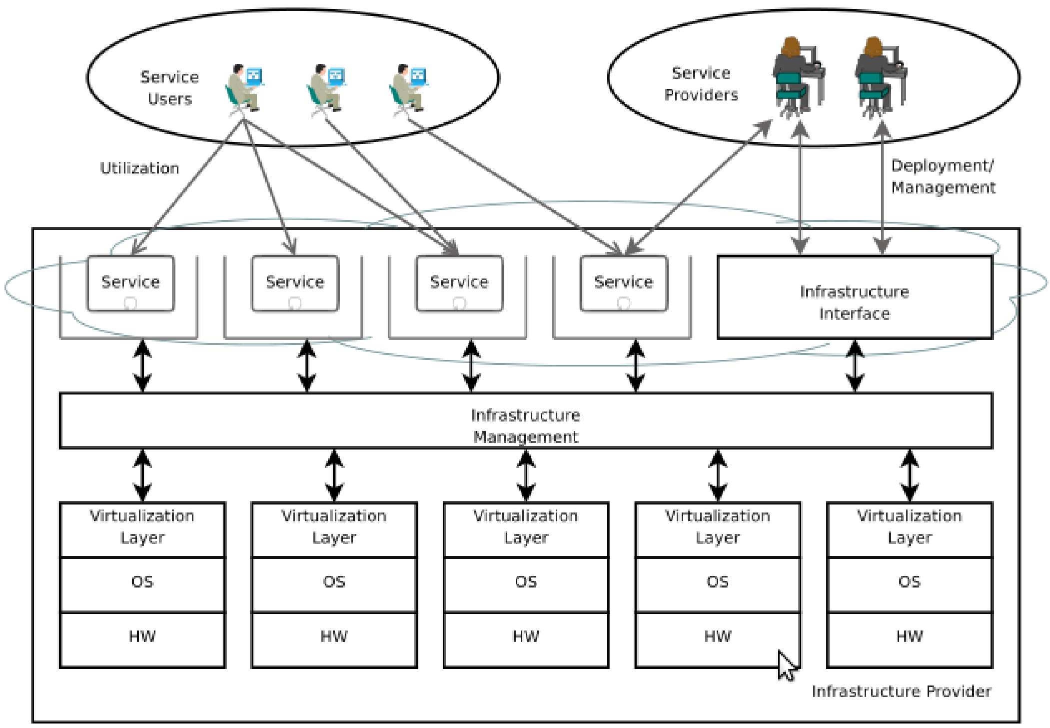


Figure 2.1: Cloud Actors. [BYV+09]

service-level agreement, which should be established *à priori* with a “negotiation” and accessible as a composable service via Web Service technologies.

Vaquero et al state that the paradigm of Cloud computing shifts the infrastructure to the network in order to reduce the costs that are normally associated with the management of hardware and software resources.

Having in mind Gartner’s Hype Cycle[2], Vaquero et al state that Cloud computing is now in its first stage – Positive Hype – mixing every definition that appears into an overly general term that confuses every single person. The same thing that happened to Grids can be applied here. There are no widely accepted definitions (Foster’s being the most accepted one) and a clear definition can help transmit what it actually is and how businesses can reap benefits from it.

There are many Cloud definitions, but they all focus on certain technological aspects. Thus, Vaquero et al try to analyze all the features of Cloud computing in order to reach a clearer definition.

The authors try to distinguish the different actors and scenarios that can arise:

[2]Graphic representation of the maturity, adoption and social application of specific technologies. It has five phases: 1 Product launch that generates interest; 2 - Frenzy of publicity generates over-enthusiasm and unrealistic expectations; 3 - Technologies fail to meet expectations and become unfashionable; 4 - Press stopped to cover the technologies, but some businesses continue to experiment and understand its benefits; 5 - Benefits become widely demonstrated and accepted. Technology becomes stable and evolves.

The actors:

Service Providers make services accessible to Service Users through Internet-based Interfaces. The computing infrastructure is offered "as a service" by the Infrastructure Providers, moving computing resources from the SPs to the IPs, in order to give the firsts flexibility and reduced costs.

The scenarios:

- Infrastructure as a Service – IPs are responsible for the management of a large set of computing resources, such as storing and processing capacity. If they use virtualization, they can split, assign and dynamically resize the resources to build ad-hoc systems as the customers (SPs) demand, by deploying the software stacks that run their services.

- Platform as a Service – Clouds offer an additional abstraction layer – they can provide the software platform where systems run on. The sizing of the hardware resources demanded by the execution of the services is made in a transparent manner. The applications developed are run on the provider's infrastructure and are delivered through the Internet from the provider's servers. The Google Apps Engine is a very good example.

- Software as a Service – Cloud systems can host many services that users can be interested in, such as online word processors or even Google Apps. [KG09, VRMCL08]

In the article written by Vaquero et al, many Cloud definitions are gathered. Markus Klems states that the key elements for the Cloud are immediate scalability and the optimization of resources usage, these being bring provided by increased monitoring and automation of resources management. Jeff Kaplan and Reuven Cohen prefer to focus on the business model, paying more attention to the collaboration and pay-as-you-go and reducing the costs of investment. Douglas Gourlay and Kirill Sheynkman define the Cloud as being simple virtualized hardware and software, combined with monitoring and provisioning technologies. [GKC+, VRMCL08]

McFedries believes that the basic unit of the Cloud is nothing other than data centers - huge collection of clusters - that can offer a large supply of computing power and storage simply by using whatever resources they can spare.[McF]

Kevin Hartig defines Cloud Computing as being able to access resources and services needed to perform certain tasks with needs that are constantly changing. The application or user requests access from the cloud rather than on a specific endpoint of the network or a resource. The Cloud becomes a virtualization of resources that is both self maintainable and manageable, a view also shared by Jan Pritzker, who focuses his definition on virtualization and on-demand resource allocation. Other authors such as Reuven Cohen, Praising Gaw, Damon Edwards and Ben Kepes (to name a few) are strong believers that Cloud computing is nothing more other than a buzz word, grouping concepts such as deployment, load balancing, provisioning and data and processing outsourcing.[GKC+]

Having this in mind, Vaquero et al believe that the Cloud is a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services), these being dynamically reconfigured to adjust to a variable load (scaling) also allowing for an optimum resource utilization. The pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized Service-Level Agreements. The authors also state the set of features that resemble this minimum definition would be scalability, pay-per-use utility model and virtualization. [VRMCL08]

Brian Hayes states in his article that even though the future of Cloud computing is still unclear, there are a few directions in which it can go. One of those directions is Web based services, such as Google Docs or even Photoshop Express. Salesforce.com also offers a variety of online applications and its slogan is actually "No software!".

As mentioned earlier in the report, Amazon.com also ventured into this new paradigm, offering data storage and computing capacity, each of these services being able to expand and contract as the users need (elasticity) and Google has its App Engine, providing hosting on Google server farms.

There is great concern in terms of scalability in the Cloud, as it might be necessary to organize resources so that the program runs flawlessly even though the number of concurrent users increases might arise. Hayes also mentions that Cloud computing raises questions in terms of privacy, security and reliability, since personal documents are being delivered to a third-party service.[Hay08]

Nicholas Carr writes in his book that a shift is happening, where the Cloud is becoming similar (if not equal) to the electric grid, as we can connect to the Cloud and get data, storage space and processing power cheaply and instantly (Utility Computing). [Car09]

Aaron Weiss writes in his article that the Cloud is robust, even self-healing, as it has many sources from where to get the power to recover from whatever accident occurs. Weiss states that the Cloud is also very power consuming, as roughly 50 percent of the energy it consumes comes from the cooling process alone. Giants such as IBM and Microsoft are also scouting locations where the hydroelectric power is cheaper and greener, so they can establish their cluster centers.[Wei07]

2.3.1 Utility Computing

Computing is going in such a direction that the services that are made available to the user are being done in such a way that computing is becoming equal to traditional utilities such as water, gas, electricity and telephone services.[BYV+09]

In an article published by InfoWorld (formerly the Intelligent Machines Journal), Utility computing is mentioned as being a form of Cloud computing, where storage and virtual servers are being offered and can be accessed on demand, such as the services offered by Amazon.com, Sun or IBM. [B08]

IBM Global Services provide the following definition for Utility Computing:

> “Utility computing is the on demand delivery of infrastructure, applications, and business processes in a security-rich, shared, scalable, and standards-based computer en-

> vironment over the Internet for a fee. Customers will tap into IT[3] resources - and pay for them - as easily as they now get their electricity and water." [Rap04]

Utilities have the following characteristics:

- Necessity - Users depend on utility services to fulfill their day-to-day needs. It takes time for distribution networks to spread and costs to decline, as it also takes time for users to adapt to the service. Once they do, the service may grow in importance as users begin to find new ways to reap benefits from it;

- Reliability - The service must be readily available when and where the user requires it, as a temporal or intermittent loss of service may cause several issues to the user. Redundancy must be built into production capacity in order to make up for hypothetical service failures;

- Usability - Users have a "plug-and-play" mentality and they need to feel at ease with whatever feature they are using;

- Utilization rates - Utilities are driven by a necessity to carefully manage utilization rates. User demands for utility serves mays vary over time and across service regions. This may lead to spikes in utilization of the service and under-utilization in off-peak periods. Service providers must have in mind that how the service is billed may influence how users use that service;

- Scalability - As production capacity grows, the unit cost of production shrinks. It might be expected that as the demand for the service rises, the quality of service may decline or vice-versa [Rap04].

Bhattacharya and Vashistha state that utility based computing allows computing resources to be available for a customer on demand, as the customers subscribe to the services of the utility provider and only pay for the quantum of the resources used. This allows any customer to cut down on IT infrastructure spendings as they can simply subscribe to the provider's services and use the computing resources at will, only paying for as much as they use. Typical measures of usage include metered CPU hours and memory space usage [BV08].

Ross and Westerman write in their article that utility computing relies on several important technical capabilities to deliver what it promises - services available on-demand. The authors believe that for most firms, the impact of utility computing will be on the extent and nature of outsourcing. The benefits that can be obtained only enhance the current benefits of IT and business processes outsourcing: lower cost, variable capacity and increased strategic focus. On demand capacity leads to firms to invest less in computing capacity. Advances in autonomic computing may reduce the number of people needed to monitor operations and thus reduce labor costs.

The authors believe that firms will be able to do more with less and will be able to allocate their most strategic resources to their most strategic opportunities [RW04].

[3] Information Technologies

2.4 Grids VS. Clouds

As one knows, Grids and Clouds share a few goals, such as reducing computing costs and increasing flexibility and reliability through the use of third-party operated hardware.

Vaquero et al lay out a very comprehensive list of features and discuss the similarities and differences between them. The list includes resource sharing, heterogeneity, virtualization, security, the offer of high level services such as metadata search, the awareness of architecture, dependencies and platform, software workflow, scalability and self management, standardization, payment model and quality of service. The list is shown in Figure A.1 which is in Appendix A.

The authors also believe that Grids are meant to be user friendly, virtualized and automatically scalable utilities, something that steps into the Clouds' path, but they still need to be able to incorporate virtualization techniques in order to obtain some advantages already present in the use of Clouds, like migrability and hardware level scalability.[VRMCL08]

A few members of the Enabling Grids for E-sciencE (EGEE, now part of the European Grid Infrastructure) performed a comparative analysis on Grids and Clouds, focusing two implementations of both: the EGEE project for Grid and the *Amazon Web Service* (AWS) for Cloud, using metrics such as performance, scale, ease of use, costs and functionality, amongst others. The Grid in use by the EGEE runs on gLite, an open source software which had development funding from the EGEE, described in a later section of this document, as it is used in some extent by FEUP's cluster system.

When comparing both EGEE Grid and the *Amazon Web Service*, the authors of the analysis encounter a set of differences and similarities:

- The AWS does not expose how they operate their data centers and how they implement the user interfaces, execute the user requests and maintain their accounting, its back-end is still a grey area;
- The EGEE Grid exposes both user interface as well as the resource interface to permit providers to connect their resources. The AWS hides this second interface;
- The authors assume that on the resource side, both systems work in similar manner, as both cases require a queueing mechanism whether the data center is dispatching a grid job via a batch system or is requested to instantiate a new virtual machine;
- The greatest benefit of the Cloud proposed by Amazon is its interfaces and usage patterns, focused on simplicity;
- Both services are not fail-proof, but the authors consider that a centralized Cloud might not be able to provide the resilience that the distributed nature of EGEE does;
- Grids are typically used for job execution - limited duration execution of a program, part of a larger set of jobs, consuming or producing a significant amount of data. Clouds, even though they support a job usage pattern, they seem to be more often used for long-serving services;

- Amazon bills users for computing resources usage with a minimum of one hour usage. This stops being efficient when dealing with a large number of small jobs;
- Elasticity in the Grid is made by adding worker nodes at a site or adding new sites;
- The complexity in the Cloud is kept server-side, which makes its entry point very low, something that is still considered a goal to achieve for Grids. [B0́8]

2.5 Technology Review

With the shift of the computing industry towards a provision of Platform as Service and Software as a Service, consumers can access resources on-demand without having to preoccupy themselves with time and location, Buyya et al believe that there will be an increasing number of Cloud platforms being developed [BYV+09]. Two of those platforms are *OpenNebula* and *OpenStack*, open-source toolkits for Cloud management. In this report they will both be analyzed, but as they play a major role in this project, they were given their own section in this chapter (2.7).

The technologies considered to build the web application will be presented in this section, as well as which one will be used.

Amazon's computing service (2.5.2), *Google's App Engine* (2.5.3) and *Microsoft Azure* (2.5.4) are described in this section.

2.5.1 Web technologies

As one of the objectives is the creation of a "Web System", it is necessary to approach the candidate technologies for the application.

Since two cloud platforms will be revised, they are both built in two different programming languages and both of those technologies can be used in a web context (with the appropriate platforms supporting them), *Python* on the *Django* platform and *Ruby* on the *Rails* platform will be discussed in this section of the document.

2.5.1.1 Ruby on Rails

> "Ruby on Rails is a breakthrough in lowering the barriers of entry to programming. Powerful web applications that formerly might have taken weeks or months to develop can be produced in a matter of days." -Tim O'Reilly, Founder of O'Reilly Media [HH]

Ruby on Rails is a Web 2.0 framework that attempts to combine PHP's simple immediacy with Java's architecture, purity and quality. It forms an environment and provides all the tools to create business-critical, database-supported web applications. Its basic objectives are simplicity, reusability, expandability, testability, productivity and maintainability.

It implements an MVC (Model-View-Controller) architecture, which clearly separates code according to its purpose.Ruby code is easy to read and is based on languages such as *Python*, *Perl* and *Lisp* [BK07].

Ruby on Rails official website offers a wide range of APIs, guides and books, which make for an extremely well documented framework [HH].

2.5.1.2 Python and Django

Django is a high-level *Python* web framework that encourages rapid develop and clean, pragmatic design. It was designed to handle two challenges: intensive deadlines and the requirements of the experienced web developers who wrote it.

