Instituto Superior de Engenharia do Porto
Departamento de Engenharia Eletrotécnica
Rua Dr. António Bernardino de Almeida, 431, 4200-072 Porto

Integrated Management of Cloud Computing Resources

Dissertation submitted to the Instituto Superior de Engenharia do Porto in partial fulfilment of the requirements for the degree of Master of Science in Electrical and Computers Engineering - Major in Telecommunications

Fernando Miguel Dias Meireles

Supervisor: Prof. Maria Benedita Campos Neves Malheiro
Co-supervisor: Prof. Pedro João De-Francesco Assis

Academic Year: 2013-2014

To my family.

Abstract

Lunacloud is a cloud service provider with offices in Portugal, Spain, France and UK that focus on delivering reliable, elastic and low cost cloud Infrastructure as a Service (IaaS) solutions. The company currently relies on a proprietary IaaS platform - the Parallels Automation for Cloud Infrastructure (PACI) - and wishes to expand and integrate other IaaS solutions seamlessly, namely open source solutions. This is the challenge addressed in this thesis. This proposal, which was fostered by Eurocloud Portugal Association, contributes to the promotion of interoperability and standardisation in Cloud Computing.

The goal is to investigate, propose and develop an interoperable open source solution with standard interfaces for the integrated management of IaaS Cloud Computing resources based on new as well as existing abstraction libraries or frameworks. The solution should provide both Web and application programming interfaces.

The research conducted consisted of two surveys covering existing open source IaaS platforms and PACI (features and API) and open source IaaS abstraction solutions. The first study was focussed on the characteristics of most popular open source IaaS platforms, namely OpenNebula, OpenStack, CloudStack and Eucalyptus, as well as PACI and included a thorough inventory of the provided Application Programming Interfaces (API), *i.e.*, offered operations, followed by a comparison of these platforms in order to establish their similarities and dissimilarities. The second study on existing open source interoperability solutions included the analysis of existing abstraction libraries and frameworks and their comparison.

The approach proposed and adopted, which was supported on the conclusions of the carried surveys, reuses an existing open source abstraction solution - the Apache Deltacloud framework. Deltacloud relies on the development of software driver modules to interface with different IaaS platforms, officially provides and supports drivers to sixteen IaaS platform, including OpenNebula and OpenStack, and allows the development of new provider drivers. The latter functionality

was used to develop a new Deltacloud driver for PACI. Furthermore, Deltacloud provides a Web dashboard and REpresentational State Transfer (REST) API interfaces.

To evaluate the adopted solution, a test bed integrating OpenNebula, OpenStack and PACI nodes was assembled and deployed. The tests conducted involved time elapsed and data payload measurements via the Deltacloud framework as well as via the pre-existing IaaS platform API. The Deltacloud framework behaved as expected, *i.e.*, introduced additional delays, but no substantial overheads. Both the Web and the REST interfaces were tested and showed identical measurements.

The developed interoperable solution for the seamless integration and provision of IaaS resources from PACI, OpenNebula and OpenStack IaaS platforms fulfils the specified requirements, *i.e.*, provides Lunacloud with the ability to expand the range of adopted IaaS platforms and offers a Web dashboard and REST API for the integrated management. The contributions of this work include the surveys and comparisons made, the selection of the abstraction framework and, last, but not the least, the PACI driver developed.

Resumo

A Lunacloud é uma empresa provedora de serviços de computação em nuvem com escritórios em Portugal, Espanha, França e Reino Unido e focada no provisionamento de serviços ao nível da infraestrutura (IaaS). Atualmente, esta empresa depende de uma plataforma de IaaS proprietária – Parallels Automation for Cloud Infrastructure (PACI) – e pretende expandir a oferta de serviços através da integração de novas plataformas de IaaS, nomeadamente do tipo *open-source*. Este é o desafio proposto para a presente tese. Esta proposta encontra-se enquadrada nos esforços de promoção da normalização e interoperabilidade da Associação EuroCloud Portugal.

O objetivo do trabalho consiste em investigar, propor e desenvolver uma solução interoperável, baseada na utilização de ferramentas ou bibliotecas de abstracção *open-source* existentes ou a desenvolver, que apresente interfaces normalizadas para a gestão integrada de recursos de computação em nuvem ao nível da infraestrutura. A solução desenvolvida deverá incluir interfaces *Web* e de programação de aplicações.

O trabalho de investigação realizado foi composto por dois estudos referentes à caracterização e análise das interfaces das plataformas de IaaS (plataforma proprietária PACI e plataformas *open-source*) selecionadas e soluções *open-source* de abstracção de plataformas de IaaS existentes. O primeiro estudo caracterizou as plataformas de IaaS *open-source* mais conhecidas, nomeadamente, OpenNebula, OpenStack, CloudStack e Eucalyptus, assim como a plataforma de IaaS proprietária (usada pela Lunacloud) PACI. Foi também incluído neste estudo um inventário extensivo e comparação das respetivas bibliotecas de interface de forma a identificar similaridades e diferenças. O segundo estudo contemplou as soluções de interoperabilidade *open-source* existentes, incluindo a análise e comparação das ferramentas e bibliotecas de abstracção.

A abordagem escolhida e adoptada, que se suportou nas conclusões dos estudos realizados, reutiliza uma solução de abstracção já existente – o *framework* de abstracção Deltacloud. Esta ferramenta recorre a módulos de *software* – *drivers* –

para interagir com as diferentes plataformas de IaaS, fornece oficialmente *drivers* para dezasseis plataformas de IaaS, incluindo as plataformas OpenNebula e OpenStack, e permite o desenvolvimento de novos *drivers* para a integração de novos fornecedores/plataformas de IaaS. Esta última funcionalidade foi utilizada para criar um novo *driver* para a plataforma de IaaS proprietária PACI. Para além das funcionalidades referidas, a ferramenta Deltacloud, também fornece um *Web dashboard* e bibliotecas de interface *REpresentational State Transfer* (REST).

De forma a avaliar a solução desenvolvida, foi montada uma plataforma de testes integrando as plataformas OpenNebula, OpenStack, CloudStack e PACI. Os testes realizados consistiram na medição dos tempos de resposta e da quantidade de informação trocada durante a invocação das operações através da ferramenta Deltacloud e diretamente às bibliotecas de interface das respectivas plataformas de IaaS. A ferramenta Deltacloud comportou-se como era esperado, *i.e.*, o tempo de execução das operações sofreu atrasos adicionais e a quantidade de informação trocada manteve-se em geral. Ambas as interfaces *Web* e REST foram testadas, apresentando medidas similares.

A solução de interoperabilidade desenvolvida para a integração e disponibilização de recursos de IaaS, através das plataformas PACI, OpenNebula e OpenStack, cumpre os requisitos especificados, *i.e.*, permite que a empresa Lunacloud expanda a gama de plataformas de IaaS adoptadas, oferecendo um *Web dashboard* e uma biblioteca de interface para a gestão integrada dos recursos de IaaS. As contribuições deste trabalho incluem os estudos e comparações realizados, a selecção da ferramenta de abstracção e, por último, mas não menos importante, o *driver* desenvolvido para a plataforma PACI.

Contents

List of Figures

List of Tables

List of Code Snippets

Acronyms

Notation	Description	Page
ACL	Access Control List	42
AD	Active Directory	42
AMD	Advanced Micro Devices	24
AMD-V	AMD Virtualization	24
AMQP	Advanced Message Queue Protocol	44
API	Application Programming Interface	3
ASF	Apache Software Foundation	49
ASLv2	Apache Software License version 2.0	49
AWS	Amazon Web Services	1
B2B	Business to Business	31
B2C	Business to Consumer	31
BLOB	Binary Large Object	145
BSS	Business Support System	62
C12G	Cloud Computing	37
CIDR	Classless Inter-Domain Routing	98
CIFS	Common Internet File System	110
CIM	Cloud Infrastructure Manager	35
CIMI	Cloud Infrastructure Management Interface	144
CLI	Command Line Interface	18
CPU	Central Unit Processor	16
DAS	Direct Attached Storage	41
DB	DataBase	40
DHCP	Dynamic Host Configuration Protocol	45
DIS	Dedicated Interoperable Service	139
DMTF	Distributed Management Task Force	144
DNS	Domain Name System	51
DNSaaS	DNS as a Service	147
DS	Data-Store	84

Notation	Description	Page
DSA	Distributed Systems Architecture	36
DSL	Domain Specific Language	152
EBS	Elastic Block Store	45
EC2	Elastic Compute Cloud	1
ERB	Embedded Ruby	175
EXT	linux EXTended file system	51
FP6	6th Framework Programme	149
FP7	7th Framework Programme	149
FS	File System	41
GPL	Generic Purpose Language	151
GPLv3	GNU General Public License version 3	49
GUI	Graphical User Interface	3
HA	High Availability	50
HAML	HTML Abstraction Markup Language	152
HTML	HyperText Markup Language	145
HTTP	HyperText Transfer Protocol	37
HTTPS	HyperText Transfer Protocol Secure	62
I/O	Input/Output	22
IaaS	Infrastructure as a Service	1
IAM	Identity and Access Management	55
IBM	International Business Machines	22
ID	IDentification	32
IDE	Integrated Development Environments	16
IM	Instance Manager	63
IP	Internet Protocol	44
IRB	Interactive Ruby Shell	154
iSCSI	internet Small Computer System Interface	38
ISV	Independent Software Vendors	27
IT	Information Technology	8
JSON	JavaScript Object Notation	51
KVM	Kernel-based Virtual Machine	24
L2	Layer 2	44
L3	Layer 3	44
LBaaS	Load Balancer as a Service	147
LDAP	Lightweight Directory Access Protocol	42
LEAD	Linked Environment for Atmosphere Discovery	54
LVM	Logical Volume Manage	47
MIME	Multipurpose Internet Mail Extensions	86
MPS	Microsoft Provisioning System	63
MVC	Model View Controller	153