Just like *Ruby on Rails*, *Django* focuses itself on the DRY [4] principle, which states:

> "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system." [CCd]

Django's documentation is extremely extensive, and can be found in its official website [5] and in the online version of "The Django Book", a free book about the *Django* Web Framework [6].

Python and *Django* are the best candidates to be used, as `vmbuilder` is a suitable choice for the needs of this project, since it can run inside Ubuntu and is Python based. Python will also be used in any scripting necessary, unless it cannot be done specifically in Python. The back-end scritping is written in Bash, something that can be easily integrated into Python modules [sei].

In addition and as it will be mentioned later in this document, *OpenStack* is coded in *Python* which makes integration easier than what would happen if different languages were used.

2.5.2 AWS - Amazon Web Services

The *Amazon Web Services* consist of several components, but only two will be taken into consideration in this document, as they are the most relevant to the work discussed: *Amazon's Simple Storage System* (2.5.2.1) and *Amazon's Elastic Computing Cloud* (2.5.2.2).

2.5.2.1 Amazon's Simple Storage Service

The core service for the *Amazon Web Services* is the *Amazon's Simple Storage Service*, that gives the user the power to store large amounts of data in a reliable way which does not hinder its availability. Data is accessed through protocols such as SOAP[7] and REST[8], while also being able to be accessible via normal web browsers. The storage model runs on a two-level hierarchy, where the users can create *buckets* and place data *objects* in those buckets. Strings are used as keys for both buckets and objects, thus being able to be easily incorporated in URLs. Users are charged 15 US cents per Gigabyte per month, each user being able to have up to 100 buckets and each can hold up to 5GB of data.[Haz08]

[4] Don't Repeat Yourself.

[5] `https://docs.djangoproject.com/en/1.4/`

[6] `http://www.djangobook.com`

[7] Simple Object Access Protocol - Used to exchange information in the implementation of Web Services in computer networks.

[8] Representational State Transfer - Style of software architecture for distributed hypermedia such as the World Wide Web.

2.5.2.2 Amazon's Elastic Computing Cloud

Physically speaking, the *Elastic Computing Cloud* (EC2) is a large number of computers on which Amazon provides time to paying customers, these computers being spread all over the United States. EC2 is based on the XEN virtualization technology, which allows one physical computer to be shared by several virtual ones, each with its own operating system.

Through the use of virtualization, the users create an image of their software environment using the tools provided. This will be used to create and instance of a machine in Amazon's Cloud. Customers can freely choose configuration templates for their instance and they can create and destroy the instances at will, enabling the software to scale itself to the amount of computing power it needs.[B0́8, Haz08]

Amazon has released *Elastic IPs* (Static IPs for Dynamic Cloud Computing), which allows the assignment of static IPs to dynamic resources that are deployed via EC2, as well a service that enables users to request EC2 instances to be geographically distributed, as a response to the demand for EC2 IP addresses in a static range for application range for applications like email service hosting, as well as providing a safety net in case the operations of an Amazon Web Services data center go awry.

Amazon provides a variety of ways of requesting the EC2 instances, namely through the use of Web Services, supporting Buyya et al's Cloud definition previously mentioned in the document.

Amazon has also introduced its own performance unit named "EC2 Compute Unit". Since Amazon ventured into the Utility computing field, model it follows differs from the traditional way developers were formatted to think about CPU resources. Instead of renting a certain processor for several months or years, it is now rented by the hour. One EC2 Compute Unit provides the CPU equivalent of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.[Ama]

2.5.3 Google Cloud - Google's App Engine

The *Google Cloud*'s official name is *App Engine* or *Appengine*. It gives developers the ability to run web applications on Google's infrastructure, the same that is being used by *Google* for *GMail* and *Google Docs*. The Cloud appears to be a platform accessible over the Internet with limitless hardware, the latest software and abundant storage for deploying web applications. The *App Engine* has the following features:

- Automatic horizontal scaling and load balancing;

- APIs[9] for authenticating users with Google Accounts and for sending emails. No system administration is needed by the user to set up or allow access to these APIs;

- Fully featured Eclipse developed environment that simulates *Google App Engine* on the localhost for development and testing;

[9] Application Programming Interface

- Persistent storage and support for transactions and queries using the standard JDO[10] and JPA[11] APIs;
- Generous free quotas, which allow small universities to have access to the same hardware and software as large industries. Each user can have 10 applications created, each with 10 versions, which totals an effective development environment of 100 applications. A free account supports six and a half CPU hours a day, with 1GB of stored data and sending email to 2000 recipients a day and a max of 5 million page views a month;
- It is free, with no contracts to sign, no hardware expense and no system administration costs for maintaining, updating, patching or backing up *App Engine*;
- Eclipse plug-in available for Apple, Linux and Windows, which allows standard debugging using Eclipse debug tools. It provides menu based functionality to automatically upload the application to the Google App Engine;
- Requires no system administration;
- Simple web based, user friendly console.[HP10]

2.5.4 Microsoft Azure

Microsoft Azure platform is a cloud computing platform which offers a set of cloud computing services similar to those offered by Amazon Web Services. *Windows Azure Compute* (Microsoft's counterpart to Amazon's EC2), only supports Windows virtual machines and offers a limited variety of instance types when compared with Amazon's EC2. Its instance type configurations and cost scales up linearly from small to extra large and its instances are available in 64 bit x86_64 environments.

It has been speculated that the clock speed of a single CPU core in *Azure*'s terminology is approximately 1.5 GHz to 1.7 GHz.[GWQF10] Windows Azure enables developers to build, host and scale applications in Microsoft datacenters, not requiring upfront expenses, long term commitment and users only pay for the resources they use. Windows Azure relieves the user from the effort of configuring load balancing and failover, is designed to let developers build applications that are continuously available, even if they need software updates and hardware failures occur.[Mic]

2.6 FEUP's Computing System

In this section FEUP's computing system is analyzed in detail. The cluster system and the technologies it uses in its management are described and detailed. FEUP's cluster system currently uses three different technologies, *Moab*, *gLite* and *Condor*. Currently FEUP currently has both

[10] Java Data Objects
[11] Java Persistence API

OpenNebula and *OpenStack* running (the last one for research purposes only), both technologies having been already discussed in the previous section(2.7)

2.6.1 Clusters

Three different technologies are currently in use by FEUP's Cluster system: *Moab*(2.6.1.1), *gLite*(2.6.1.2) and *Condor*(2.6.1.3).

2.6.1.1 Moab Cluster Suite

Moab Cluster Suite is a proprietary tool for high performance computing systems, developed by the company *Cluster Resources*. It has built-in modules for work management, Cluster administration and monitoring, report creation. It is composed by three essential components:

- **Moab Workload Manager** - scheduling and workload management engine;
- **Moab Cluster Manager** - graphical interface for Cluster administration, monitoring and report analysis;
- **Moab Access Portal** - web based portal for job management and submission, directly focused on the end-user.

A resource manager supplies the system with basic functionalities for initiating, stopping, canceling or monitoring jobs. *Moab Workload Manager* uses a resource manager's services to get information about the state of the resources and the node workload. It is also used to manage jobs and to send information on how they should be run and it can be configured to manage more than one resource manager simultaneously. Its composing nodes can be split into three groups:

1. **Master node** - Manages the resources;
2. **Submissive/Interactive nodes** - Allow users to manage and submit jobs into the system;
3. **Computing nodes** - Execute the submitted jobs.

Is is also possible to split the nodes into two groups - source and destination nodes. The first ones are the nodes where there users, portals or other systems can submit their jobs and the latter ones are where the jobs are executed. Jobs originate in a source node and are transferred to the destination nodes. Decisions are made in the source nodes, so it is possible to choose which nodes will execute the submitted jobs. Moab also allows for the establishment of connections between several Grid systems, which permit access to additional resources.[Pin10, Resb]

2.6.1.2 gLite

gLite consists in a set of components designed with the objective of building a Grid computing infrastructure for resource sharing developed by the project EGEE (Enabling Grids for E-sciencE), also mentioned in an earlier section. *gLite* is based on four core concepts:

- **Computing Element (CE)** - Set of local computational resources, namely a Cluster. It is composed by three components:
  - **Grid Gate** - generic interface for the Cluster who receives jobs and submits it to the Local Resource Management System (LRMS);
  - **Local Resource Management System (LRMS)** - Sends the jobs to the worker nodes for execution;
  - **Worker Nodes** - Cluster nodes where jobs are executed.
- **Storage Element (SE)** - Supplies access to data storage resources;
- **Information Service (IS)** - Resource research is made through this component, which is also responsible for supplying information regarding resources and their state;
- **Workload Management System (WMS)** - Receives jobs from users, appropriately allocates a CE, saves jobs states and gets the final results.[Pin10, CER]

2.6.1.3 Condor

Condor is a free and open-source workload management system, developed by the *Condor Research Project*. It has built-in job queueing mechanisms, scheduling and priority policies, resource monitoring and management. Users submit jobs to *Condor*, which puts them in a queue, chooses when and where to execute them based on defined policies, carefully monitors their progress and informs the user when the jobs are finished. *Condor* can manage dedicated nodes or harness the CPU energy wasted in workstations that are turned on but unused. If the system detects the machine became suddenly unavailable, *Condor* can migrate the state of the job into a different machine and resume work. It offers an extremely flexible structure to assign resources to jobs, allowing these to have specific requisites and resource preferences, as well as enabling the resources to specify preferences over jobs to execute. Each machine from *Condor* can play several roles:

- Central Manager - Machine that collects information and makes the negotiation between resources and resource requests. All resource requests go through the *Central Manager*. There can only be one *Central Manager* in a *Condor* infrastructure;
- Execute - Machine which executes jobs, therefore allowing the network to take advantage of its resources. Any machine can be configured to take this role;
- Submit - Machine responsible for the job reception and submission to the *Central Manager*. Any machine can be configured to take this role;
- Checkpoint Server - Machine which stores checkpoint files for all submitted jobs.[Pin10, Tea]

2.7 Cloud Creation and Management Software

As mentioned in the previous section (2.5), *OpenStack* and *OpenNebula* are two projects that deserve their own section when talking about the developments, applications and services that exist in their field.

Since the objectives of the project involved the implementation of a virtual environment creator and manager, both these platforms were deemed worthy of a more detailed inspection. Both projects are open source (*OpenNebula* offers an "Enterprise Edition", which can put its open source status in question (2.7.2)) and one of them will run alongside the web page that will be developed and will interact with.

Image contextualization will be approached, since it is directly linked to one of the objectives of this project — creating different virtual environments according to the users' needs.

Project *Aeolus* will also be discussed in this section, as it contains one of the tools that will be discussed in the next section (Image creation) and is related to the objective refered earlier, Oz.

2.7.1 OpenStack

OpenStack is a global collaboration of developers and cloud computing technologists producing the open source cloud computing platform for public and private clouds. The aim of this project is to deliver solutions for all types of clouds by being simple to implement, massively scalable and filled with features.

First released in October 2010 and now on its fifth version (codename Essex), OpenStack has undergone major changes and revamps over the past months [CCb].

It was founded by *Rackspace Hosting* and NASA[12] (deployed as NASA's Nebula cloud [NASA])and it has grown to be a global software community of developers collaborating on a standard open source cloud operating system. Current companies involved with *OpenStack* include *OpenStack Foundation*, *Canonical*, *Cisco*, *Dell*, *Red Hat*, *SUSE* and *Yahoo!*. [Incb]

OpenStack's mission is to enable any organization to create and offer cloud computing services running on standard hardware.

All of its code is available under the *Apache* 2.0 license and as such, anyone can run it, build applications on it or submit changes back to the project. It is commoditizing the IaaS market, enabling the users to get from *Amazon* today into their own private data centers and cloud environments by using open source. [Incb]

2.7.1.1 OpenStack Architecture

The following figure depicts OpenStack's software diagram:

OpenStack has four major components:

- **Compute** - Also known as *Nova*, it is designed to provision and manage large networks of virtual machines. Provides an API so that developers who wish to build cloud applications

[12] North American Space Agency

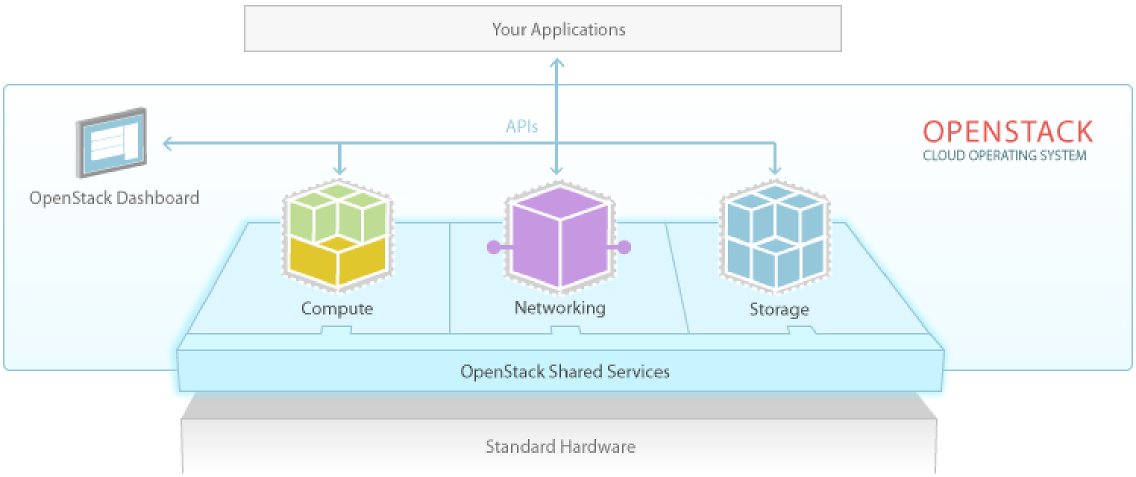


Figure 2.2: OpenStack Software Diagram. [CCb]

can access the compute resources, as well as web interfaces for administrators and users. Its architecture is designed to be flexible in the cloud design, so that no proprietary hardware or software is required and has the ability to integrate with legacy systems and third party technologies. *Nova* can manage and automate pools of compute resources and works with a great deal of virtualization technologies, enabling the administrators to use multiple hypervisors, such as KVM or XenServer.

- **Networking** - A pluggable, scalable and API-driven system for managing networks and IP addresses. Keeps the network from bottlenecking or being a limitation factor in the cloud deployment. Designed to provide flexible networking models to cater the needs of different applications and user groups. Manages IP addresses, allowing both static IPs or DHCP. Allows the administrator or the user to reroute traffic in case of maintenance or failure. *OpenStack Networking* has an extension framework which allows extra network services, such as intrusion detection systems, firewalls and VPNs to be deployed and managed.