Notation	Description	Page
NAS	Network-Attached Storage	41
NASA	National Aeronautics and Space Administration	42
NAT	Network Address Translation	45
NFS	Network File System	38
NIC	Network Interface Controller	54
NIST	National Institute of Standards and Technology	9
NSF	National Science Foundation	54
NTP	Network Time Protocol	180
OAM&P	Operation, Administration, Maintenance & Provision	51
OCA	OpenNebula Cloud API	40
OCCI	Open Cloud Computing Interface	39
OGF	Open Grid Forum	31
ONF	Open Networking Foundation	44
ORM	Object-Relational Mapper	47
OS	Operating System	16
OSS	Operations Support System	62
PaaS	Platform as a Service	15
PACI	Parallels Automation for Cloud Infrastructure	4
PAM	Pluggable Authentication Modules	49
PBA	Parallels Business Automation	62
PC	Personal Computer	1
PET	Privacy Enhancing Technologies	22
PID	Process ID	167
PKI	Public Key Infrastructure	189
POA	Parallels Operations Automation	62
PSBM	Parallels Server Bare Metal	62
QCOW2	QEMU Copy on Write Version 2	185
QEMU	Quick Emulator	185
QoS	Quality of Service	7
R&D	Research & Development	1
RBD	Rados Block Device	44
RDBMS	Relational Database Management System	187
REST	REpresentational State Transfer	39
REXML	Ruby Electric XML	175
RFC	Request For Comments	68
RHEL	Red Hat Enterprise Linux	188
RPC	Remote Procedure Call	37

Notation	Description	Page
RSA	Rivest r., Shamir a. and Adleman l.	42
S3	Simple Storage Service	44
SaaS	Software as a Service	9
SAGA	Simple API for Grid Applications	149
SAN	Storage Area Network	41
SDK	Software Development Kits	17
SELinux	Security-Enhanced Linux	188
SHA-1	Secure Hash Algorithm 1	206
SME	Small and Medium sized Enterprises	19
SOA	Service Oriented Architecture	31
SPI	Software, Platform, and Infrastructure	15
SQL	Structured Query Language	37
SSH	Secure Shell	38
SSL	Secure Sockets Layer	205
TCP	Transmission Control Protocol	59
UI	User Interface	50
UID	Unique IDentifier	74
UML	Unified Model Language	170
URI	Uniform Resource Identifier	67
URL	Uniform Resource Locator	69
UUID	Universally Unique IDentifier	92
VE	Virtual Environment	25
VGrDS	Virtual Grid application Development Software	54
VIM	Virtual Infrastructure Manager	35
VLAN	Virtual Local Area Network	11
VM	Virtual Machine	3
VM/370	Virtual Machine Facilityl370	23
VMM	Virtual Machine Monitor	16
VN	Virtual Network	16
VNC	Virtual Network Computing	46
VPC	Virtual Private Cloud	121
VPN	Virtual Private Network	10
VPS	Virtual Private Server	25
vSwitch	virtual Switch	41
WS-Agreement	Web Service Agreement	31
WSDL	Web Services Description Language	31
WSGI	Web Server Gateway Interface	46
WSLA	Web Service Level Agreement	31
WWW	World Wide Web	21

Notation	**Description**	**Page**
XaaS	Something as a Service	15
XML	eXtensible Mark-up Language	32
YAML	YAML Ain't Markup Language	165

Acknowledgements

First and foremost, I have to sincerely thank my supervisor, Professor Maria Benedita Campos Neves Malheiro, for the essential guidance, availability, and dedicated involvement in every step throughout the process.

I am also eternally grateful to Professor Pedro João De-Francesco Assis. He introduced me to an interesting topic on which I have enjoyed so much to work.

I would like to show gratitude to Eng.º Paulo Calçada for the opportunity given to me to work and learn from a state-of-the-art project.

Last, but not least, To my family and friends for the continuous encouragement and support during all this time.

Thank you all.

Chapter 1

Introduction

Cloud Computing has become a mainstream topic. This technology trend integrates the underlying structure, from the infrastructural to the applicational level, of well-known public Web services, *e.g.*, Amazon Web Services (AWS), Google App Engine and Spotify [1][2][3]. While the emergence of this technology on the market is related to the creation of Salesforce Web applications for enterprises or the launch of AWS, the concept of Cloud Computing dates back to 1984 when Sun Microsystems adopted John Gage's famous expression "The Network Is The Computer" [4].

The term "Cloud Computing" idealizes the provision of computing resources, abstracted by the cloud as Web services, that are available everywhere through a different set of interface devices connected to the Internet. This technology contrasts with the individual and local computation resources, *e.g.*, Personal Computer (PC), that provide confined software programs with a limited hardware configuration.

The Elastic Compute Cloud (EC2) created by Amazon back in 2006 initiated the provision of the Infrastructure as a Service (IaaS) concept. This type of service was rapidly adopted by other well-known technology enterprises with large computing resources, that launched their own IaaS platforms. As a result, the Research & Development (R&D) community as well as the involved enterprises concentrated efforts on the development of open source and proprietary software solutions for enabling the deployment of IaaS platforms. Since Cloud Computing was a recent concept, lacking pre-defined standards and even a consensual definition, the resulting platforms were highly heterogeneous in terms of functionalities, architecture and interface libraries. This diversity hinders the selection of an IaaS platform and, above all, constitutes an obstacle to the interoperability among service providers.

The European cloud services provider LunaCloud intends to face the presented problems with the implementation of multiple IaaS platforms for the high level management and monitoring of its IaaS business segment. Currently, the services offered by LunaCloud have limitations in terms of automation and orchestration of actions and the advanced proprietary platform management library has no compatibility and interoperability support with other IaaS platforms. To overcome this problem, LunaCloud considers the development of an open source solution for the integrated management of cloud computing infrastructure resources, promoting interoperability and standardization between heterogeneous IaaS platforms. This thesis addresses this challenge and proposes a solution.

1.1 Motivation

The choice for the current thesis theme relates to the personal interest to deepen the knowledge associated with the Cloud Computing paradigm, more specifically, with the IaaS provisioning level. To work with technologies related with cloud infrastructure resources supply - virtualization, management, monitoring and interface systems - as well as to get to know the existing diversity of IaaS platform approaches, mainly the open-source solutions.

From a professional perspective, this work provided the opportunity to participate and learn from a state-of-the-art project - the study, selection and deployment of a multi-platform IaaS abstraction solution.

From a social perspective, this project provided the possibility to contribute to existing open-source communities.

1.2 Thesis

This research work intends to answer the following questions:

1. How much do current IaaS platforms differ?
2. Can heterogeneous IaaS platforms be managed in an integrated manner from the client perspective?
3. Is it possible to develop an interoperable service for the high level management of the IaaS business segment of Lunacloud?

1.3 Objectives

The main goal of this project is to propose and develop an interoperable service for the integrated management of cloud resources provisioned by the IaaS platforms

used by Lunacloud. Inherent to the accomplishment of the presented objective, the following tasks were defined:

- Survey of open-source IaaS platforms as well as of Lunacloud proprietary solution;
- Study of different IaaS platform Application Programming Interface (API);
- Survey on existing IaaS interoperable open-source solutions;
- Proposal, specification and development of a meta-library/meta-service to interface with the IaaS platforms used by Lunacloud;
- Development of a front-end application to configure and manage the infrastructure resources provided by the Lunacloud proprietary infrastructure platform as well as other open source IaaS platforms.

1.4 Functional Tests

The implemented solution should allow the interaction with the included IaaS platforms via a common interface. In particular, the solution adopted for the integration of the different IaaS platforms should allow the user to:

- Create and delete a Virtual Machine (VM);
- Perform a VM action (*e.g.*, start and stop a VM);
- List his/her resources (*e.g.*, images and VM);
- Obtain the characteristics of a specified resource.

1.5 Expected Results

The proposed solution should:

- Provide a standard programming interface for the integrated management of the Lunacloud proprietary interface platform as well as other open-source IaaS platforms;
- Perform all adaptation and data transformation operations;
- Allow the management of virtual resources from the different integrated IaaS platforms;
- Expose a Graphical User Interface (GUI).

1.6 Contributions

The provision of an interoperable service for the integrated management of IaaS resources was supported by the following contributions: (*i*) a thorough survey and comparison of the IaaS platforms to be integrated (open source platforms and the PACI proprietary platform); (*ii*) a detailed survey, comparison and selection of a candidate abstraction solution; (*iii*) the set up of a test bed platform to assess the interoperable service proposed; and (*iv*) the implementation of the PACI driver. The PACI driver was shared with the Deltacloud community and the detected OpenNebula and OpenStack driver malfunctions were also reported.

1.7 Work Plan

The project Gantt chart presented in Figure 1.1 includes the project milestones and the tasks performed: the state of the art on Cloud Computing, a survey on the most popular open-source IaaS platforms, the study of OpenNebula, OpenStack, CloudStack and Parallels Automation for Cloud Infrastructure (PACI) user interfaces, a survey on cloud interface abstraction solutions, the familiarization with the Ruby programming language and the Deltacloud abstraction framework, the development of a PACI driver for Deltacloud, the test bed set-up, the tests conducted and corrections performed and the dissertation writing.

ID	Task Name	Start	Finish	Duration
1	State of the art on Cloud Computing	13-03-2013	15-05-2013	46d
2	Writing the "Cloud Computing" chapter	10-04-2013	05-06-2013	41d
3	Official Thesis proposition	05-06-2013	05-06-2013	0d
4	Survey on open-source cloud IaaS platforms	29-05-2013	08-08-2013	52d
5	Meeting with Lunacloud and EuroCloud PT	15-07-2013	15-07-2013	0d
6	Writing the "Cloud Infrastructure Platforms" chapter	19-06-2013	28-08-2013	51d
7	Study of OpenNebula, OpenStack, CloudStack and PACI user interfaces	28-08-2013	25-09-2013	21d
8	Writing the "Interface Libraries Comparison" chapter	11-09-2013	30-10-2013	36d
9	Survey on cloud interface abstraction solutions	16-10-2013	14-11-2013	22d
10	Writing the "Interoperable Interface Proposal" and the "Introduction" chapters	31-10-2013	19-12-2013	36d
11	Interoperable Solution Proposition	18-12-2013	18-12-2013	0d
12	Familiarization with the Ruby programming language	18-12-2013	16-01-2014	22d
13	Familiarization with the Deltacloud abstraction framework	02-01-2014	05-02-2014	25d
14	Development of the PACI driver	06-02-2014	04-03-2014	19d
15	Test bed set-up	22-01-2014	12-03-2014	36d
16	Test, debugging and validation	12-03-2014	16-04-2014	26d
17	Writing the "Development Environment" and "Project Development" chapters	12-02-2014	21-04-2014	49d
18	Writing the "Test, debugging and validation" and "Conclusions" chapters	23-04-2014	12-05-2014	14d
19	Conclusion of the "Integrated Management of Cloud Computing Resources" Dissertation document	16-05-2014	16-05-2014	0d

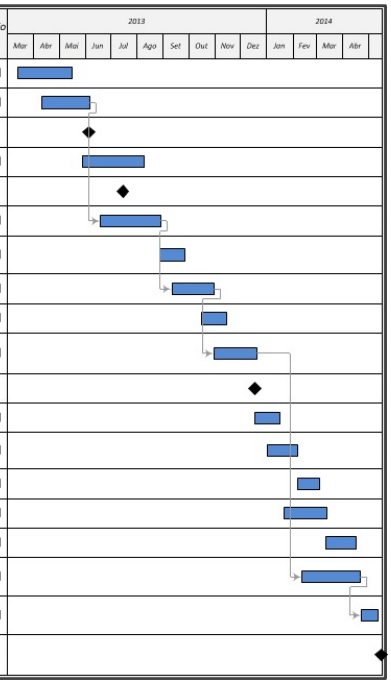


Figure 1.1: Project schedule

1.8 Structure of the Dissertation

This document is structured in nine chapters:

- The first chapter, "Introduction", presents and contextualizes the developed project, emphasizing the motivation, main objectives and followed methodology, the work plan and the structure of the dissertation;
- The second chapter, "Cloud Computing", addresses the fundamental concepts of the Cloud Computing technology, including its definition, the deployment and service models, the main features and concerns, virtualization technologies and the business perspective;
- The third chapter, "Cloud Infrastructure Platforms", describes and compares four different open-source cloud infrastructure platform solutions - OpenNebula, OpenStack, CloudStack and Eucalyptus - and a proprietary cloud infrastructure platform - PACI - for the identification of common features that will enable the development of a management solution that meets the scope of this thesis;
- The fourth chapter, "Interface Libraries Comparison", presents a study and comparison of the client interface libraries provided by OpenNebula, OpenStack, CloudStack and PACI IaaS platforms in terms of programmable components and available operations, including the parameters and respective attributes;
- The fifth chapter, "Interoperable Interface Proposal", considers two possible approaches for the creation of an interoperable service for the IaaS platforms studied in the previous chapter;
- The sixth chapter, "Project Development", describes the development environment, including the programming languages, front-end and back-end technologies and the development tools adopted in the development of the project, and details the project development process, including the conceptual architecture, the interoperable front-end services, the deployment configuration of the interoperable service and the included driver modules (reused and developed);
- The seventh chapter, "Test, Debugging and Validation", describes the installation and configuration of the Interoperable Service test bed and presents the tests performed and results obtained;
- The eighth chapter, "Conclusions", presents the discussion of the main dissertation conclusions, the envisaged usage cases and future developments.

Chapter 2

Cloud Computing

This chapter presents the definition, deployment and service models, characteristics, concerns, virtualization technologies and business perspective of the Cloud Computing paradigm.

2.1 Definition

Cloud computing is a relatively recent computation paradigm in which virtualized and dynamically scalable resources are provided as services (*i.e.* programs, storage, application-development platforms) over a network, usually the Internet, to end users/clients through a variety of devices (*i.e.* laptops, smartphones, PC) [5].

This new computation model has evolved from itspredecessor computation paradigm, Grid Computing, introduced by Foster and Kesselman in 1999 [6], combining the Grid (as its backbone and infrastructure support) with Virtualization and Utility Computing. Grid Computing can be defined as a type of parallel and distributed system that enables the sharing, selection and aggregation of geographically distributed autonomous computer resources dynamically at run-time depending on their availability, capability, performance, cost, and users Quality of Service (QoS) requirements [7]. In other words, this computation model provides a combination of computer resources linked by a network or by the Internet (forming a grid) to provide the essential requirements (performance, cost, QoS) to deal with scientific or technical problems that require a great number of computer processing cycles or large amounts of data. Thus, a cloud computing technology takes advantage of the Grid to provide his computation resources but a Grid is not necessarily a Cloud or part of a Cloud. In fact, the evolution from Grid Computing to Cloud Computing was a result

of a shift in focus from the infrastructure concept of the Grid Computing (an infrastructure that delivers storage and compute resources and that focus on a single specific utilization) to one that is economy based aiming to deliver more abstract resources and services [8]. Virtualization and Utility Computing have an important role in this "economy concept shift". The first, virtualization, is essential for the provision of an encapsulation and abstraction layer from the underlying hardware and system software running in the data centre servers. This way, at the raw hardware level, resources can be added or withdraw according with the demand, while the interface to the user is not changing. The application of virtualization as part of the technologies used by Cloud Computing enables scalability, making computation resources more profitable, and flexibility on its physical layer [9]. The virtualization techniques and their architecture will be discussed, at greater detail, further, in the Virtualization Technologies section. The last, Utility Computing, is a technology that can be defined as "a business model in which computing resources, such as computation and storage, are packaged as metered services similar to a physical public utility, such as electricity and public switched telephone network." [8]. The concept is similar to the one found in the electric grid or water supply payment models, as the user/client pays for the amount of resources consumed. In a similar way, Cloud Computer services are charged based on the processing power or amount of data utilized, in a "pay as you go" philosophy [5]. This approach provides advantages to the end users as the need for up-front infrastructure investment is minimal, either with software licences (and unused but paid software licenses) or with hardware infrastructure and related maintenance and staff [10].

The definition of Cloud Computing has been a controversial topic. Computer researchers, analyst firms and Information Technology (IT) institutions define Cloud Computing in different manners [9][11][12]. Gartner and Merrill Lynch [13][14], two analyst firms, define Cloud Computing as follows:

- "A style of computing in which massively scalable IT related capabilities are provided "as a service" using Internet technologies to multiple external customers." - Gartner 2008 [15];
- "The idea of delivering personal (*e.g.*, email, word processing, presentations.) and business productivity applications (e.g., sales force automation, customer service, accounting) from centralized servers" - Merrill Lynch 2008 [16].

As from the scientific community the opinion on the mater is not unanimous but still adds, to the end user perspective of the earlier definitions, the architectural aspects of the infrastructure (data centre hardware, virtualization and scalability), as can be read in the following definitions:

- "Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as Software as a Service (SaaS), so we use that term. The datacenter hardware and software is what we will call a Cloud." - Berkeley RAD [17];

- "A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualised computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers." - R. Buyya, C.S Yeo, and S.Venugopal [18];

- Cloud Computing is "A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet." - Foster, Zhao and Raicu, Lu, 2008 [8].

More recently, the U.S. Government's National Institute of Standards and Technology (NIST) published on September 2011 his 16th and final version Cloud Computing definition [19] as:

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

The NIST definition lists four deployment models (private, community, public and hybrid), three service models (software, platform and infrastructure) and five essential characteristics of cloud computing (on-demand self-service, broad network access, resource pooling, rapid elasticity or expansion and measured service). Although this definition is not consensual, *e.g.* [20][21][22], it is a reference for cloud services and deployment strategies broad comparisons and to provide a baseline for cloud computing discussions [23]. Thus, the NIST cloud computing definition will be the reference used in the development of this project.

2.2 Deployment Models

Clouds can be classified based on the underlying infrastructure deployment model. A deployment model defines the architecture and the purpose of the cloud and

refers to the location and management of its infrastructure. As previously mentioned, the NIST [19] identifies four different deployment models for Cloud Computing: private clouds, community clouds, public clouds and hybrid clouds.

2.2.1 Private Clouds

The private cloud infrastructures are intended for the exclusive use of a single organization with multiple consumers and emulate cloud computing on private networks. In this operating model the infrastructure may be owned, managed and operated by the organization (on premises), a cloud provider (off premises) with some contractual SLA or a combination of them. The on premises and off premises variants can be separated in: (*i*) on-site private clouds and (*ii*) outsourced private clouds. This two models, regarding its physical location, affect differently the security perimeter of the computation resources. On the on-site private cloud scenario presented in Figure 2.1 the private cloud may be secured and controlled inside the organization by its IT personal. Clients can access the cloud from within the security perimeter or outside the security perimeter trough a boundary controller composed by firewalls and Virtual Private Network (VPN) connections.

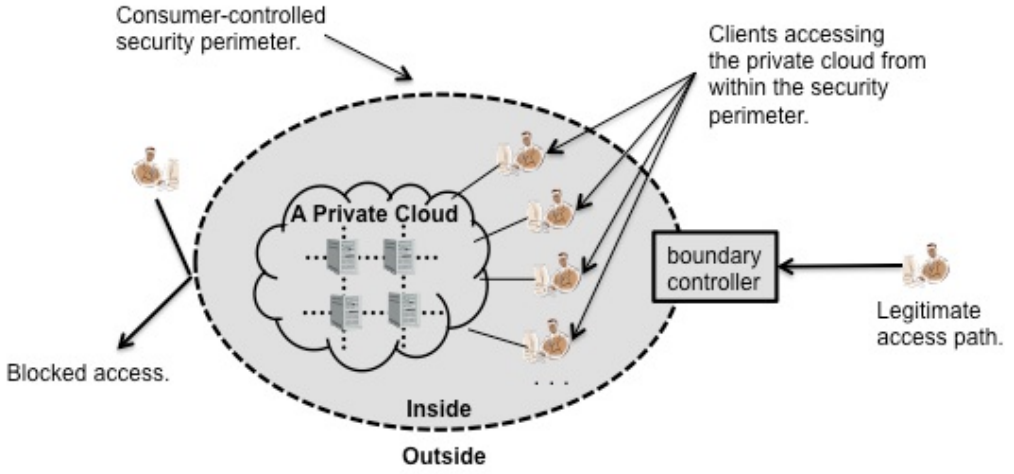


Figure 2.1: On-site (on premises) private cloud representation [24].

The outsourced private cloud scenario represents an off premises type of private cloud. The cloud provider, sells to the cloud consumer (*e.g.*, an organization) its infrastructure services. In this case, the physical infrastructure belongs to the cloud provider, which is responsible for its management, but the provided service must follow the parameters celebrated by the contractual SLA between the two parties regarding security, management, privacy and other policies that concern the cloud consumer. Thus, as presented in Figure 2.2, the outsourced

private cloud has two security perimeters that are connected by a protected communication link: one implemented by a cloud consumer (on the right) and one implemented by the cloud provider (on the left).