- **Storage** - Also known as *Swift*, it is ideal for cost effective and scale-out storage [13]. It has a fully distributed and API-accessible storage platform which can be integrated directly into applications or used as a backup, achiving and data retention tool. It allows for block devices to be exposed and connected to compute instances for expanded storage, better performance and integration with enterprise storage platforms. *OpenStack Swift*'s object storage is a distributed storage system for static data such as VM images, photo and email storage, backups and archives. It has no central point of control thus providing greater scalability, redunndancy and durability. Storage clusters can scale horizontally by adding new servers. If one of the servers or a hard drive fails, OpenStack replicates its content from

[13] A storage system that uses a scaling methodology in order to create a dynamic storage environment which will support balanced data growth on an as-needed basis. Its architecture uses a number of storage nods that are configured to create a storage pool or are configured to increase computing power and is designed to scale boyth capacity and performance [Incc]

other active nodes to a new location in the cluster. Furthermore, *OpenStack* uses algorithms in order to replicate and distribute data accross different devices which allows for the use of inexpensive hard drives and servers.

- **OpenStack Dashboard** - Also known as *Horizon*, it provides administrators and users with a GUI to access, provision and automate cloud-based resources. It is extensible, making it easy to attach third party services, such as billing, monitoring and additional management tools. It is a simpler way to access the resources, which can also be done by building their own tools using either *OpenStack*'s or EC2' compatible API.

Alongside the four components described above, *OpenStack* also has a few shared services which make implementing and controlling the cloud an easier job. They are designed in a way that they can integrate with themselves as well as the components above.

OpenStack has its own identity service - named *Keystone* - which shows a central directory of users mapped to the *OpenStack* services they can access. *Keystone* acts as a common authentication system accross the operating system that the cloud sits on and can integrate with existing backend services such as LDAP. This identity service allows the administrator to configure centralized policies accross the users and systems as well as defining permissions for the four major components depicted above, whereas the users can get a list of services they can access, make API requests or log into the web dashboard - *Horizon* - to create resources linked to their account.

Another important serviced provided by *OpenStack* is its image service *Glance*. It provides discovery, registration and delivery services for disk and server images. It has the ability to snapshot a server image and store it, which can then be used as a template to get new servers up and running very quickly - and consitently in the case of setting up multiple servers - rather than installing a server OS and individually configuring the additional services required. *Glance* can store both disk and server images in a great variety of backends, including *OpenStack*'s own object storage. A standard REST interface is provided for querying information on disk images and that lets clients stream the images to new servers. The image registry supports a wide range of formats, which include images generated by KVM, Qemu, VMWARE and RAW images.

This project is under close surveillance by CICA as it is viewed as a possible substitute for *OpenNebula*. *OpenStack* is also discussed in the following chapter 3 as it is one of the focus points of the work realized.

2.7.1.2 DevStack

Since *OpenStack* still has a somewhat complex deployment process, *DevStack* was created in order to provide whoever wishes to try out *OpenStack* for development purposes.

It is essentially a set of scripts and utilities to quickly deploy an *OpenStack* cloud.

Its goals include the following:

- To enable the user to quickly build dev *OpenStack* environments in a clean Ubuntu or Fedora environment;

- To describe working configurations of *OpenStack*;
- To make it easier for developers to get familiar with *OpenStack* without the need to understand every single part of the system at once.

Created by *Rackspace Cloud Builders* [14], *DevStack* will be used as it simplifies the deployment process (and is used by FEUP's *OpenStack* researcher). *DevStack* will be further expanded in Chapter 3, when its deployment is discussed.

2.7.2 OpenNebula

OpenNebula was initially created as a research project in 2005 by Ignacio M. Llorente and Rubén S. Montero from *Universidad Complutense Madrid*, being publicly released in 2008. It now works as an open source project after having evolved through several releases (now on version 3.4). It is the result of many years of research and development in efficient and scalable management of virtual machines on large-scale distributed infrastructures in close collaboration with *OpenNebula*'s user community and leading experts in cloud computing.

Most of *OpenNebula*'s features have been developed as a response to the use cases from many of the companies involved in the project (these include RESERVOIR [15], *StratusLab* [16] and *4CaaSt* [17]) and its technology has evolved mostly thanks to the effort the community has put into it. [Ope]

It was first released as a software package in *Ubuntu* 9.04, has its own command-line tools and gives the user different configuration scripts which enable a simple and flexible way to design and manage running virtual machines. Since the release of version 3.0, *OpenNebula* has introduced a GUI called *Sunstone* (only runs on *Firefox* and *Chrome* browsers), which allow the users and administrators to manage all *OpenNebula*'s resources (as long as they have access to them, something that can be regulated via ACLs or external modules) [Pin10].

OpenNebula's architecture is presented in Figure 2.3 and its main components are presented in Figure 2.4.

- **Interfaces and APIs** — *OpenNebula* offers two main ways to manage its instances: CLI or GUI (*Sunstone*). Several cloud interfaces such as *OCCI* [18] and EC2 Query [19];

[14] Business launched by *Rackspace* that helps other businesses deploy *OpenStack* [Ince]

[15] *Framework* developed to aid both techonology and information specialists in enterprises in creating a cloud with all the coding and architecture specifications needed. [resa]

[16] A project aiming to develop a complete and open-source cloud distribution that allows both grid and non-grid resource centers to offer and exploit an IaaS cloud. It is particularly focused on enhancing distributed computing infrastructures such as the European Grid Infrastructure (EGI) [str]

[17] Project created for the development of an advanced PaaS cloud platform which supports the optimized and elastic hosting of Internet-scale multitier applications, embedding all the necessary features, so that the programming of rich applications is simplified. [Hid]

[18] Open Cloud Computing Interface. Web service that enables the user to launch and manage virtual machines in the *OpenNebula* installation. [OCC]

[19] Web service that enables the use of virtual machines through Amazon's EC2 Query Interface (2.5.2.2)

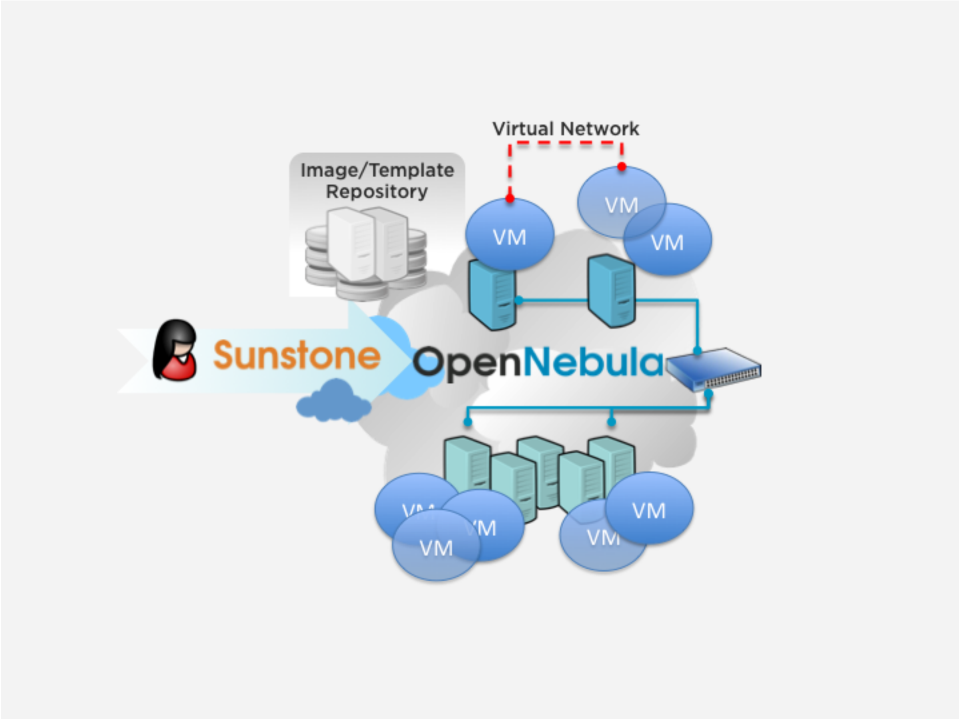


Figure 2.3: OpenNebula's Architecture. [PLa]

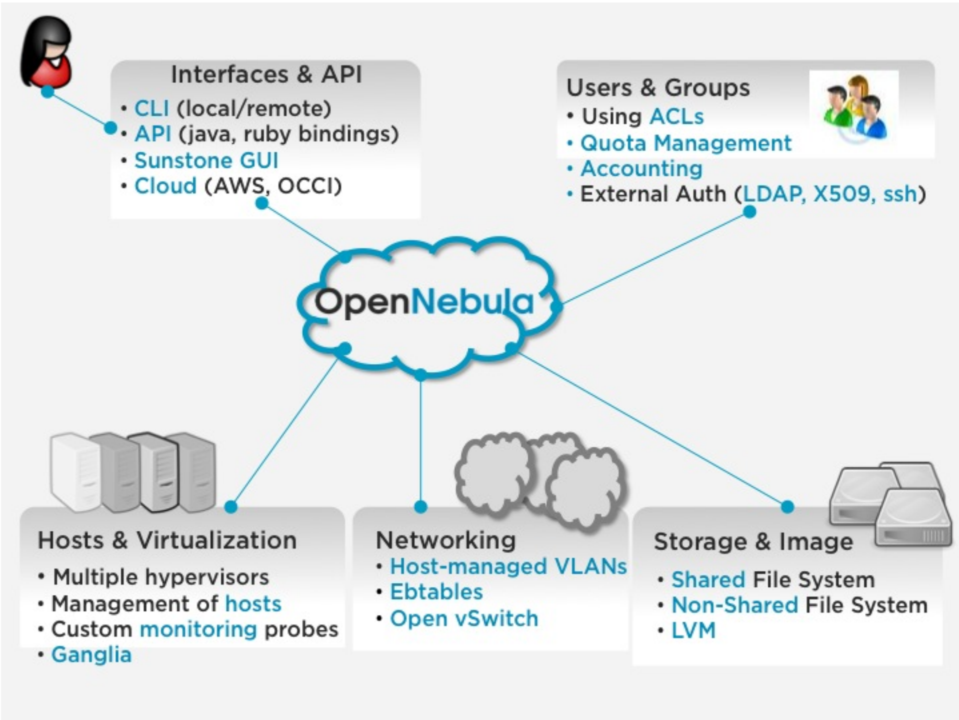


Figure 2.4: OpenNebula's components. [PLb]

- **Users and Groups** — *OpenNebula* supports user accounts and groups, as well as several authentication and authorization mechanisms. These can be used to create isolated compartments inside the same cloud (multi-tenancy). An ACL mechanist also exists to allow different role management;

- **Hosts** — Various hypervisors are supported by the virtualization manager, which has the ability to control and monitor the lifecycle of VMs, something that can be extended to the physical hosts. It is compatible with Xen, KVM and VMware, three *platform virtual machines* that emulate the whole physical computer machine;

- **Networking** — A network subsystem that allows *OpenNebula* to easily integrate with specific network requirements of existing datacenters;

- **Storage** — *OpenNebula* supports multiple data stores in its storage subsystem which provides extreme flexibility in planning the storage backend. Disk images can be stored in both file and block device, also having support for the VMware datastore;

- **Clusters** — Clusters are pools of hosts that share datastores and virtual networks. They are used for load balancing, high availability and high performance computing.

OpenNebula does not have a built-in utility to create VMs from scratch, but its templates allow the VMs to boot an ISO image, leaving the user with just creating an empty hard disk image.

It provides monitoring capabilities which become rather useful when there is a need to troubleshoot, scale or control resource allocation scenarios. *OpenNebula* exports drivers that communicate directly with the hypervisor (KVM - 2.1.1) and return useful data, such as the amount of CPU used, reserved and used memory and network traffic.[CGH09]

In March 2010, *OpenNebula*'s main authors founded *C12G Labs* [20], which has led to it being referred to as "...proprietary tech with an element of openness...", which could limit its growth. [Cor]

2.7.3 Project Aeolus

Aeolus consists of a set of tools to build and manage groups of VMs across clouds (both public and private). It has four components:

- **Conductor** — Provides cloud resources to users, manages their access to and use of those resources, controls user's instances in clouds, launches a VM on a cloud, keeps track of it during its lifecycle and cleans up after the VM is no longer needed. It alerts the user when something happens, such as the VM crashing or the user is about to exhaust the hours of monthly usage, for example. It instructs the user on which cloud to chose depending on certain parameters, including cost, quality of service (QOS) data and other metrics. If the user wishes to, the Conductor can perform that choice;

[20] Company that provides enterprise-grade solutions built around *OpenNebula*

- **Composer** — Builds cloud-specific images from generic templates, in order to allow users to choose clouds freely by using compatible images. The user can specify arguments to pass into the build process so that the software installation can be configured. Images can be rebuilt from the original template in order to update packages or for feature enhancements, bug fixes or security issues.

- **Orchestrator** — Manages groups of instances. Users can automatically bring up a set of different instances on a single cloud or group of clouds, configure them and make them aware of each other;

- **HA Manager** — Makes instances or groups of instances manageable. Provides isolation, recovery and notification of failed applications or instances. Lets the user create policies for maintaining or relaunching services within a specific infrastructure.

2.7.4 Contextualization

At the end of instantiating a VM, its data can be set or overrun by user request, thus creating the possibility of setting up an infinite number of environments using only one VM image.

OpenStack and *OpenNebula* have different ways of handling this process, which are described in this section of the document.

2.7.4.1 *OpenStack*

OpenStack's *Image Service* provides discovery, registration and delivery services for disk and server images, taking advantage of *OpenStack*'s ability to copy or snapshot a server image and store it immediately (2.7.1.1).

Currently only *Ubuntu* and *Amazon* AMI images can be contextualized using *OpenStack*. This happens because *cloud-init* [21] is used [CCa].

Cloud-init handles importing ssh keys for password-less login and setting the host name, amongst other things. The instance acquires the instance specific configuration from Nova-compute by connecting to a meta data interface [CCa].

If an image from a distribution that does not have *cloud-init* is being used, this setup needs to be performed via the use of a set of scripts which should be included in a specific file (rc.local) so they can be executed at boot time.

In Figure 2.5 and Figure 2.6 the *Horizon* dashboard is shown when launching a new instance. Figure 2.6 shows the area where the *cloud-init* configuration file is placed so it can be ran at boot time.

This technology will be used in the solution proposed as it matches perfectly with one of the objectives — being able to create virtual environments according to the users specifications. *Ubuntu* AMI images will be used as concept proof and to simplify the whole process.

[21] *Ubuntu* package that handles early initialization of a cloud instance. It is installed in the *Ubuntu Enterprise Cloud* (UEC) images and also in the official Ubuntu images available on *Amazon*'s EC2.