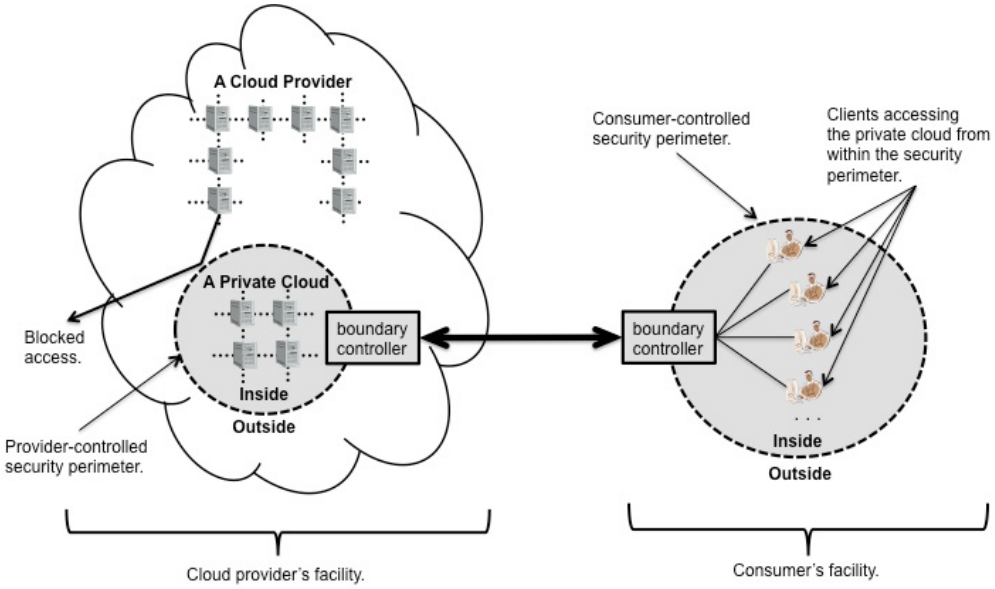


Figure 2.2: Outsourced (off premises) private cloud representation [24].

The cloud provider is responsible to enforce his security perimeter and to separate the private cloud resources from the other cloud resources that are outside the security perimeter. Depending on the consumer's security requirements, different mechanisms such as Virtual Local Area Network (VLAN), VPN, separate network segments or clusters can be applied. The cloud consumer implements and controls within his security perimeter the access to the private cloud resources.

2.2.2 Community Clouds

The community cloud deployment model is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (*e.g.*, mission, security requirements, policy, and compliance considerations) [19]. The use of a community cloud can help to reduce the costs of organizations as compared to the implementation of individual private clouds as the costs are supported and shared by a larger group. Like in the private model of the cloud, the infrastructure can be owned, managed and operated within the community by one or more of its organizations, in an on premises on-site scenario, or outside the community by a third party (cloud provider), in an off premises outsourced scenario.

Figure 2.3 presents an example of an on-site community cloud. The repres-

ented community is composed by a set of participant organizations. Each one of them may provide cloud infrastructure services, consume cloud services or both.

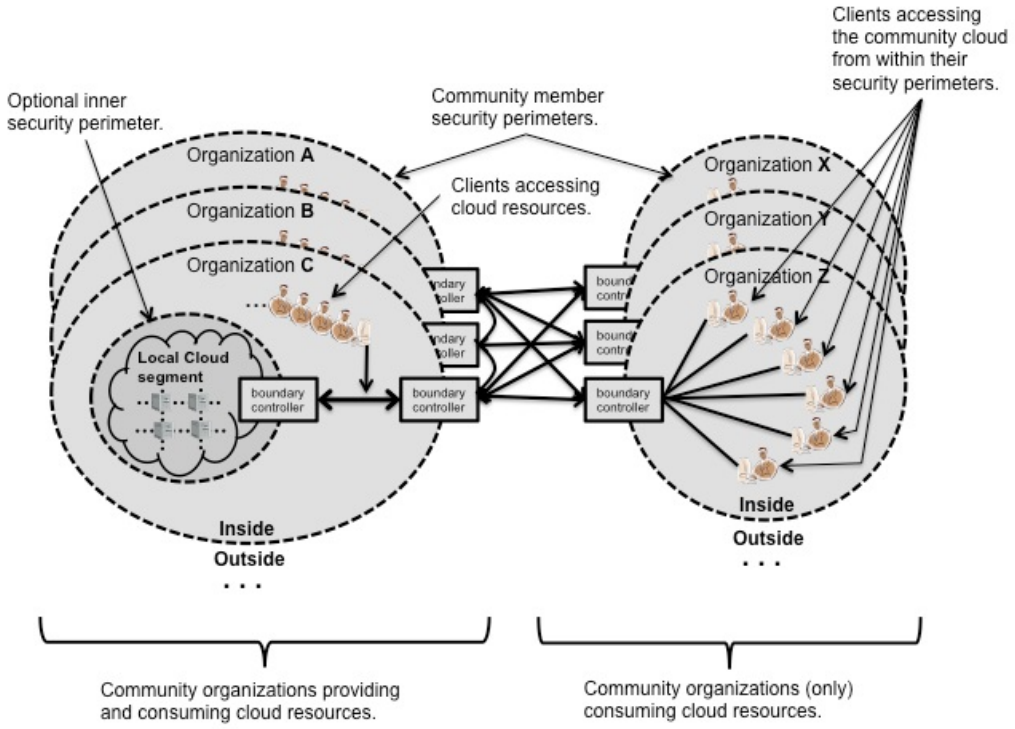


Figure 2.3: On-site (on premises)community cloud representation [24].

In Figure 2.3 the organizations providing services are on the left and the ones that consume cloud services are on the right. The inner organizations networks are separated from the outside networks by a security perimeter like in the on-site private cloud scenario. The communication inside the community, *i.e.*, between the organizations that provide services and the ones that consume them, are made through the boundary controller of each organization. Optionally, organizations can protect the cloud infrastructure from other computation resources with another layer of a security perimeter.

On the outsourced community cloud, the organizations of a community only consume cloud resources. As Figure 2.4 illustrates, the infrastructure of the cloud is provided by a third party with conditions identical to the ones described in the outsourced private cloud scenario, *i.e.*, the provided service is specified trough SLA agreements. The community organizations access the cloud resources inside the security perimeter trough the boundary controller and the communication between the provider and consumers is done through a security link between the cloud provider and the community's organizations.

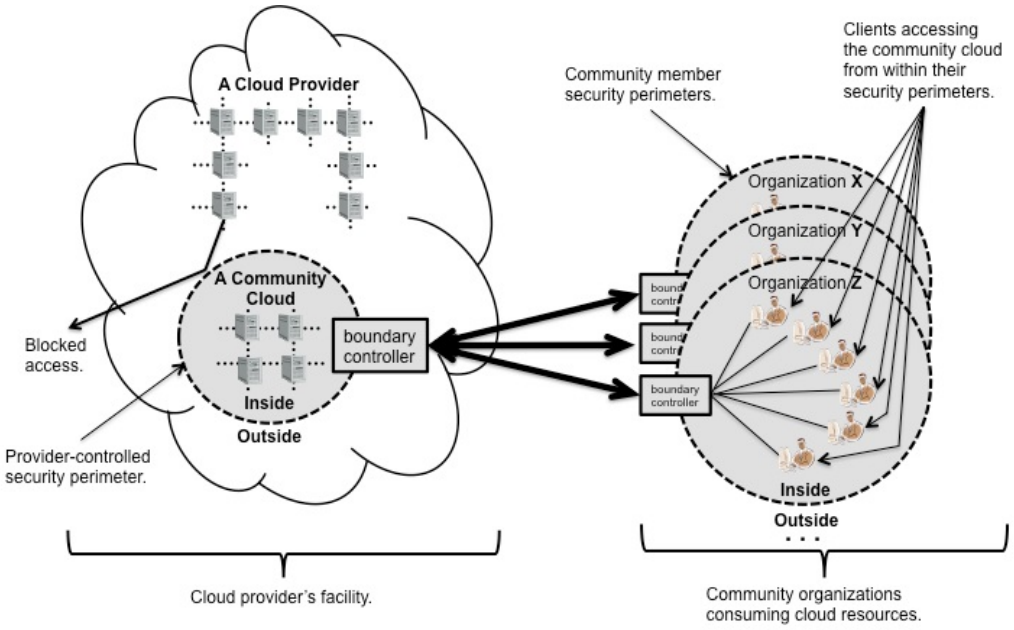


Figure 2.4: Outsourced (off premises)community cloud representation [24].

2.2.3 Public Clouds

The public cloud model describe cloud computing in the traditional mainstream sense whereby resources are dynamically provisioned on a self-service basis over the Internet. The cloud infrastructure that may be owned, managed and operated by a business, academic or governmental organization is intended for open use by the general public.

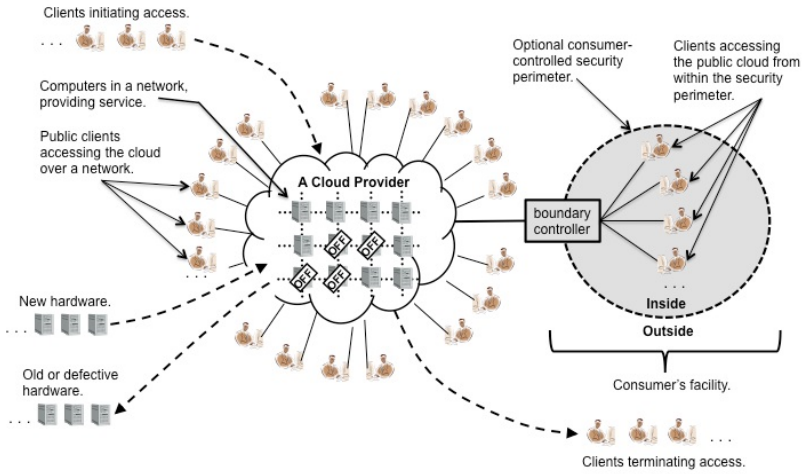


Figure 2.5: Public cloud representation [24].

In a public cloud, security management, infrastructure maintenance and other operations are relegated to the cloud provider as illustrated in Figure 2.5. Compared to the private cloud model, the public cloud service offering has a low degree of control and oversight of the physical and logical security aspects such as security perimeter to separate computational resources (usually present in the outsourced type of the private cloud model).

2.2.4 Hybrid Clouds

Any composition of private, community or public clouds form a hybrid cloud as represented in Figure 2.6. The individual clouds remain unique entities and, so, a hybrid cloud can change over time with constituent clouds joining or leaving.