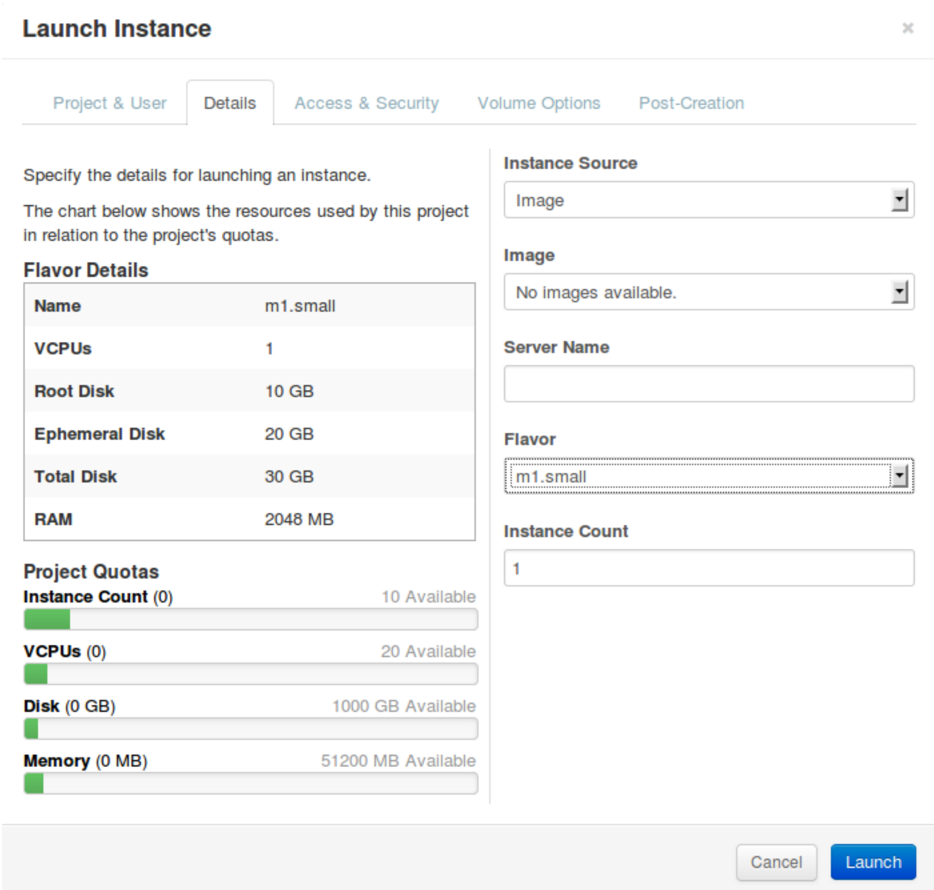


Figure 2.5: Launching an instance in *Horizon*.

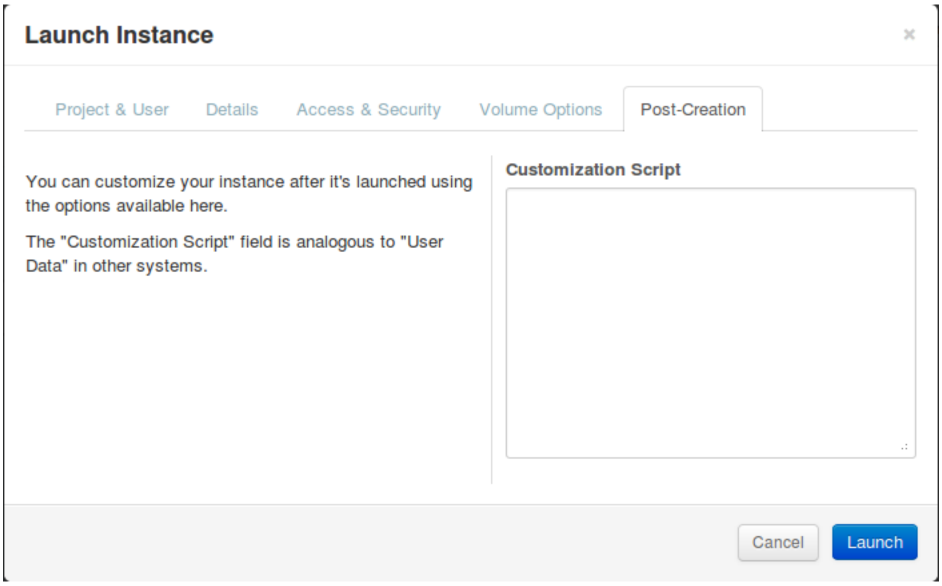


Figure 2.6: Script to be ran once the instance is launched.

It is important to mention that information on *cloud-init* was obtained after going on *OpenStack*'s *Internet Relay Chat* (IRC) channel in the Freenode IRC network [22] and talking directly to the main contributers of the *cloud-init* module, Joshua Harlow and Scott Moser.

According to both of them, *cloud-init* is in the process of being available to a wider selection of distributions, which include *openSUSE*, *Fedora* and *Red Hat*.

The conversation is attached in Appendix C.

[22]#openstack on irc.freenode.net

2.7.4.2 *OpenNebula*

As mentioned in section 2.7.2, *OpenNebula*'s *Storage System* allows administrators and users to set up images (OS or data) to be be used in VMs.

OpenNebula supports three types of images:

- **OS** — Contains a working operative system;
- **CDROM** — Readonly data;
- **DATABLOCK** — Storage for data, which can be accessed and modificed from different VMs. They can be created from previous existing data, or as an empty drive.

The type of an existing image can be changed when performed a specific command and the images can be managed either by using *Sunstone* or *OpenNebula*'s CLI.

In order to create an **OS Image**, a contextualized VM needs to be created and its disk extracted. *OpenNebula*- has two contextualization mechanisms available:

- **Automatic IP assignment** — Several pre-created scripts are provided by *OpenNebula* for *Debian*, *Ubuntu*, *CentOS* and *openSUSE* based systems, all of which can be adapted for other distributions.
- **Generic Contextualization** — Configuration parameters are given to a newly started VM by using an ISO image. The VM description file contains the contents of the ISO file (files and directories), instructs the device that the ISO image will become accessible and specifies the configuration parameters that will be written to a file for later use inside the VM.

When using the generic contextualization mechanism, the VM description file can be used to create a contextualization image, which will contain the context values. These include the *hostname*, *root* password and the Domain Name Server (DNS). These values will be held inside the CONTEXT parameter residing inside the contextualization image, whose variables can be specified in three different ways:

- Hardcoded;
- Using template variables;
- Pre-defined variables. [PLc]

2.8 Image creation

One of the big objectives in this project is the automatic creation of virtual environments. As such, it is necessary to present some of the tools used for this process, having in mind the restrictions enforced by the choices made regarding the rest of the technologies which will be used in the rest of the project, namely *OpenStack* and its restriction regarding the use of *cloud-init* to contextualize the virtual images.

Considering these restrictions, it was chosen to follow the software recommendations made by *OpenStack*'s documentation when building virtual images:

- **Oz** — Part of the *Aeolus* project mentioned in section 2.7, it is a command line tool used by *Rackspace Cloudbuilders* to images for Linux distributions;
- **VMBuilder** — *Python* based software package for creating VM images of free software GNU/Linux-based OS. It supports *Xen*, *VirtualBox*, *VMware*, *KVM* and *Amazon EC2* [dt];
- **VeeWee - Vagrant** — *VeeWee* being the tool used to easily build *Vagrant*-based boxes or *KVM, Virtual Box* and *Fusion* images [Incd, Inca].

These technologies will be reviewed for further use in the project.

2.8.1 Oz

As mentioned earlier, *Oz* is part of a bigger project called *Aeolus* [23].

It was created in order to simplify the automaticinstallation of guest OS, akways using the native OS tools to do the installs. This is done so that when when the installation finishes, the disk image left by *Oz* is exactly the same as if was used an installation CD.

There are three funcionalities available in *Oz*:

- Install the OS — *Oz* was built with enough knowledge to install minimal operating systems with little input. It just needs to be told which OS to install and where the installation media is, doing the rest automatically;
- Customize the OS — This is the relevant part for the project. *Oz* has the ability to install additional packages and files into the OS;
- Generate metadata on that OS — This includes package manifest. This metadata is represented in an XML file denominated ICICLE.

In the OS customization process (which is always done as a separate step from the OS install in order to reduce the chances of failure), *Oz* is able to run the native tools, such as *Yum* and *apt-get*; modify the OS disk image in order to allow remote access, start up the OS in a controlled *KVM* guest; run remote commands (for example: *ssh*) to install packages and files and shut down the OS. It then undoes the changes to the disk and generates the metadata on that OS [HI].

[23] http://aeolusproject.org/index.html

2.8.2 JeOS and vmbuilder

Ubuntu JeOS (pronounced "juice") is a variant of the *Ubuntu* Server OS, which is configured specifically for virtual appliances. It is no longer available as a CD-ROM ISO for download, but can be built using *Ubuntu*'s `vmbuilder`.

JeOS is a specialized installation of *Ubuntu Server Editio* with a tuned kernel that only contains the base elements needed to run in a virtualized environment and as such, suitable for the project covered by this document. The tuning is done in order to take advantage of key performance technologies in the virtualization products from *VMware*, creating a combination of reduced size and optimized performance which ensures that *JeOS* delivers a highly efficient use of server resources in a large virtual environment.

With only the minimal required packages and without unnecessary drivers, software companies can configure their OS according to their needs. They have the safety of knowing that the updates required will be limited and the users will be able to deploy virtual appliances on top of *JeOS* which will need less maintenance than what would have been needed were they installed on top of a full server.

`vmbuilder` takes away the need of downloading a *JeOS* ISO. It will fetch the various package and build a VM specifically designed for what the users desire. It is a script that automates the process of creating a ready to use *Linux* based VM, currently supporting the *KVM* and *Xen* hypervisors.

Command line options can be used to perform actions such as adding or removing packages, choosing which *Ubuntu* version and mirror and more.

`vmbuilder` was first introduced as a shell script in *Ubuntu* 8.04 LTS, as a hack to help developers test their code in a VM without needing to restart the server from scratch every time they needed. A few of *Ubuntu* administrators noticed the script, improved and adapted it to so many use cases that the author of this script (Soren Hansen) rewrote it as a *Python* script with revamped goals:

- Ability to be reused by other distributions other than *Ubuntu*;
- Use plugin mechanisms for all virtualization interactions so that others can easily add logic for other virtualization environments;
- Provide an easy to maintain web interface as an alternative to the CLI [Ubu].

This technology becomes extremely relevant to the project as it covers the creation of customizable environments.

2.8.3 VeeWee and Vagrant

Vagrant is a free and open-source *Ruby* based project, started by Mitchell Hashimoto and John Bender on January 21st, 2010. With its first release on March 7, 2010, *Vagrant*'s goal is to create a tool to manage all the complex parts of development within a virtual environment without affecting the developer's workflow. Its developers are currently working on getting *Vagrant* working on every major OS platform (Linux, OS X and Windows).

Vagrant's development is not supported by any company, as its developers work on the project on their free time. The only external help they get is via contributions, which can be done via completing *Vagrant*'s documentation, its code (open-source project) or submitting financial donations (the developers prefer other types of contributions) [HJB].

VeeWee is a toll built by Patrick Debois as a way to customize the boxes *Vagrant* creates. This process is simple, as *Vagrant* creates three files, one of which is `definition.rb`, a *Ruby* file which contains the main definition of the template created by *VeeWee*. In this file the user can define settings like the memory and disk size. Another file *VeeWee* creates is `preseed.cfg` which can be modified to configure the actual install process, controlling details including, but not limited to, the partitions and their size and timezone setup [MD].

Besides *Vagrant* boxes, *VeeWee* can be used for:

- Creating *VMware* and *KVM* VMs;
- Interacting (creating, destroying, halting and remote accessing them via the `ssh` command) with those VMs;
- Exporting those VMs.

The customization process relevant to the project by modifying the templates used for creating the images.

2.9 Conlusions

In this chapter the main technologies to consider for the project implementation were presented, as well as some of the key concepts needed for contextualization. Some of the technologies related to the development of the web application were also discussed (*Python, Django, Ruby and Rails*).

Python and *Django* were chosen as the tools to build the web application, since they integrate well with *OpenStack*, which is coded almost fully in *Python* and the *Horizon* dashboard is built in *Django*.

As for the image creation process, it was decided to use `vmbuilder` to build instances of *Ubuntu JeOS* and compare it to the use of *cloud-init* in the contextualization process (*cloud-init* is used on already built images).

Chapter 3

Problem Statement

In this chapter the problem will be described and justified, using references from the bibliographic study presented in Chapter 2.

3.1 Problem Description

Currently, FEUP's computing infrastructures are only accessed by those who have the technical knowledge to interact with the system. These people are technicians whose area of expertise encompasses outsourcing computing resources to perform computing jobs.

If someone from an area unrelated to the computing system wants to perform any operation in it, that someone must contact the said technicians and waste valuable time for both parties cutting through red tape.

Having this in mind, CICA has started developing a project that reduces the amount of knowledge necessary to perform the said computing operations.

This document focuses only on the front-end of the project, the back-end having already been developed by former MIEIC student Nuno Cardoso as part of his Master Thesis. CICA's project is described in greater detail in the following section.

3.2 The project

The project aims at simplifying the whole process and to make FEUP's computing infrastructures more accessible to the academic community, without the users having to spend time learning about the technologies and how the system actually works.

In order to better understand the full scope of the issue, Figure 3.1 shows the full system as it should function, through the means of an hypotetic and yet plausible use case scenario:

Firstly, a researcher of a specific field of study wants to conduct a more complex operation that involves greater computing efforts than his/her home and work computer. As such, the researcher proceeds to enter the designed system through a web page where he/she can:

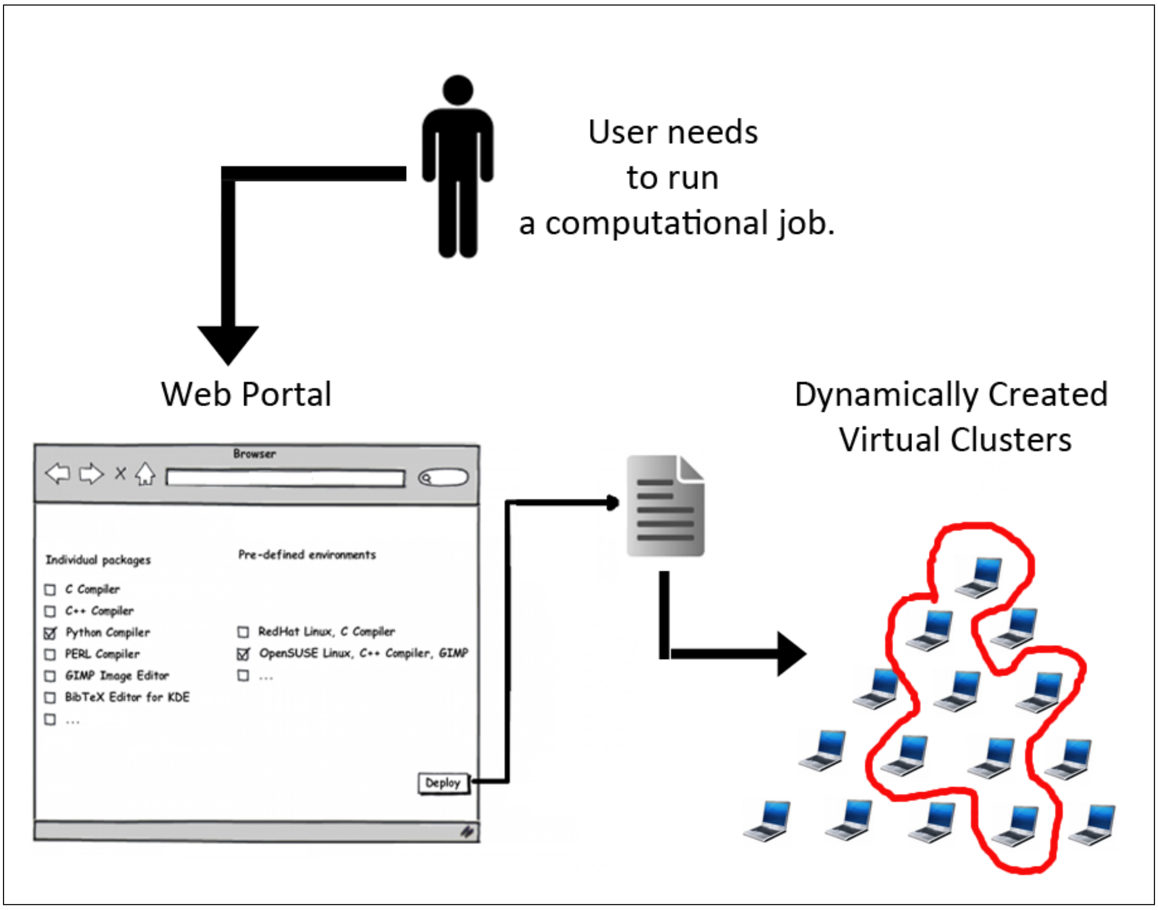


Figure 3.1: CICA's full computing project.

- Chose a suitable work environment for his/her computational needs according to a set of predefined parameters;
- Create his/her own work environment according to the specifications he/she provides the system with.

The system will then automatically create a virtual environment (image containing all the information needed), which will be passed onto the back-end of the project where a virtual cluster will be created according to that virtual environment. Finally a username and password combination should be returned so that the researcher can enter the created environment and perform his/her operations.

The objectives for this project can be divided as following:

1. Implementing the creation of virtual images;
2. Making the creation according to what the users specify — image contextualization;
3. Passing the images created to *OpenStack*. This includes connecting with both *Horizon*(dashboard) and *Glance* (image service).

4. Creating a web system that makes the objectives above transparent for the user.

3.3 The solution

As mentioned in the above section, one of the main objectives of this dissertation is the integration of an *OpenStack* — presented in Chapter 2 — deployment environment as it should simplify the cloud creation and management.

In this section of this chapter the technological choices are presented and justified.

3.3.1 The chosen technologies

One thing was missing in the cloud computing scene... A cloud management layer. A cloud operating system that added automation and control at scale. That is where *OpenStack* comes into play. As mentioned in Chapter 2, section 2.7.1 it is built by a world wide community of developers, something that made it a good choice to investigate, as the open source culture is something always worthy of enriching. [Incb]

There is one thing one must keep in mind: as it was described in Chapter 2, *OpenStack* is not the only solution available. *OpenNebula* was also available and is already up and running at FEUP. So why choose *OpenStack*?

First of all, *OpenStack* is a more recent project and *OpenNebula*. It is backed up by some renowned names in the industry, such as *Dell*, *AMD*, *Intel*, *Canonical*, *Cisco*, *StackOps*, *HP*, *NEC*, *AT & T*, *Yahoo!* and *Red Hat*. Some of these companies also support *OpenNebula*.

The coding activity on both projects was also taken into account when chosing which to deploy. With the help of OHLOH [1], the differences can be easily observed as it is shown in Figure B.1, which is included in Appendix B.

OpenStack has more favourable statistics, such as the number of committers and number of commits (shown in Figures 3.2 and 3.3) over time. If this is viewed with the knowledge that *OpenNebula* was created first, and *OpenStack* managed to outdo it, great things can be expected.

In addition to this and as it was refered in Chapter 2, section 2.7.2, *OpenNebula*'s creators have founded an enterprise of their own (C12G Labs) which offers an enterpreise version of *OpenNebula* (named *OpenNebulaPro*). This deviates from the open source philosophy, something that *OpenStack* maintains.

On a more technical aspect, *OpenStack* is mainly written in *Python* whereas *OpenNebula* is mainly coded in *C++* and *Ruby*, as it can be observed in Figure 3.4.

In order to understand the relevance of this detail, it must be said that there is an extensive and obligatory contact with *C++* in MIEIC, something that does not happen with *Ruby*, and *Python* is not presented at all. Previous experience with both *C++* and *Ruby* proved to be unfulfilling (*C++* due to its not-so-high-level nature and *Ruby* because it was used in the context of *Ruby on Rails*, which due to some installation quirks was deemed impossible to start and use).

[1] An open source directory that anyone can edit. It features comprehensive metrics and analysis on thousands of open source projects. [DSI]

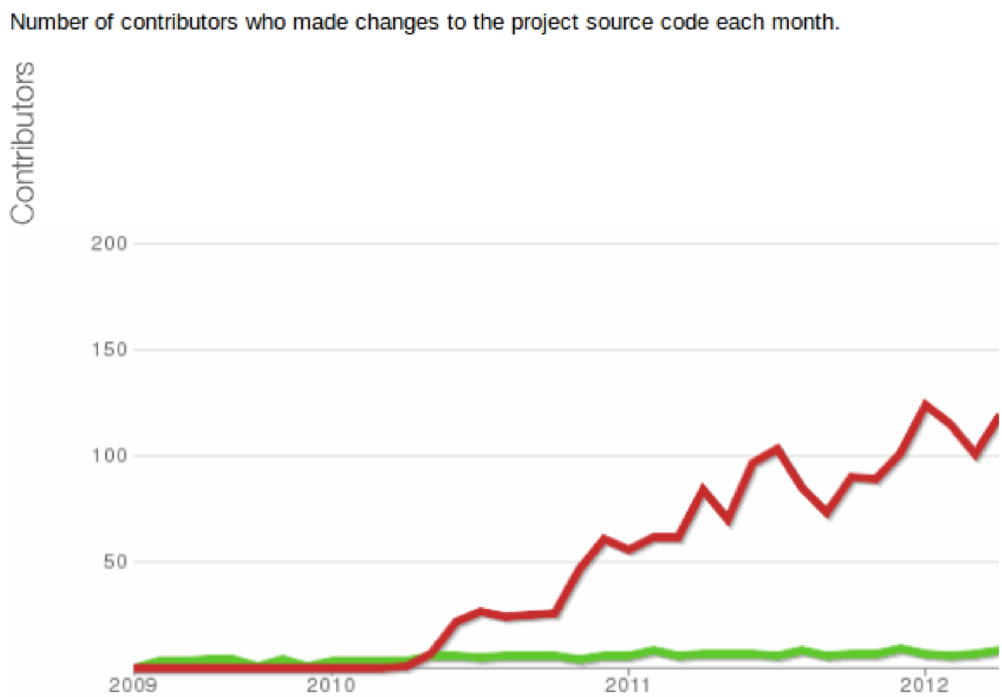


Figure 3.2: Comparison between the number of committers on *OpenStack* and *OpenNebula*. [DSI]

Python on the other hand, was a different programming language, something that was going to be a challenge. Coupled with *Django*, it promised the same advantages as *Ruby on Rails*, but with less trouble getting it up and running. Since *Horizon* is built on *Django*, the choice seemed obvious. The cherry on top of the cake would be contributing to FEUP's knowledge on the new technologies to be researched (*Django*, *Python* and of course, *OpenStack*).

The previous experience with *Ruby* paired with the desire for new challenges and learning new programming languages, *OpenStack* was chosen. This would also allow to contribute to FEUP's knowledge on this new technology.

Rodrigo Benzaquen, director of site operations and infrastructure at MercadoLibre, a Latin America e-commerce market leader which chose to use *OpenStack* as their cloud solution, stated the following:

> "Before this [*OpenStack*'s deployment], we would have had someone physically deploy the server which would take a day or longer. With *OpenStack*, we don't have to do that; our developers are now able to create and manage their servers."[CCc]

which came directly into the objective of this dissertation, easing the cloud creation and management process.

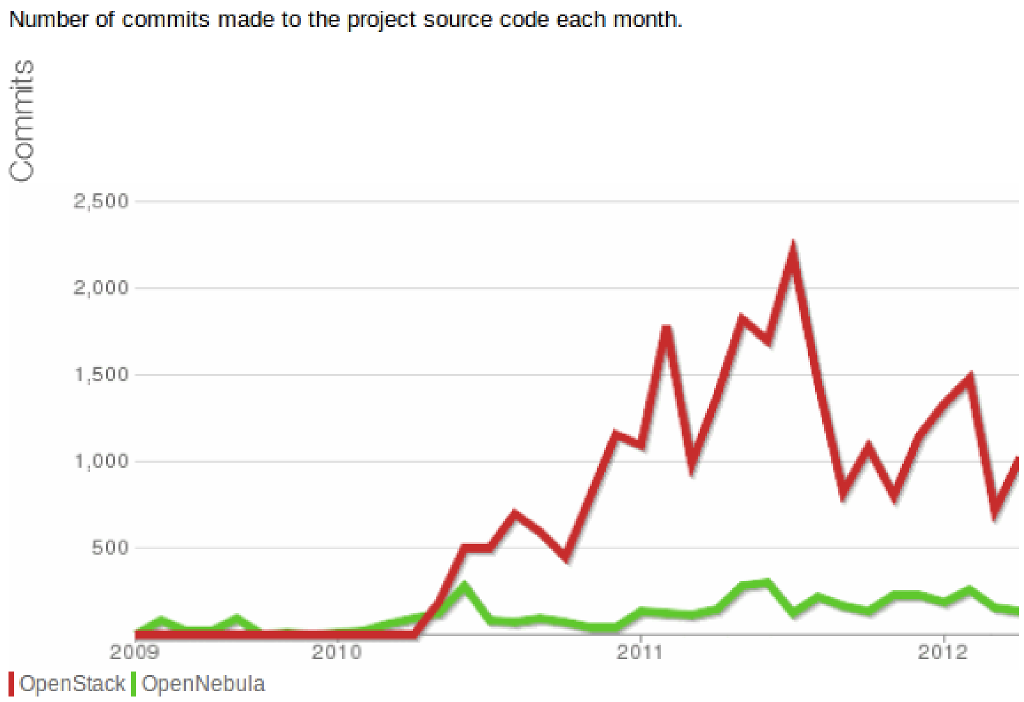


Figure 3.3: Comparison between the number of commits on *OpenStack* and *OpenNebula*. [DSI]

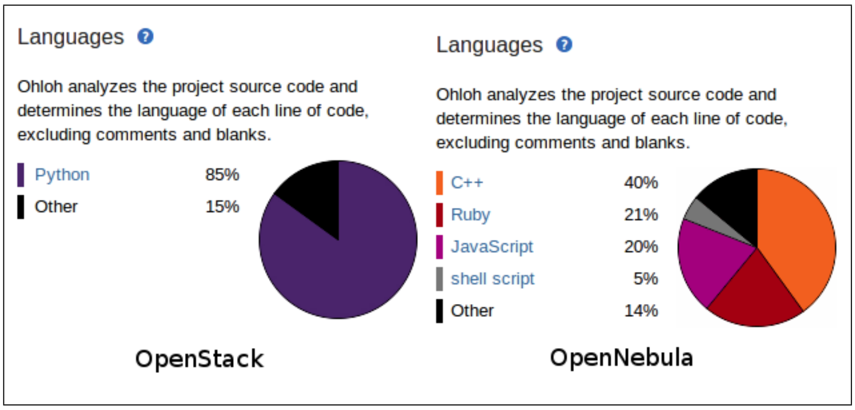


Figure 3.4: Comparison between the programming languages in *OpenStack* and *OpenNebula*. [DSI]

3.3.2 Connecting the dots

As mentioned in Chapter 2, section 2.7.1, *OpenStack* is designed to deliver a massively scalable cloud operating system, each of the components being designed to work together in order to prodive complete IaaS. This integration is facilitated through pulic APIs that each service offers, being available to the cloud's end users. [Pep].

Expanding the diagram shown in Figure 2.2, the relationships between the services are shown in Figure 3.5:

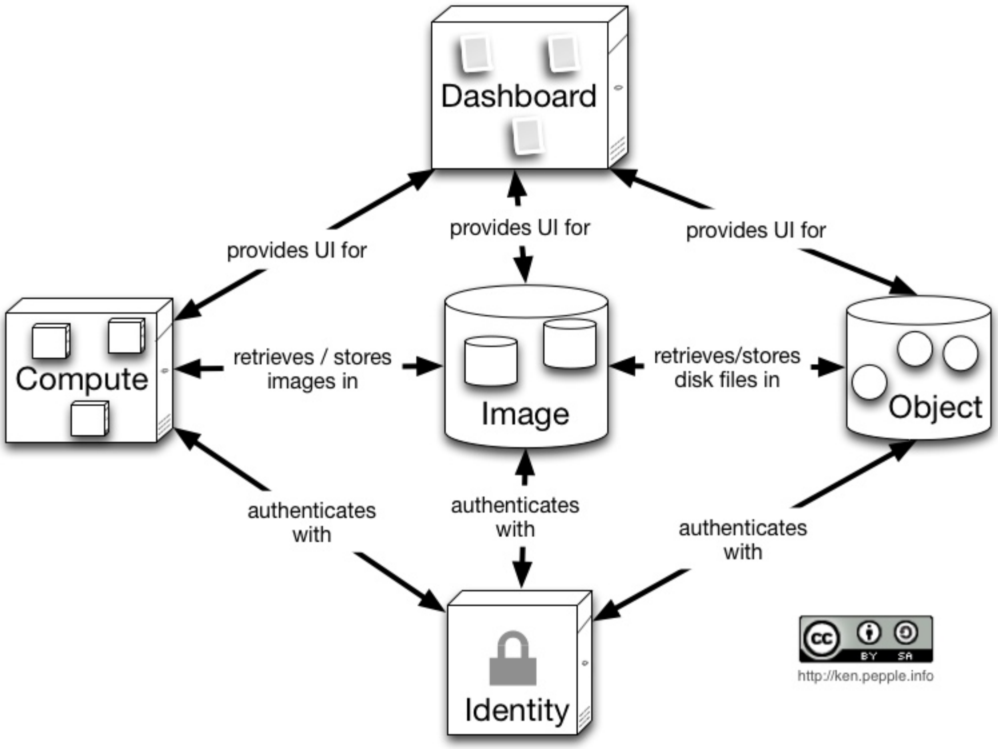


Figure 3.5: Relationships between the different *OpenStack* services. [Pep]

The solution proposed for this project links the *OpenStack* Dashboard — *Horizon* — with the designed *Web application* developed in *Python* and *Django*, as shown in Figure 3.6.

As it can be observed, the web application will use `vmbuilder` and *cloudinit* whenever needed and then passing that information to the *OpenStack Horizon* dashboard, which will communicate with the rest of *OpenStack* services.

Since `vmbuilder` and *cloudinit* work for different purposes (`vmbuilder` creates contextualized VM images and *cloudinit* contextualizes clean VM images), different tools will be used for different purposes.

An interesting feature to complete in future work could be eliminating the web system and passing the image creation and contextualization to *OpenStack*, modifying the *Horizon* dashboard itself.