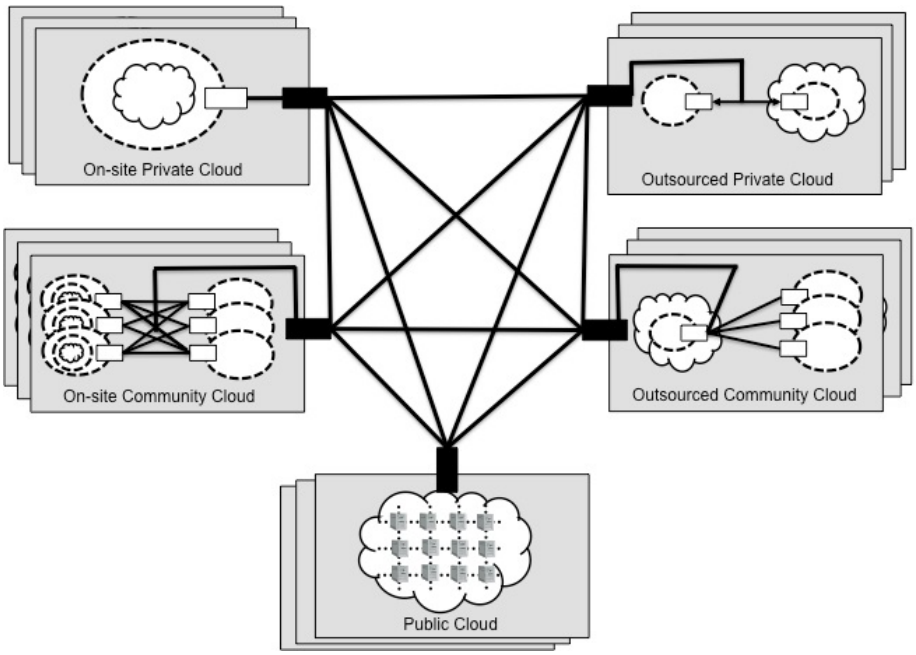


Figure 2.6: Hybrid cloud representation [24].

The aggregation of clouds are bound together by standardised or proprietary technologies that enable data and application portability. Some of the benefits of this complex model are the access to external clouds during periods of high demand, to perform or to provide backup resources or to run non-core applications in a public cloud while maintaining core applications and sensitive data in an on premises private cloud.

2.3 Service Models

Cloud Computing enables hardware and software to be delivered as services. In this context, the term service is used to reflect the fact that they are provided on demand and paid on a utility computing usage basis. As cloud computing has developed, different vendors offer clouds that have different services associated. These services are usually described on a Something as a Service (XaaS) taxonomy. According to NIST, these services can be classified into one of three delivery models: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). These three service classes are also known as the Software, Platform, and Infrastructure (SPI) service model.

As shown in Figure 2.7, the SaaS layer is built on top of the PaaS and this on the top of the Iaas service layer and all of these services are abstracted from the physical resource layer through a resource abstraction and control layer.

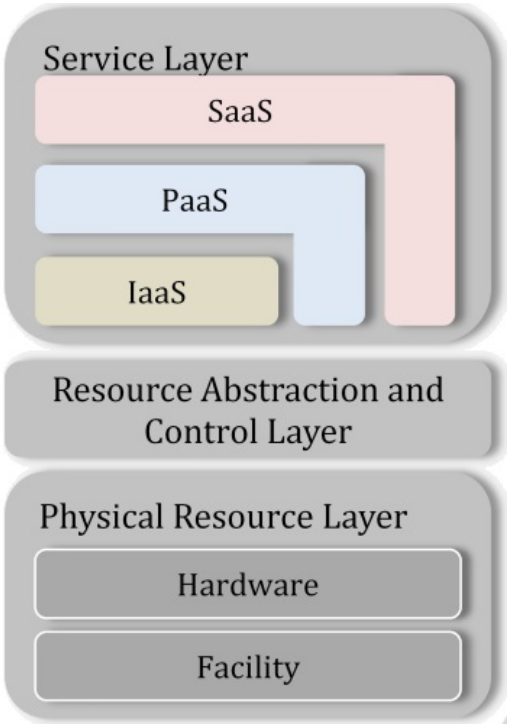


Figure 2.7: Service orchestration (modified from [24]).

This service orchestration indicates that the provisioning of a top service, like SaaS, implies the deployment of the underlying services by the cloud provider. This means that when the cloud consumer requests a SaaS the cloud provider assumes most of the responsibilities in managing and controlling the underlying applications and infrastructure. In the same way, when a IaaS is required by the cloud consumer the cloud provider has control over the physical hardware and

cloud software that makes the provisioning of infrastructure services possible, *i.e.*, the physical servers, network equipments, storage devices, host Operating System (OS) and hypervisors for virtualization, while the cloud consumer uses the available computing resources (*e.g.*, a virtual computer), for their fundamental computing needs. Compared to SaaS and PaaS cloud consumers, an IaaS cloud consumer has access to more fundamental forms of computing resources and thus has more control over the more software components in an application stack, including the OS and network [24]. Depending on the contracted service, the cloud consumer can choose from a variety of solutions, *i.e.*, from top level software applications to lower level computation. The SPI service model is described in the following Infrastructure as a Service, Platform as a Service and Software as a Service subsections.

2.3.1 Infrastructure as a Service

This type of service provides on-demand virtualized resources (*e.g.*, virtual computers, virtual storage, Virtual Network (VN) and other hardware assets as resources to the cloud consumer (system administrators). The service consumer gets access to the contracted virtual machines, network-accessible storage and network infrastructure components (*e.g.*, firewalls and configuration services). The cloud infrastructure orchestration will be discussed in detail in the Cloud Infrastructure Platforms chapter since it is the main focus of this work.

The IaaS cloud provider manages all the physical infrastructure hardware and has administrative control over the resource abstraction and control layer that wraps the Virtual Machine Monitor (VMM), also known as Hypervisor, billing software solutions, servers host OS and other necessary tools to implement the desired service. The consumer, on the other hand, can make requests to the hypervisor through the interface provided by the service vendor to manage and create new virtual machines, chose the guest OS to run on the VM and control all the middle-ware and applications that run on top of the guest OS.

The service usage fees for a IaaS are calculated typically per Central Unit Processor (CPU) hour, GB stored per hour, network bandwidth consumed and value-added services used (*e.g.*, monitoring and automatic scaling) [25]. The Amazon Web Services [1] and LunaCloud Cloud Server [26] are two examples of IaaS providers.

2.3.2 Platform as a Service

In a platform as a service model, the PaaS provider supports the development, deployment and management process of the PaaS consumers (developers) by providing tools such as application programming interfaces, Integrated Devel-

opment Environments (IDE), development version of cloud software, Software Development Kits (SDK) as well as deployment and management tools.

In comparison to the IaaS model this service offers a higher level of abstraction to the cloud consumer. The PaaS provider manages the cloud infrastructure, the operating system and the enabling software whereas the PaaS consumers are responsible for installing and managing the applications that are deploying, to comply with the programming language requirements and distribution as well as with the payment terms.

In this service model cloud consumers have limited or no access to the underlying infrastructure platform, *i.e.*, network, servers, operating systems, or storage. The billing procedures are implemented according to, processing, database storage and network resources consumed by the developed application as well as the duration of the platform usage. The Google AppEngine [2] is an example of an PaaS provider.

2.3.3 Software as a Service

The software as a service model resides on the top of the SPI stack. This service provides applications accessible trough a Web browser to the cloud consumers. In comparison with the other two underlying models (PaaS and IaaS), the SaaS is the most abstract service solution from a consumer's perspective.

The SaaS provider is responsible for the underlying infrastructure, the platform where the applications are located and for the installation and maintenance of the provided applications. On the other hand, the SaaS consumer is only responsible for the accession, uploading, maintaining and interaction with his data.

The SaaS can ease the burden with software licensing and maintenance and the billing system applied is based on the number of end users, the time of use, the network bandwidth consumed, the amount of data stored or duration of stored data. Examples of SaaS providers are Google Apps for Business [27] and Salesforce productivity applications [28].

2.4 Characteristics

Due to the lack of standardization and consensus regarding cloud computing, the characteristics that define the cloud vary between all interested parties [29][30][31].

In order to promote simplification and standardization, the NIST identifies five essential and generally accepted main characteristics to describe the behaviour of cloud computing [19]: (*i*) on-demand self-service; (*ii*) broad network access; (*iii*) resource pooling; (*iv*) scalability; and (*v*) measured service. These cloud characteristics will be described in the following subsections.

2.4.1 On-demand Self-service

The cloud consumer can use resources of the contracted cloud computing service (*e.g.*, system and network resources) as needed without human interaction between consumer and cloud provider in an on-demand form. The resources can be controlled trough a user-friendly self-service interface, *i.e.*, a Web page dashboard or Command Line Interface (CLI) console, providing effective means to manage the service offerings and promoting cost savings to both consumer and cloud provider. This cloud feature is important since it solves the resource demand fluctuation problem and reduces the costs by avoiding planning ahead the installation of additional resources to meet peak demands and underusing installed resources.

2.4.2 Broad Network Access

The access to resources hosted in the cloud is available over private networks, *e.g.*, private cloud and community cloud implementations, or over the Internet, *e.g.*, public clouds, trough standard mechanisms. Therefore, the broad network access provides platform independence as the services can be accessed from all types of operating systems and different types of computating platforms, *e.g.*, desktops, tablets or smart phones. This feature provides access to resources (mainly in the public cloud scenario) from a wide range of locations as long as they have Internet access.

2.4.3 Resource Pooling

The resources pooled together by the cloud provider are assigned and reassigned to serve multiple consumers in a multi-tenant model. These physical and virtual systems are dynamically allocated or reallocated as needed. The resources can be physically located at many geographic locations and assigned as virtual components of the service as they are requested, creating, as defined by NIST [19], "a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources".

2.4.4 Rapid Elasticity

Rapid elasticity refers to the ability of the cloud to expand or reduce provisioned resources quickly and efficiently, fulfilling the requirements of the on-demand, self-service, characteristic of cloud computing. Through the broad network access diverse computing infrastructures can cooperate to allocate these resources (*e.g.*, federation of clouds in a hybrid cloud or a community cloud scenario), manually or automatically, appearing to the consumer as a large, limitless pool of dynamic resources that can be purchased at any time and in any quantity. This feature

is normally negotiated between cloud providers and cloud consumers through a SLA. In this type of agreement, the provider negotiates with the client the time response for the provision of the resources (as they are in fact finite) in order to respond to the customer's instant demands as well as the penalties for not meeting the negotiated contract terms (in this case the accorded time response).

2.4.5 Measured Service

Due to the service oriented characteristics of cloud computing, the amount of cloud resources used by the cloud consumer are monitored, measured and reported as they are allocated or consumed. This procedure is important for quality of service requirements, billing procedures and to provide transparency for both the cloud provider and the cloud consumer. As previously mentioned, cloud computing uses an utility computing model in which the consumer is charged based on the level of service provided. For that purpose known indicators (metrics) such as amount of storage, network resources or level of processing power are used to measure the amount of consumed resources. Normally, this information is provided automatically via the consumer self-service interface.

2.5 Concerns

Although cloud computing has several benefits, there are also significant concerns that challenge this new computing model. Generally, the advantages of cloud computing are more compelling to Small and Medium sized Enterprises (SME) than to large organizations. Larger organizations can easily create and maintain IT teams and infrastructures as well as develop custom software solutions customised to their own needs. On the other hand, the adoption of cloud computing implies application customization, control delegation and, most importantly, cloud provider dependency. Outsourced services evade the physical, logical and personal control of an organization IT staff and data outside a company's control security perimeter pose an intrinsic risk threat. These concerns can make corporate executives hesitate to migrate to cloud computing.

Three of the most significant barriers to the adoption of cloud computing are: (*i*) unpredictable performance; (*ii*) security risks and (*iii*) data privacy.

2.5.1 Unpredictable Performance

Cloud consumers have certain expectations regarding the service level provided by the cloud. Some of these expectations include availability of service and overall performance. Due to the abstracting nature of the cloud provided services, the cloud consumer ignores how or where the contracted service runs on top of the physical infrastructure, the number of physical machines used or their loca-

tion. On the other hand, the economic benefits of cloud computing are based on the ability to increase the usage level of the infrastructure trough multi-tenancy. Multi-tenancy introduces service under-performances, since it cannot ensure that the activities of different consumers are totally independent and disjoint. The cloud provider's challenge is, thus, to efficiently manage all the virtualized resources so that service performance remains as unaffected as possible by the number of consumer cloud resource demands.

Physical resources such as CPU cores, disk space and network bandwidth must be divided and shared among virtual machines, running potentially heterogeneous workloads. The mapping between resources and corresponding virtual machines becomes increasingly complex to ensure the profitability of the physical interface. The VM mapping is dynamic and provides mechanisms to suspend, migrate and resume virtual machines by pre-empting low-priority allocations in favour of high-priority ones. These operations combined with load balancing, backup and recovery mechanisms cause performance variation.

Other important factors are external, non controllable network related parameters such as latency, introduced by the communication link to the data centre where the servers are hosted, as well as competing traffic in the communications link.

Cloud consumers expect a fixed performance/fee ratio, but this cannot be guaranteed as the performance may fluctuate. Furthermore, the cloud end user is unable to predict these variations, their magnitude and duration because the service measurements are made by the cloud provider services and not at the consumer premises. In these cases, the SLA is used as a warranty to compensate the cloud under-performance.

2.5.2 Security Risks

Security is one of the main problems for enterprises to move their "in-house" data to public clouds or outsource private clouds. Most of the cloud providers do not guarantee the security of data while being transferred to the cloud [32]. On the other hand, because cloud computing represents a relatively new computing model, there is a great deal of uncertainty regarding how to provide security at the network, host, application and data levels.

Compared to traditional technologies, the cloud has many specific features that make the traditional security mechanisms (*e.g.*, identity, authentication and authorization) insufficient [33]. Additionally, the cloud service models are interdependent since, on one hand, PaaS and SaaS are hosted on top of IaaS and, on the other hand, PaaS provides the platform to SaaS. Due to these dependencies, any attack to a cloud service layer can compromise the upper layers, *e.g.*, a breach in the infrastructure service will impact the security of both PaaS and SaaS.

Relationships and dependencies between cloud models can also be a source of security risks. One SaaS provider may rent a development environment from a PaaS provider, which might also rent an infrastructure from other IaaS provider. This may result in an inconsistent combination of security models because each provider implements its own security model.

There are security issues in all three SPI model layers: (*i*) SaaS applications suffer from vulnerabilities and accessibility issues; (*ii*) PaaS, which depend on third-party relationships and development life-cycle, inherits security issues; and (*iii*) IaaS experiences virtualization vulnerabilities and malicious VM images. Some of the countermeasures used to minimize security threats are the implementation of dynamic credentials, digital signatures, data encryption, frameworks to secure virtual networks and hypervisor data flow inspection [32].

2.5.3 Data Privacy

Privacy issues are increasingly important in the World Wide Web (WWW). The delegation of data control to the cloud provider when utilizing a cloud service makes cloud consumers uncomfortable because of the risks to data integrity, confidentiality and privacy principles as data becomes accessible to an augmented number of parties.

Security breaches like the ones described in the security section can lead to private data access and exposure, but data confidentiality can also be breached unintentionally through, *e.g.*, data remanence. Data remanence is the residual representation of data that have been erased or removed. Due to the virtual separation of logical drivers and lack of hardware separation between multiple users on a single infrastructure, the remaining residual information stored in the physical storage units may lead to the unwilling disclosure of private data or to the intentional exploitation of that vulnerability.

Another fact that concerns enterprises regarding data privacy protection is the geographical distribution of the information inside the clouds. The fault tolerance and backup mechanisms that cloud providers utilize to prevent system failures as well as to provide data storage resources to fulfil cloud consumers space requirements lead to data fragmentation across geographic locations. The challenge with this distribution is that information can be stored in different countries with diverse data protection laws and, thus, cloud providers cannot ensure personal data requirements against the countries legal framework.

The solution to this problem involves the creation of initiatives like the Madrid International Conference [34] with the aim to create international agreements where a set of principles and rights guaranteeing the effective and internationally uniform protection of privacy with regard to the processing of personal data

between different countries to facilitate the international flow of personal information and ease data privacy concerns.

On the other hand, the adoption of proactive-measures requirements can be met trough the implementation of Privacy Enhancing Technologies (PET)[1]. PET safeguards the individual data privacy and rights by protecting personal data and preventing its unnecessary or undesired processing.

2.6 Virtualization Technologies

Virtualization is a key technology that enables part of the most important characteristics of cloud computing (*e.g.*, resource pooling and rapid elasticity). This technology has been proposed and developed over a relatively long period by International Business Machines (IBM) in 1960-1970 [35][36][37]. Its purpose in cloud computing is to divide and abstract the resources of the physical infrastructure into multiple segregated virtual systems, virtual machines with virtual networks and virtual storage, through which the cloud services are provided.

"A virtual machine is taken to be an efficient, isolated duplicate of the real machine." [38]. It has its own address space memory, processor resource allocation and device Input/Output (I/O) trough virtual device drivers. These virtual systems are independent from each other and provide a complete platform for support and execution of an operating system. Normally, an OS running on a virtual machine is called a Guest OS and the OS that runs on top of the physical hardware is called Host OS. The VM lifetime has six phases: create, suspend, resume, save, migrate and destroy. Multiple virtual machines can run simultaneously on the same physical infrastructure node (*i.e.*, physical server) and each VM, residing in the same computing node, can execute different Guest OS.

To enable virtualization a low-level program called Virtual Machine Monitor or Hypervisor is required to provide an environment for programs identical with the one of an physical machine. The hypervisor is in control of the system resources and provides them access to virtual machines and management functions regarding the existing VM on the computing node. The access to resources relies on load balancing techniques that are responsible for mapping logic addresses to physical addresses and for the management of workloads. Depending on the installation of the virtualization layer, two types of hypervisors can be identified:

- Type 1 hypervisors: The virtualization layer containing the hypervisor is directly installed on top of the physical infrastructure node without the need for a host operating system. This architecture, also known as bare metal virtualization, enables direct access to the hardware resources;

[1] Set of computer technologies that allow online users to protect the privacy of their personally identifiable information provided to and handled by Web services or applications.

- Type 2 hypervisor: The virtualization layer is installed on top of the host operating system rather than on top of the infrastructure hardware of the node. This architecture has no direct access to the hardware resources.

Figure 2.8 shows a diagram of Type 1 and Type hypervisors.

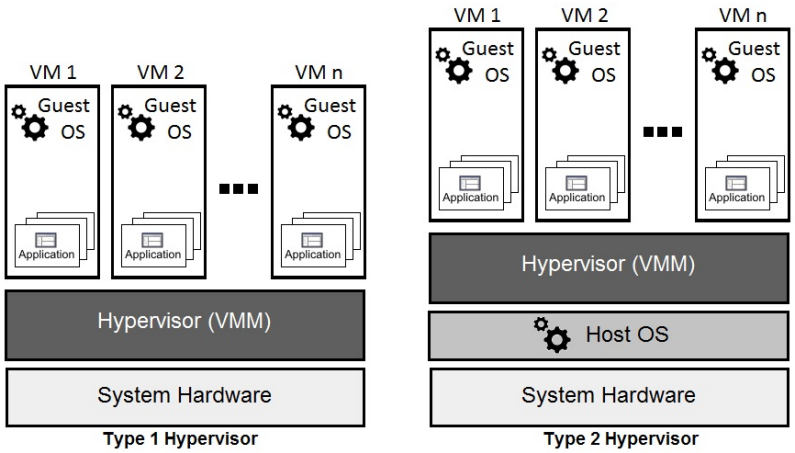


Figure 2.8: Type 1 and Type 2 hypervisors.

Type 1 hypervisor architectures are more robust and, due to the close proximity to hardware resources, they can achieve higher virtualization efficiency than Type 2 hypervisors. Type 2 hypervisors are used mainly on client systems where efficiency is less critical as well as on systems where support for a broad range of I/O devices is important and can be provided by the host operating system [39].

There are also virtualization methods that are classified according to whether or not the Guest OS kernel needs to be modified, *e.g.*, Full virtualization and Para-virtualization, or whether the kernel of an operating system allows for multiple isolated user-space instances, instead of just one, *e.g.*, OS-level virtualization. These methods will be presented in the following subchapters.

2.6.1 Full Virtualization

Full virtualization was fist introduced by IBM with the Virtual Machine Facilityl370 (VM/370) operating system [40]. This virtualization method, illustrated on Figure 2.9, emulates a full hardware environment of a computing node, through binary code translation/rewriting, into independent virtual machines created on top of the hypervisor layer. The binary translations are used to adapt the non-virtualizable instructions into virtualized ones. More recently, the full virtualization method was enhanced with the appearance of the hardware virtualization.

Hardware virtualization provide mechanisms to run unmodified guest virtual machines without the overheads inherent in the standard full virtualization method. It started to be supported by Intel and Advanced Micro Devices (AMD) x86 architecture processor families through the Intel VT-X and AMD Virtualization (AMD-V) hardware assisting virtualization technologies [41][42].