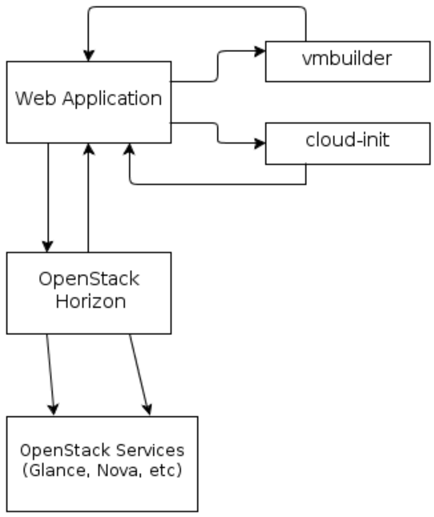


Figure 3.6: Proposed architecture implementation.

3.4 Conclusions

In this chapter was presented the architecture to be followed in Chapter 4 in the implementation phase of the project. The objectives were also outlined, as well as which technologies to use.

Problem Statement

Chapter 4

Approach and Results

This chapter presents the main work which derived from the study performed in Chapter 2. This includes the use cases that originated from having a web application that will be interacted with by the users.

4.1 Use Cases

As this project depicts the creation of a web application that users need to interact with, it is necessary to describe the possible scenarios that can arise, via use case diagrams.

4.1.1 Search the existing VM images

Actors:

- Researcher.

Use Case description: After the web application is accessed, the researcher will need to choose the appropriate option displayed — "Search for an image.". The researcher will then be shown a search form, where he/she will need to input the search terms deemed fit for the researcher's needs.

The web application will connect with *OpenStack's* image service (*Glance*), which will search for the terms inputted by the researcher. If one or more VM images are found according to what was inputted, the search term and a list of VM images are presented. The items in the list are clickable, which mean they link to the details of the VM images displayed. If no images are found, a specific text is displayed warning the researcher.

This use case is shown in Figure 4.1.

4.1.2 View all the available VM images

Actors:

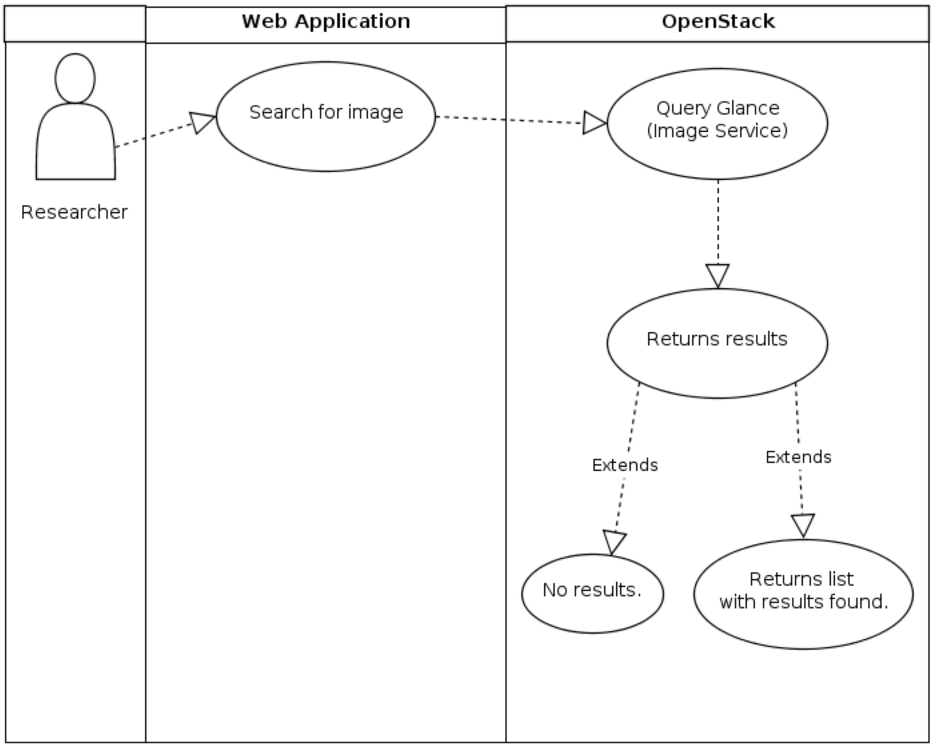


Figure 4.1: Use Case 1: Search the existing VM images.

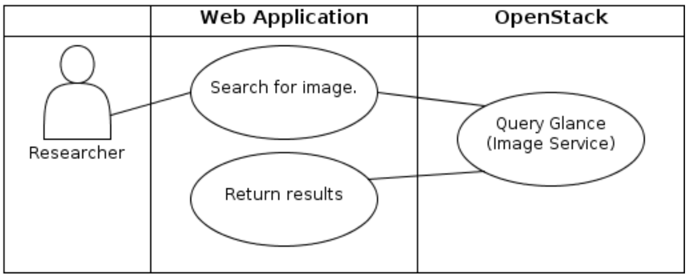


Figure 4.2: Use Case 2: View all the available VM images.

- Researcher.

Use Case description: After the web application is accessed, the researcher will need to choose the appropriate option displayed — “View available images, details and statistics”. The researcher will then be shown a list of all the available images in the system.

This use case is shown in Figure 4.2.

4.1.3 Create a VM image from scratch

Actors:

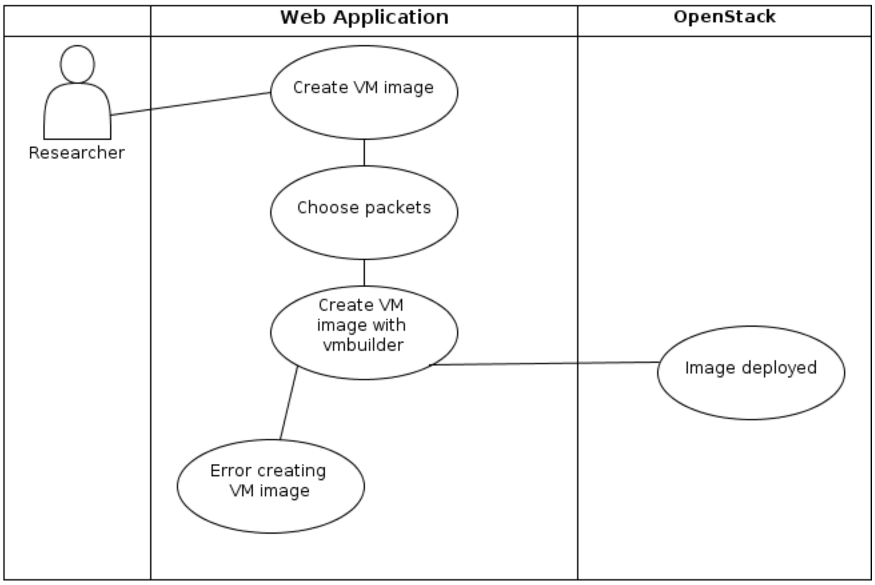


Figure 4.3: Use Case 3: Create a VM from scratch.

- Researcher.

Use Case description: After the web application is accessed, the researcher will need to choose the appropriate option displayed — “Create an image suiting your needs (advanced)”. The researcher will then be shown a list of packages which will have a check box next to them. If the researcher wishes to include one package in the VM image to be created, he/she should click the check box next to the package name.

This use case is shown in Figure 4.3.

A feature for including the packages outside the available list is discussed in Section 5.2 - Future Work.

4.1.4 Launching an already existing VM image

Actors:

- Researcher.

Use Case description: After the web application is accessed, the researcher will need to choose the appropriate option displayed — “Use an already existing image.”. The researcher will then be shown a list of all the available VM images, from which he/she will be able to choose one for deployment. The web application will then connect with *Glance*

This use case is shown in Figure 4.4.

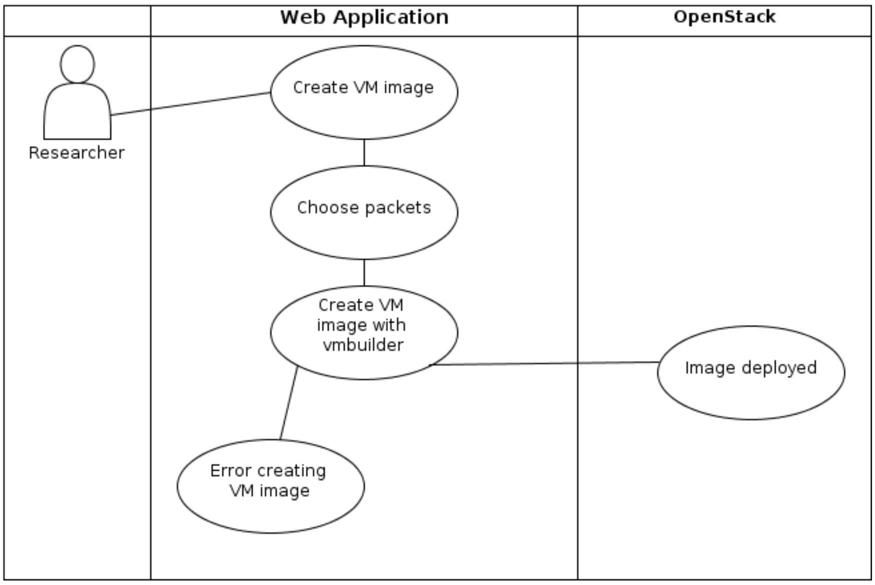


Figure 4.4: Use Case 4: Launch an already existing VM image.

4.2 Implementation

The application of the design and architecture described in Chapter 3 is presented, being divided in three main areas:

- Cloud environment deployment (*OpenStack*);
- Development of a web application;
- Creation and management of VM images (communication between the two parts of the system mentioned in the previous bullet points).

4.2.1 Cloud environment deployment

As mentioned in Chapter 2, section 2.7.1.2, *DevStack* was used in order to simplify the cloud deployment.

Firstly, a clean installation of Ubuntu 12.04 LTS (as recommended by *DevStack*' homepage) was created on Linux's *Virtual Machine Manager* (*libvirt*).

DevStack deployment instructions were followed as they are in its webpage [1] and after the script finished, the *Horizon* Dashboard was accessible via a webpage, as it can be seen in Figure 4.5.

All the desired services were up and running, as shown in Figure 4.6. Even though the image service (*Glance*) was what was needed the most, it showed that the *DevStack* deployment is a viable *OpenStack* development tool.

[1] http://devstack.org

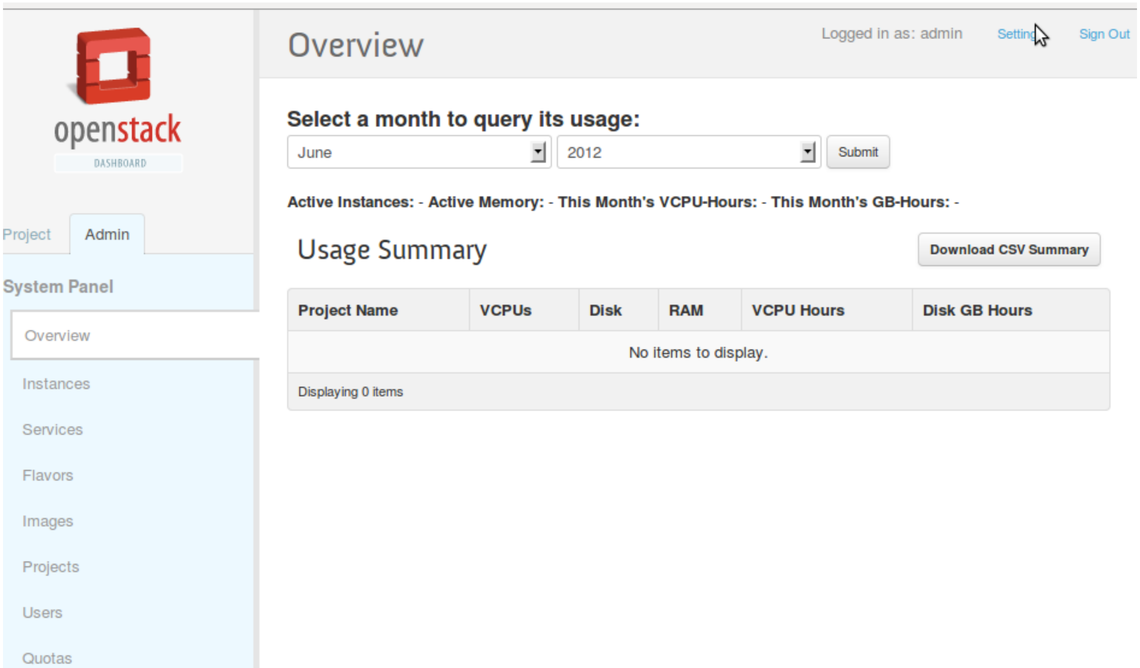


Figure 4.5: *OpenStack Horizon* Dashboard.

4.2.2 Creation and management of VM images

As it was concluded in Chapter 3 (Problem Statement), the VM creation was implemented by using *Ubuntu*'s `vmbuilder`.

A `bash` [2] script was created in order to automate the process. Since it is nothing more than a text file, it is easily modified by the *Python* classes which are called by the web application. The user needs "write" permissions to run that file. An example of the script is attached in Appendix D - vmbuilder script.

This script is run when the user finishes the VM image creating process in the web application.

4.2.3 Web Applicationand VM image creation and contextualization

Since one of the goals of this project was to make submitting computing jobs an easier task, the web application was designed as simple as possible. The actions anyone is able to perform are limited to a bare minimum, while keeping the project's objectives in mind.

The user is only able to execute the following tasks:

- View all VM images available in the system;
- Create a new VM image according to what the user needs — limited to a set or pre-defined packages, due to the fact that packages can have long and sometimes not so obvious names;
- Search for an existing VM in the system;

[2] Unix command shell.

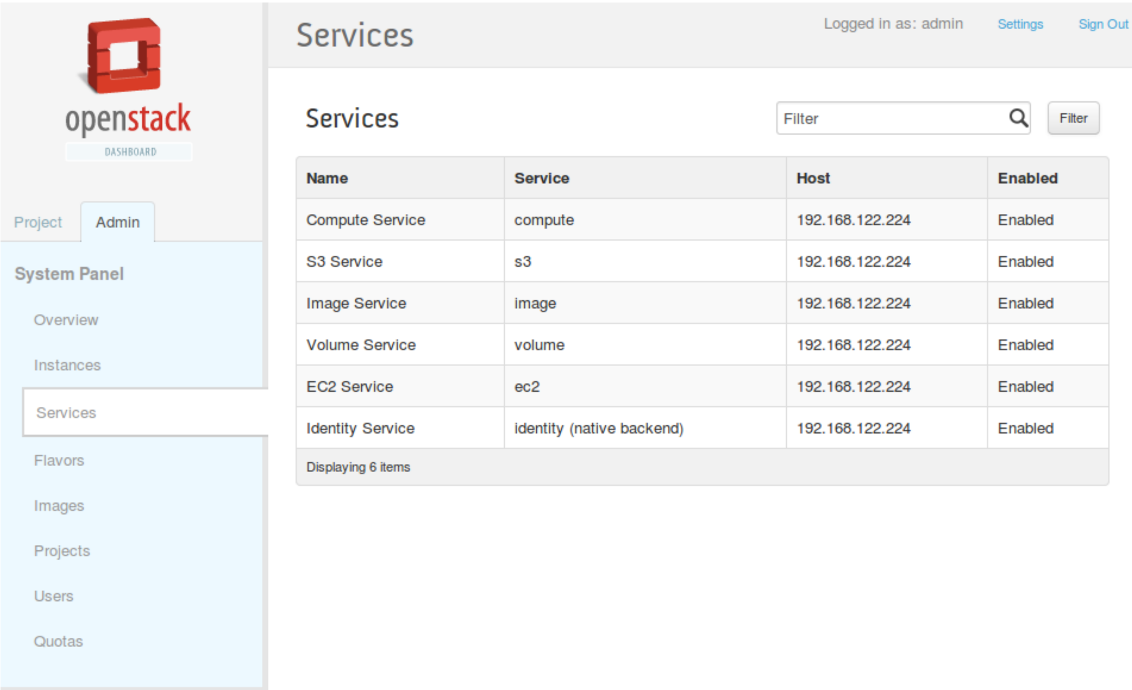


Figure 4.6: *OpenStack* services.