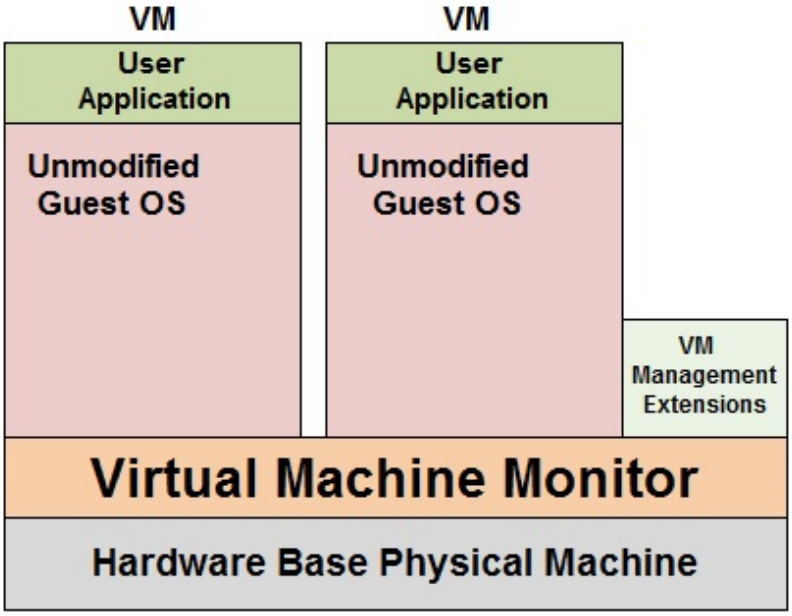


Figure 2.9: Full virtualization (modified from [37]).

Since the VM emulates the full system, the guest operating system does not need to be modified, meaning that both open-source and proprietary OS work with this type of virtualization methodology. The guest OS kernel is not aware of the virtualization and communicates directly to the hypervisor layer as if it is the hardware base physical infrastructure. In this approach, the hypervisor has to manage, control and map the requests from all guest OS to the limited amount of physical resources. As a consequence, the performance experienced is lower when compared to native OS. On the other hand, the actual quantity of hardware resources, provided by the computing node, define the limits to the creation of simultaneous VM. Two examples of known commercial and open-source software tools for server virtualization that implement full virtualization are: VMware and Kernel-based Virtual Machine (KVM) [43][44].

2.6.2 Para-virtualization

In para-virtualization the guest OS kernel is modified to become aware of the hypervisor as illustrated on Figure 2.10. In this virtualization methodology, OS-

level information about the virtual machines can be passed explicitly from the guest OS to the hypervisor without the necessity to trap or translate every OS call.

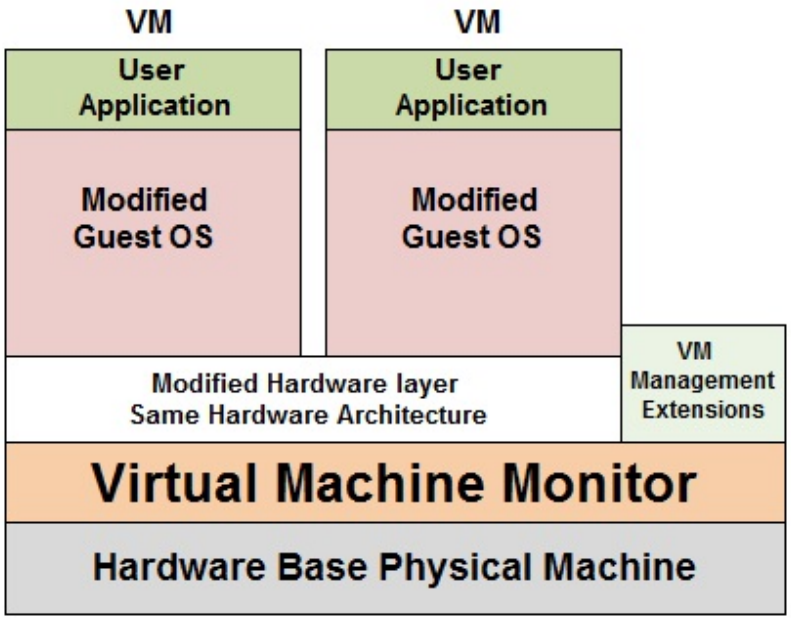


Figure 2.10: Para-virtualization (modified from [37]).

Due to the described behaviour, the para-virtualization approach results in lower virtualization overhead than full virtualization , providing close to native guest OS performance. However, this methodology is less flexible than full virtualization because it requires OS-level modifications, which is a characteristic non implementable with legacy and close-sourced commercial OS. An open-source example of a software tool for server virtualization capable of para-virtualization implementation is Xen [45].

2.6.3 OS-level Virtualization

The operating system level virtualization differs from the other two virtualization methodologies by utilizing the kernel of a host OS embedded with a virtualization layer. This approach allows the creation of multiple user-space instances known as containers, Virtual Private Server (VPS) or Virtual Environment (VE) as illustrated on Figure 2.11. Since normal OS calls are used and no emulation is performed, the container performance is practically native with little imposed overhead. However, this methodology is not as flexible as the other two virtualization approaches since it cannot use guest OS that are different from the host OS, as the OS kernel is shared by the host OS and all guest OS containers. Examples

of software tools capable of OS level virtualization implementation are OpenVZ and Parallels Virtuozzo Containers [46][47].

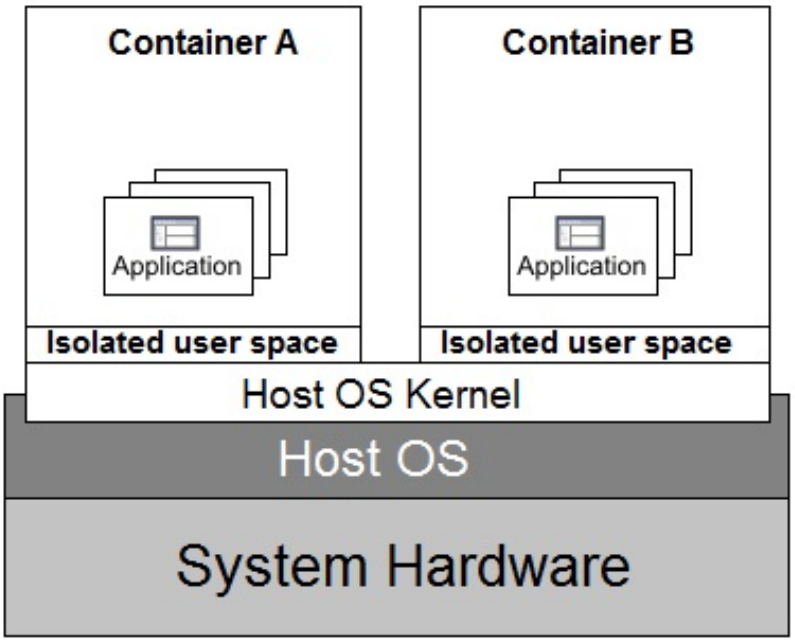


Figure 2.11: OS level virtualization.

2.7 Business Perspective

An important factor that differentiates cloud computing from the other computing technologies is its ability to converge existing technologies with an utility model making the cloud flexible to the adoption of different business perspectives. This approach enables IT efficiency, whereby the power of modern computers is utilized more efficiently trough highly scalable hardware and software resources, as well as business agility, whereby IT can be used as a competitive tool trough rapid deployment, parallel batch processing, use of computing-intense business analysis and mobile interactive applications that respond in real time to user requirements. Thus, cloud attractiveness increases as organizations discover that their substantial capital investments in information technology are often grossly underutilized and that the maintenance and service costs do not pay [48]. On the other hand, the adoption of an utility model enables cloud computing services to reuse the traditional electricity, water, telecommunications price models, including the pay as you go model or metering model, the subscription model and the combination of both.

This chapter will describe the business models used for the three service layers

(IaaS, PaaS and SaaS), the existing stakeholders and the contracted SLA between service vendors and service consumers in the cloud.

2.7.1 IaaS Business Model

The infrastructure layer focuses on enabling technologies with the provision of storage capabilities and computer power through an outsourcing business model. The processing power, use of storage devices, servers and other hardware that composes the IaaS layer are standardized, regulated products allowing the adoption of a metered usage model, *i.e.*, following the utility model.

When cloud providers define the IaaS layer pricing models, they need to consider: (*i*) the availability of the infrastructure and the financial and legal consequences of any disruptions; (*ii*) the readiness of the service regarding both place and time; (*iii*) the friendliness of the consumer interface; (*iv*) the capability to respond to user demand fluctuations; (*v*) the determination of a realistic threshold in order to maintain QoS; and (*vi*) the legal issues (*e.g.*, physical infrastructures that are outside the cloud consumer's national boundaries, involving the provided infrastructure).

All the factors presented need to be considered and their costs included in the price models. This way, the pay as you go model or metering model combined with the subscription model, to allow for more flexibility, are the most frequent models used in the IaaS.

2.7.2 PaaS Business Model

The platform layer offers solutions on top of a cloud infrastructure that provide value-added services from both a technical and a business perspective.

All PaaS solutions are based on integration of SaaS applications in the underlying infrastructure via a cloud-capable PaaS development environment. The largest variant involves extensive middle-ware components. A further aspect of PaaS will give rise to further business model components: billing, metering and monitoring. These components provide the pay as you go price model with the necessary inputs. Creation of full-service platform solutions means that independent software vendors (ISV) and system integrators IT departments can develop and deliver on line applications using third-party infrastructure services [49]. So, the PaaS cloud provider may allow Independent Software Vendors (ISV) to develop and sell their software for a percentage of the sales or for an additional fee for the utilization of the environment platform.

2.7.3 SaaS Business Model

The SaaS layer delivers software services on demand on top of the platform and infrastructure layers (which are transparent to the end user). SaaS can drastically reduce software expenses (*e.g.*, software purchase price or licensing) that have been a major part of IT expenses of companies. It can also offer ISV opportunities to lower their application development costs, reduce their time to market (*e.g.*, utilizing Web stores), extend the service offerings of other vendors (as is the case of SME ISV) and create new software solutions. These opportunities are divided between the larger ISV like Microsoft [50], Google [51] and Facebook [52], which take advantage of their cloud infrastructure to focus on distribution and marketing, and the SME ISV, which build solutions around the offerings of larger ISV to provide specialized services, following the example of the mobile application model.

The SaaS layer provides a diverse number of applications and, so, the business approach has to be more flexible. In terms of payment model, the pay as you go, the subscription model, the "freemium" and free models are utilized to promote new products and services or to explore new ways to attract customers. The subscription model is the most frequently adopted model.

The "freemium" model is a combination of the free model and the subscription model. It offers a limited free subscription, temporal or with service limitations, and an upgrade to a premium subscription version where all the program's features are available. The free version helps to demonstrate the product while it intends to encourage users to upgrade to the premium version. On the other hand, the free model is supported most exclusively by advertising where the profit is obtained trough the number of clicks or views per minute.