- Use an already existing VM to be deployed.

This was explored in greater detail in the previous section, Use Cases.

The web application is fully developed in *Python* and *Django*. The searching of VM images is also built on *Python* and one of its modules (`python-tagging`) is used. What the system currently does is assign a set of "tags" to the VM images. When the user uses the "search" function, the system is searching for these attacked tags. Should the user input match (partially or fully) any of the tags in the VMs, they will be returned in the final list.

The creation of VM images is made by using `vmbuilder`, which is called by using a `bash` script. This script is modified in real time by using the inputs from the user. The user chooses which packages to include in the VM image creation by selecting the appropriate boxes in the web application.

The script needs to be ran in `sudo` mode due to some restrictions enforced by the use of `vmbuilder`. In order to not compromise the server that runs the web application, the user will **only** be allowed to run this and only this script with `sudo` permitions. To do this the `sudoers` file had to be altered in order to provide passwordless `sudo` access to the files in the scripts folder.

One thing to have in mind is that the VM image creation process is very time consuming. Several tests were ran and the average time passed the twenty minute timeframe, on a small deployment (only `vim` and `openssh-server` were downloaded and installed). One of the script executions can be seen in Figure 4.7.

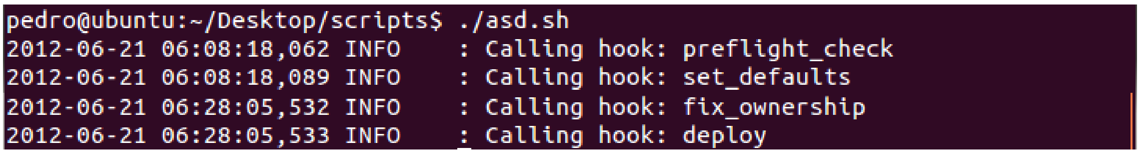


Figure 4.7: One of the VM image creation test runs.

Connection with *Glance* is made through a RESTful API. Communication with *Glance* is established when the user wants to store a newly created VM image, wishes to use an already existing one and when the user wishes to view the images already stored in the server. This last case can be eliminated by storing the previous results on a text file, along with the timestamp of the last change made to that list (which should happen when a user creates and inserts a new VM image into the system).

Since the web application has *Python* code on the background and *OpenStack* is coded in the same programming language, this connection is seamless.

The modules `python-tagging` and `South` are needed for the web application to function properly. `python-tagging`, as it was mentioned earlier, is crucial for the search function, whereas `South` is used in database migrations.

4.3 Conclusions

This chapter presented how the architecture described in the previous chapter (Problem Statement) was implemented, as well as what were the use cases, the challenges encountered and if and how they were overcome. As for the goal completion, *OpenStack* was fully deployed, as well as the web application. VM image creation was successful, as well as the contextualization process.

Chapter 5

Conclusion

The problem described in Chapter 3 has been solved as described in Chapter 4

5.1 Conclusions

During the realization of this project a few conclusions were drawn.

First of all, cloud computing is one major topic in this scene at the moment. Researchers are very fond of the possibilities that cloud computing has and what it has to offer and they want to take as much out of it as they can.

Companies are also shifting towards the cloud. They are setting their IT foundations in it, abandoning the physical infrastructures and costs that come with them. More and more enterprises offer cloud deployment services and the open source community

OpenStack is a major competitor in the cloud computing scene. If it continues with the same pace it had (and maintained) since its first release, it will overthrow its competitors. It is extremely powerful and has both the community and industry backup. *OpenNebula* is also trying to keep up the pace, using the revenue from their enterprise edition to fuel the development of the project.

Furthermore, the creation of VM images on-the-fly is only worth it when there are few to no usable VMs present in the system. As time passes, almost every scenario will be covered and the need for VM image creation will decrease. It may be profitable to setup a VM image repository so that others can benefit from the VM images created by this project, but further research is needed in this subject.

In addition, the *Django* framework came to be a powerful tool in this project. Its MVC architecture simplified the development process and *Python* is an easy to pickup programming language with a very strong community behind it. The downside was that even though *Python* is relatively simple, *Django* can become troublesome in some areas. If someone is not used to this kind of *frameworks*, they will have a somewhat slow learning process.

It is easy to be stuck in the same step for quite some time and not understanding why something is not working, as even though the MVC architecture helps in separating the user interface from the rest of the code, understanding the connections between the views and the controllers can be hard. Most of the issues encountered revolved around the same problem: defining which regular expressions to use for the URL recognition (*Django* uses regular expressions so automatically identify which view to use according to the URL in the browser). Luckily *Django* (similarly to what was described in Contextualization with *cloud-init*) has its own IRC channel in the Freenode IRC network [1] and the users in there were able to solve most of the issues encountered.

Django and *Python*'s full potential is only unleashed after the editor used by the developer is fully optimized in terms of *plugins*. Due to the great number of functions needed to be used (most of them follow the same schema), use of a template language to code the views (which leads once more to a great repetition in the code process), having the appropriate *plugins* to reduce the amount this repetition can reduce the coding time by a huge amount. *Plugins* like `Snippets` for *gedit* (the text editor used for this project) improved the coding time after they were installed.

5.2 Future Work

One of the main improvements that can be done is the full integration with *OpenStack*'s dashboard, instead of relying on a middleman (the web application).

Another improvement would be the possibility of adding new packages to the VM images configuration by user input, instead of choosing them on a fixed list, since this limits what the user can choose from. Removing this limitation can potentially allow the project to scale outside of FEUP's range and open the possibility of deployment on other facilities. This could be achieved by possibly searching repositories in real time, so that the user can have as many choices as possible.

The direct comparison with *OpenNebula* would be a great contribution as well. Comparing VM creation and contextualization time, even comparing the development process by using *Ruby on Rails* would lead to discover which process is more appropriate and benefitial.

[1] irc.freenode.net

Appendix A

2 Grids VS. Clouds

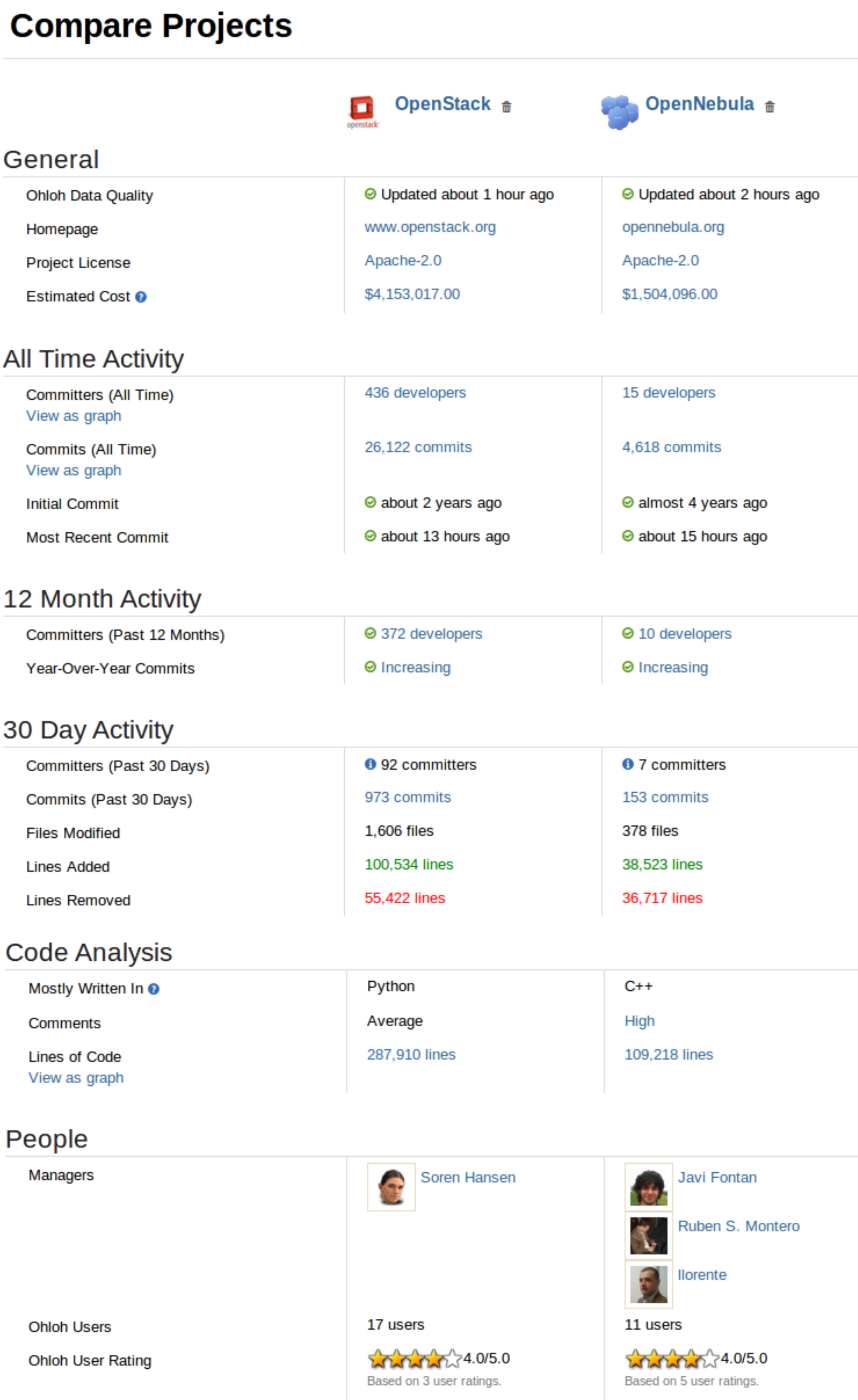
Feature	**Grid**	**Cloud**
Resource Sharing	Collaboration (VOs, fair share).	Assigned resources are not shared.
Resource Heterogeneity	Aggregation of heterogeneous resources.	Aggregation of heterogeneous resources.
Virtualization	Virtualization of data and computing resources.	Virtualization of hardware and software platforms.
Security	Security through credential delegations.	Security through isolation.
High Level Services	Plenty of high level services.	No high level services defined yet.
Architecture	Service orientated.	User chosen architecture.
Software Dependencies	Application domain-dependent software.	Application domain-independent software.
Platform Awareness	The client software must be Grid-enabled.	The SP software works on a customized environment.
Software Workflow	Applications require a predefined workflow of services.	Workflow is not essential for most applications.
Scalability	Nodes and sites scalability.	Nodes, sites, and hardware scalability.
Self-Management	Reconfigurability.	Reconfigurability, self-healing.
Centralization Degree	Decentralized control.	Centralized control (until now).
Usability	Hard to manage.	User friendliness.
Standardization	Standardization and interoperability.	Lack of standards for Clouds interoperability.
User Access	Access transparency for the end user.	Access transparency for the end user.
Payment Model	Rigid.	Flexible.
QoS Guarantees	Limited support, often best-effort only.	Limited support, focused on availability and uptime.

Figure A.1: Comparing Grids and Clouds [VRMCL08].

Appendix B

2 OpenStack VS. OpenNebula

OpenStack VS. OpenNebula



Compare Projects

	OpenStack	OpenNebula
General		
Ohloh Data Quality	Updated about 1 hour ago	Updated about 2 hours ago
Homepage	www.openstack.org	opennebula.org
Project License	Apache-2.0	Apache-2.0
Estimated Cost	$4,153,017.00	$1,504,096.00
All Time Activity		
Committers (All Time) View as graph	436 developers	15 developers
Commits (All Time) View as graph	26,122 commits	4,618 commits
Initial Commit	about 2 years ago	almost 4 years ago
Most Recent Commit	about 13 hours ago	about 15 hours ago
12 Month Activity		
Committers (Past 12 Months)	372 developers	10 developers
Year-Over-Year Commits	Increasing	Increasing
30 Day Activity		
Committers (Past 30 Days)	92 committers	7 committers
Commits (Past 30 Days)	973 commits	153 commits
Files Modified	1,606 files	378 files
Lines Added	100,534 lines	38,523 lines
Lines Removed	55,422 lines	36,717 lines
Code Analysis		
Mostly Written In	Python	C++
Comments	Average	High
Lines of Code View as graph	287,910 lines	109,218 lines
People		
Managers	Soren Hansen	Javi Fontan Ruben S. Montero llorente
Ohloh Users	17 users	11 users
Ohloh User Rating	4.0/5.0 Based on 3 user ratings.	4.0/5.0 Based on 5 user ratings.

Figure B.1: Comparing *OpenStack* and *OpenNebula* on *Ohloh.com*.[DSI]

Appendix C

IRC conversation about *Cloud-init*

Below is presented the IRC conversation on *cloud-init*. Some parts have been left out so that what is important can be understood.

Joshua Harlow uses the alias “harlowja”, Scott Moser uses the alias “smoser” and the author of this report uses the alias “pteixeira”.

[03:07] <pteixeira> the way on how different instances from the same images are configured (ip, authentication, etc etc etc)

[03:08] <zaitcev> So, it’s like what Audrey does?

[03:08] <pteixeira> audrey?

[03:09] <zaitcev> Actually, I think the fashionable tool these days is cloud-init. Audrey was originally put together by Aeolus people.

[...]

[03:11] <pteixeira> cloud-init only works on ubuntu based images, right?

[...]

[03:12] <smoser> coming soon to an RPM based distro near you.

[03:12] <smoser> (thanks to harlowja and others)

[...]

[03:12] <harlowja> ya, it will be nicer to work with other distros and debugging and such soon

[03:12] <harlowja> that is the hope

[03:12] <smoser> pteixeira, it does exist in fedora at the moment, but in a limited fashion

[...]

[03:13] <smoser> pteixeira, and there is cloud-init in debian sid right now, although there is work to be don there also.

[03:13] <harlowja> ya, don’t expect it to do to much, i am working on something that abstracts away as much of the distro stuff as possible to helper classes, some stuff won’t work in fedora/rh... ie, aptupgrades and such but those can be removed

[03:14] <pteixeira> ill use the ubuntu one for now... are there any links that i can follow so i can get some more richness on the document?

[03:15] <harlowja> code level, or just regular docs, or capability docs?

[03:15] <harlowja> https://help.ubuntu.com/community/CloudInit

[03:15] <harlowja> depends on how deep down the rabbit hole u want to go

[...]

[03:17] <pteixeira> actually, can you link me to the capability docs?

[03:18] <harlowja> the capabilities, are in that main one, http://bazaar.launchpad.net/ cloud-init-dev/cloud-init/trunk/files/head:/doc/examples/ + code unless there is another better place

[03:18] <harlowja> i would almost say look at http://bazaar.launchpad.net/ cloud-init-dev/cloud-init/trunk/files/head:/cloudinit/CloudConfig/ also

[03:18] <harlowja> docs for this kind of stuff could be better i think

[03:19] <harlowja> datasource* modules there are how data gets loaded

[03:19] <harlowja> http://bazaar.launchpad.net/ cloud-init-dev/cloud-init/trunk/files/head:/cloudinit/

[...]

[03:20] <harlowja> http://bazaar.launchpad.net/ harlowja/cloud-init/rework/files might be easier to follow, basically starting in stages.py/init and then to the stages.py/transforms but don't expect that branch to work yet

[03:20] <harlowja> but for refereence it might be better

[03:21] <harlowja> sorry, http://bazaar.launchpad.net/ harlowja/cloud-init/rework/files/head:/cloudinit/, not the main dir

[...]

[03:22] <smoser> almost all of cloudconfig function is documented in the doc/ link or source that harlow pointed at above.

[03:22] <smoser> and the wiki doc shows what all to feed CloudInit (one of the things you can feed it is cloudconfig).

[...]

Appendix D

vmbuilder script

Script used to create VM images dynamically.

This creates a `KVM Ubuntu` image, version *Precise Pangolin* (`-suite=precise`), suitable for virtual environments (`-flavour=virtual`), for 32 bit machine (`-arch=i386`), using the official *Ubuntu* mirrors to get the packages (`-mirror`).

The section `-o -libvirt=qemu:///system` tells the system to register the newly created with the system's virtual machine manager.

In this case, a file named `vmbuilder.partition` was used to define the disk partitioning. The section `-templates=templates` points to the folder where `vmbuilder` should use the templates to build the image.

After this, we have the definition of the user, the name to be used and the password. This password is reset on first boot, as described in the file `boot.sh`.

`-addpkg` tells `vmbuilder` to install the security updates, `vim` and *acpid* (used for functions such as closing a laptop lid, pressing the power button, etc).

Finally, `-mem=256` specifies the total RAM and *–hostname*, defines the machine's hostname.

```
#!/bin/bash
sudo vmbuilder kvm ubuntu --suite=precise --flavour=virtual \
--arch=i386 --mirror=http://de.archive.ubuntu.com/ubuntu \
-o --libvirt=qemu:///system --ip=192.168.0.101 \
--part=vmbuilder.partition --templates=templates \
--user=administrator --name=Administrator --pass=howtoforge \
--addpkg=vim-nox --addpkg=unattended-upgrades \
--addpkg=acpid --firstboot=/home/pedro/Desktop/scripts/boot.sh \
--mem=256 --hostname=vm1
```

vmbuilder script

References

[Ama] Amazon. Amazon EC2 FAQ - what is a “EC2 Compute Unit” and why did you introduce it? http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it. Last accessed 16 July 2011.

[B0́8] Marc-Elian Bégin. An EGEE comparative study: Grids and clouds - evolution or revolution. Technical report, CERN - Engeneering and Equipment Data Management Service, June 2008.

[BAFB05] M. Baker, A. Apon, C. Ferner, and J. Brown. Emerging grid standards. *Computer*, 38(4):43 – 50, april 2005.

[BK07] M. Bachle and P. Kirchberg. Ruby on rails. *Software, IEEE*, 24(6):105 –108, nov.-dec. 2007.

[BLDGS04] Miguel Bote-Lorenzo, Yannis Dimitriadis, and Eduardo Gómez-Sánchez. Grid characteristics and uses: A grid definition. In Francisco Fernández Rivera, Marian Bubak, Andrés Gómez Tato, and Ramón Doallo, editors, *Grid Computing*, volume 2970 of *Lecture Notes in Computer Science*, pages 291–298. Springer Berlin / Heidelberg, 2004.

[BV08] Jaijit Bhattacharya and Sushant Vashistha. Utility computing-based framework for e-governance. In *Proceedings of the 2nd international conference on Theory and practice of electronic governance*, ICEGOV ’08, pages 303–309, New York, NY, USA, 2008. ACM.

[BYV$^+$09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

[Car09] Nicholas Carr. *The Big Switch: Rewiring the World, from Edison to Google*. W.W. Norton & Company, 2009.

[Car11] Nuno Cardoso. Virtual clusters sustained by cloud computing infrastructures. Master’s thesis, FEUP - Faculdade de Engenharia da Universidade do Porto, 2011.

[CCa] Rackspace Cloud Computing. Introduction - Openstack Compute Starter Guide - essex. http://docs.openstack.org/essex/openstack-compute/starter/content/Introduction-d1e1257.html. Last accessed 14 June 2012.

[CCb] Rackspace Cloud Computing. Open Stack - Open Source Cloud Computing Software. http://www.openstack.org/software/. Last accessed 12 June 2012.

[CCc] Rackspace Cloud Computing. User Stories - OpenStack Open Source Cloud Computing Software. http://www.openstack.org/user-stories/. Last accessed 14 June 2012.

[CCd] Inc. Cunningham & Cunningham. Don't Repeat Yourself. http://c2.com/cgi/wiki?DontRepeatYourself. Last accessed 16 July 2011.

[CER] CERN. gLite - Lightweight Middleware for Grid Computing. http://glite.cern.ch/. Last accessed 16 July 2011.

[CGH09] Damien Cerbelaud, Shishir Garg, and Jeremy Huylebroeck. Opening the clouds: qualitative overview of the state-of-the-art open source vm-based cloud management platforms. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '09, pages 22:1–22:8, New York, NY, USA, 2009. Springer-Verlag New York, Inc.

[Cor] LinkedIn Corporation. OpenStack vs Eucalyptus vs OpenNebula. http://www.linkedin.com/groups/OpenStack-vs-Eucalyptus-vs-OpenNebula-2685473.S.54382975. Last accessed 14 June 2012.

[DSI] Black Duck Software, Inc. Compare projects - Ohloh. http://www.ohloh.net/p/compare?metric=Summary&project_0=OpenStack&project_1=OpenNebula&project_2=. Last accessed 14 June 2012.

[dt] Ubuntu documentation team. JeOS and vmbuilder. https://help.ubuntu.com/12.04/serverguide/jeos-and-vmbuilder.html. Last accessed 14 June 2012.

[FFK+06] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang. Virtual clusters for grid communities. *Cluster Computing and the Grid, IEEE International Symposium on*, pages 513–520, 2006.

[Fos02] Ian Foster. What is the grid? a three point checklist. *Grid Today*, 1(6):22–25, 2002.

[GKC+] Jeremy Geelan, Markus Klems, Reuven Cohen, Jeff Kaplan, Douglas Gourlay, Praising Gaw, Damon Edwards, Brian de Haaff, Ben Kepes, Kirill Sheynkman, Omar Sultan, Kevin Hartig, Jan Pritzker, Trevor Doerksen, Thorsten von Eicken, Paul Wallis, Michael Sheehan, Don Dodge, Aaron Ricadela, Bill Martin, Ben Kepes, and Irving W. Berger. Twenty-One Experts Define Cloud Computing. http://virtualization.sys-con.com/node/612375. Last accessed in 16 July 2011.

[GWQF10] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox. Cloud computing paradigms for pleasingly parallel biomedical applications. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 460–469, New York, NY, USA, 2010. ACM.

[Hay08] Brian Hayes. Cloud computing. *Commun. ACM*, 51:9–11, July 2008.

[Haz08] Scott Hazelhurst. Scientific computing using virtual high-performance computing: a case study using the amazon elastic computing cloud. In *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*, SAICSIT '08, pages 94–103, New York, NY, USA, 2008. ACM.

[HH] David Heinemeier Hansson. Ruby on Rails. http://rubyonrails.org/. Last accessed 16 July 2011.

[HI] Red Hat, Inc. Aeolus Project - Oz. http://aeolusproject.org/oz.html. Last accessed 14 June 2012.

[Hid] Arif Hidayat. Morfeo 4CaaSt. http://4caast.morfeo-project.org/. Last accessed 14 June 2012.

[HJB] Mitchell Hashimoto and John Bender. Vagrant - Contribute to Vagrant. http://vagrantup.com/contribute/index.html. Last accessed 14 June 2012.

[HP10] Joel Hollingsworth and David J. Powell. Teaching web programming using the google cloud. In *Proceedings of the 48th Annual Southeast Regional Conference*, ACM SE '10, pages 76:1–76:5, New York, NY, USA, 2010. ACM.

[IDE+06] Alexandru Iosup, Catalin Dumitrescu, Dick Epema, Hui Li, and Lex Wolters. How are real grids used? the analysis of four grid traces and its implications. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, GRID '06, pages 262–269, Washington, DC, USA, 2006. IEEE Computer Society.

[Inca] GitHub, Inc. jedi4ever/veewee. https://github.com/jedi4ever/veewee. Last accessed 14 June 2012.

[Incb] GitHub, Inc. OpenStack Essex Deploy Day · dellcloudedge/crowbar Wiki. https://github.com/dellcloudedge/crowbar/wiki/OpenStack-Essex-Deploy-Day. Last accessed 14 June 2012.

[Incc] QuinStreet Inc. What is Scale-Out Storage? An IT Definition From Webopedia.com. http://www.webopedia.com/TERM/S/scale_out_storage.html. Last accessed 14 June 2012.

[Incd] Rackspace US, Inc. Chapter 1. Getting Started with OpenStack - OpenStack Compute Administration Manual- Essex (2012.1). http://docs.openstack.org/essex/openstack-compute/admin/content/ch_getting-started-with-openstack.html. Last accessed 14 June 2012.

[Ince] Rackspace US, Inc. Open Cloud Deployment in Your Data Center Managed by Rackspace. http://www.rackspace.com/cloud/private_edition/. Last accessed 14 June 2012.

[KF98] Carl Kesselman and Ian Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998.

[KF08] K. Keahey and T. Freeman. Contextualization: Providing one-click virtual clusters. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 301 –308, dec. 2008.

[KG09] Eric Knorr and Galen Gruman. What cloud computing really means. http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031, 2009. Last accessed 16 July 2011.

[McF] Paul McFedries. Ieee spectrum - Inside Technology. http://spectrum.ieee.org/computing/hardware/the-cloud-is-the-computer. Last accessed 16 July 2011.

[MD] Marius Ducea. MDLog:/sysadmin. http://www.ducea.com/2011/08/15/building-vagrant-boxes-with-veewee/. Last accessed 14 June 2012.

[Mic] Microsoft. Windows Azure | Microsoft Paas | Cloud Services | Application Hosting. http://www.microsoft.com/windowsazure/. Last accessed 16 July 2011.

[NASA] NASA North American Space Agency. Nebula Cloud Computing Platform. http://nebula.nasa.gov/. Last accessed 14 June 2012.

[NMM07] Hideo Nishimura, Naoya Maruyama, and Satoshi Matsuoka. Virtual clusters on the fly - fast, scalable, and flexible installation. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:549–556, 2007.

[NS07] Ripal Nathuji and Karsten Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 265–278, New York, NY, USA, 2007. ACM.

[OCC] OCCI. Open Cloud Computing Interface - Open Standard - Open Community. http://occi-wg.org/. Last accessed 14 June 2012.

[Ope] OpenNebula. OpenNebula. http://opennebula.org/about:faq#what_is_opennebula. Last accessed 16 July 2011.

[Pep] Ken Pepple. Revisiting Openstack Architecture: Essex Edition. http://ken.pepple.info/openstack/2012/02/21/revisit-openstack-architecture-diablo/. Last accessed 14 June 2012.

[Pin10] Jorge Fernando Maciel Rodrigues Ruão Pinheiro. Interligação de infra-estruturas de computação de elevado desempenho heterogéneas recorrendo a um super escalonador. Master's thesis, FEUP - Faculdade de Engenharia da Universidade do Porto, 2010.

[PLa] OpenNebula Project Leads. OpenNebula: The Open Source Solution for Data Center Virtualization. http://opennebula.org/about:technology. Last accessed 12 June 2012.

[PLb] OpenNebula Project Leads. OpenNebula: The Open Source Solution for Data Center Virtualization. http://www.opennebula.org/documentation:rel3.4. Last accessed 12 June 2012.

[PLc] OpenNebula Project Leads. OpenNebula: The Open Source Solution for Data Center Virtualization. http://opennebula.org/documentation:rel3.4:img_guide. Last accessed 15 June 2012.

[Rap04] M. A. Rappa. The utility business model and the future of computing services. *IBM Syst. J.*, 43:32–42, January 2004.

[resa] Reservoir fp7 - Home. http://www.reservoir-fp7.eu/. Last accessed 14 June 2012.

[Resb] Cluster Resources. Cluster Resources: Moab Cluster Software Suite. http://www.clusterresources.com/products/moab-cluster-suite.php. Last accessed 16 July 2011.

[RW04] J. W. Ross and G. Westerman. Preparing for utility computing: The role of it architecture and relationship management. *IBM Syst. J.*, 43:5–19, January 2004.

[sei] stack exchange inc. Can I use Python as a bash replacement? http://stackoverflow.com/questions/209470/can-i-use-python-as-a-bash-replacement. Last accessed 16 July 2011.

[Sto07] Heinz Stockinger. Defining the grid: a snapshot on the current view. *The Journal of Supercomputing*, 42:3–17, 2007.

[str] StratusLab Combining Grid and Cloud Technologies. stratuslab.eu. Last accessed 14 June 2012.

[Sun] Karishma Sundaram. Bright Hub - The Hub for Bright Minds. http://www.brighthub.com/environment/green-computing/articles/68785.aspx. Last accessed 16 July 2011.

[Tea] Condor Team. Condor Project Homepage. http://www.cs.wisc.edu/condor/. Last accessed 16 July 2011.

[Tec] TechTarget. What is high-performance computing (HPC)? — Definition from WhatIs.com. http://searchenterpriselinux.techtarget.com/definition/high-performance-computing. Last accessed 14 June 2012.

[Ubu] Ubuntu. JeOS and vmbuilder. https://help.ubuntu.com/11.04/serverguide/jeos-and-vmbuilder.html. Last accessed 16 July 2011.

[VRMCL08] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39:50–55, December 2008.

[Wei07] Aaron Weiss. Computing in the clouds. *netWorker*, 11:16–25, December 2007.

[ZKFF05] Xuehai Zhang, Katarzyna Keahey, Ian Foster, and Timothy Freeman. Virtual cluster workspaces for grid applications. Technical report, 2005.