2.7.4 Stakeholders

In the previous computing paradigms the main stakeholders were the system providers and their consumers. Whereas the providers were responsible for the sales, installation, licensing, consulting and maintenance of the involved technologies, the consumers used, maintained and owned the provided systems.

With the technological shift to the cloud, some of the previously described functions of these stakeholders changed and due to the outsource characteristics of the cloud deployment services two new important parties must be considered in addition to the traditional stakeholders: the cloud enablers and the cloud regulators [48].

In cloud computing the consumers are effectively subscribers that, in comparison with the previous computing technologies, only purchase from the cloud providers the use of the system on an operational expense basis. The cloud pro-

viders own and operate the cloud computing systems in order to deliver third party services. They provide the infrastructure, systems and software, perform the corresponding maintenance and implement the pricing of the cloud services.

Cloud enablers are organizations that sell products or services that facilitate the delivery, adoption, use and management of the cloud infrastructures. This type of stakeholder is an intermediary class that stands between the cloud provider and the service consumer, taking advantage of the currently lake of core competencies from the infrastructure providers in the interaction with the customers. Some cloud enablers examples are RightScale and Vordel [53][54].

The cloud regulators, on the other hand, regulate all aspects of the cloud computing value-chain and are, typically, sovereign government bodies and international entities. Their main functions are the provision of regulations regarding data, residency, privacy and other related cloud computing features, harmonisation of a cross border regulations and promotion of inter-governance cooperation in the adoption and formulation of new cloud computing regulations.

2.7.5 Service Life-Cycle

A service life-cycle describes the different phases of a particular service provided by a cloud vendor and includes the service delivery systems necessary to meet the QoS objectives specified in SLA, *i.e.*, all stages of the SLA life-cycle. The SLA life-cycle is composed of four stages:

- SLA template design - The service provider defines the types of SLA he is willing to propose in order to ensure that the agreed QoS guarantees are realistic;
- SLA negotiation - The service provider and customer attempt to define the terms of the SLA that will bind their business relationship, *i.e.*, that the agreed QoS guarantees are realizable and that the end-to-end QoS requirements are satisfied;
- SLA runtime - The service provider and customer verify that the agreed QoS guarantees are satisfied;
- SLA (template) archiving - The established SLA are stored for future reuse or learning.

On the other hand, the overall life-cycle, represented in Figure 2.12, consists of the following stages:

- Design and development - consists on the development of artifacts needed for the service implementation;

- Service offering - includes the SLA template design, where to offer the intended service and results in the specification of SLA templates;
- Service negotiation - includes parts of the SLA negotiation, represents the actual negotiation between customers and provider and results in an agreed SLA;
- Service provisioning - covers parts of the SLA negotiation, represents all activities required in system preparation and set-up for service operation, including booking, deployment (if needed) and configuration;
- Service operations - includes SLA runtime, *i.e.*, it is a stage where the actual service instance is up and running and where adjustments can be made to enforce a SLA;
- Service decommissioning - represents the end of a service instance (the customer can no longer access it) and corresponds to the stage where the SLA (template) archiving is done.

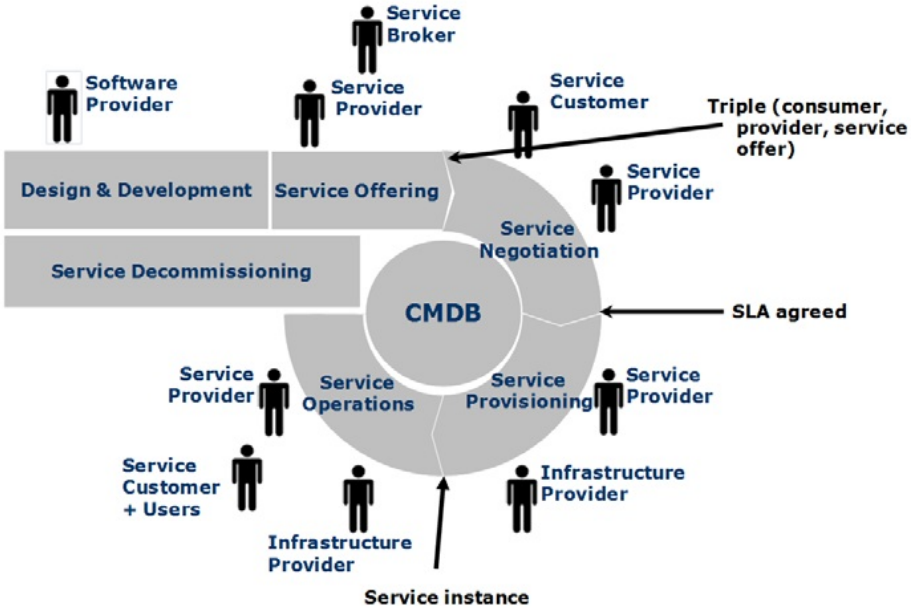


Figure 2.12: Service life-cycle [55].

2.7.6 Service Level Agreements

When consumers adopt cloud computing solutions, the quality and reliability of the provided services are essential features. As previously described in the Concerns section, it is impossible to fulfil all consumer expectations from the

service provider perspective and, hence, a balance needs to be achieved via a negotiation process. At the end of the negotiation, provider and consumer commit to an agreement. In a Service Oriented Architecture (SOA) this agreement is referred to as a Service Level Agreement [55].

The establishment of SLA is frequently adopted in the software and telecommunications domain to specify a mutual understanding regarding the transactions between providers and consumers. Typically, a SLA is a bilateral binding statement signed between a service provider and a service consumer stating the agreed terms and conditions of the given service [9]. Additionally, it also defines the remedial actions and the penalties if performance falls bellow the agreed standard.

Ideally, SLA can be negotiated between any type of service providers and consumers but, in reality, in cloud-based business scenarios, SLA are usually not negotiated in Business to Consumer (B2C) and in Business to Business (B2B) transactions involving small clients (*e.g.*, SME) [56].

A SLA acts as a legally enforceable document traditionally composed of:(*i*) subject terms; (*ii*) scope of rights; (*iii*) financial terms; (*iv*) representation; (*v*) service credits, credit requests and compensation procedures; (*vi*) evolution and support terms; (*vii*) warranty; (*viii*) indemnification; and (*ix*) limitation of liability. Although a standard cloud SLA model does not exist, the subject has been studied by the European Telecommunications Standards Institute (ETSI) [57].

In the Web service domain there are two main specifications for describing a SLA: (*i*) Web Service Agreement (WS-Agreement) from the Open Grid Forum (OGF) [58]; and (*ii*) Web Service Level Agreement (WSLA) language and framework from IBM [59].

The WSLA was proposed in 2001 and it allows the creation of machine-readable SLA for services implemented using Web services technology that define service interfaces in the Web Services Description Language (WSDL). This framework comprises several monitoring services that can be replaced with services developed by third parties. An SLA created using the WSLA language is normally formed the following sections:

- Definition of the parties - contains the description of the service provider, the service consumer and, eventually, the third parties involved (when commission part of the SLA management responsibility to additional supporting parties). The information provided about each party should include contact information and a description of the organization as well as the name of a WSDL file that contains the interface description of the Web services the party implements;
- Definition of the services and their operations - contains the description of the service provider interfaces. The services are represented by service ob-

jects and each service object is associated with one or more SLA parameters. This section specifies the relevant service level parameters and respective metrics as well as indicates which party is responsible for measuring each metric and how the measuring should be done;

- Obligations of the agreement - specifies the conditions (*e.g.*, the average operation response time shall not exceed a predefined value) and the action guarantees, *i.e.*, the actions the parties commit to perform in a given situation.

On the other hand, the WS-Agreement specification appeared in 2007 and defines an XML-based language for agreements as well as a protocol for advertising the capabilities of service providers, creating agreements between service consumers and providers and monitoring agreement compliance [58]. WS-Agreement also provides an eXtensible Mark-up Language (XML) schema that defines the overall structure of an agreement document and, in addiction to the WSLA, defines a protocol for negotiating and establishing agreements dynamically based on Web services. The main differences to the WSLA are the use of: (*i*) SLA templates that embody all the customization aspects of an agreement; (*ii*) an extensible XML structure containing several sections where the intended users are expected to define domain-specific elements and properties; and (*iii*) metrics defined by a domain-specific extension rather than associated with the parameters used in the agreement. The XML document of the WS-Agreement consists of the following parts:

- IDentification (ID) - holds an unique mandatory agreement ID followed by an optional name;
- Parties - contains the identification of the different parties, various metadata about the agreement (such as an expiration time and template ID in case the agreement was created following a template) and user-defined attributes;
- Interface declarations - defines all the information about the functional capabilities of the service. It encapsulates the information contained in traditional service description (*e.g.*, WSDL documents);
- Agreement terms - specifies the service terms and guarantee terms. Service terms identify and describe the services that are the subject of the agreement and also comprise the measurable service properties and exposed properties that can be used to express service level objectives. Guarantee terms specify the QoS that the parties are agreeing to, *e.g.*, minimum availability of a service, the number and type of CPU that the service provider should make available or the average response time for requests.

An SLA template document has the same structure as an SLA, but is not instantiated.

2.8 Conclusions

Cloud computing represents a switch from the in-home generated computing approach to the utility-supplied computing resources delivered over the Internet as Web services. This new IT paradigm is essentially a business model supported by the aggregation of several well developed technologies, *i.e.*, resource virtualization, Internet technologies (*e.g.*, Web services, SOA and Web 2.0), distributed computing (*e.g.*, cluster and grid computing) and automated management systems (*e.g.*, data centre automation).

The exact definition of Cloud Computing is not consensual among the IT institutions and scientific community. However, the NIST definition provides a common framework for the discussion of this subject. The NIST Cloud Computing features enable the adoption of different business perspectives, according to the utility, service and deployment models used, introducing advantages to the cloud consumers, mainly Small and Medium Enterprises. The reliance on third-party service providers is a major issue, *i.e.*, organizations and enterprises contemplating the migration to the cloud are concerned with data privacy, security and unpredictable performance. In terms of service features, the Service Level Agreements are essential to define the duties and obligations, *i.e.*, the service contract, binding cloud consumers and service providers. However, standard cloud consumers are offered pre-defined non-negotiable SLA, favouring the service providers.

Normally, large enterprises prefer to build and manage the cloud IT infrastructure themselves. They tackle the identified concerns by deploying private clouds that eventually evolve into hybrid ecosystems, using the public cloud to expose service offerings that amortize the investment on the IT infrastructure.

The following chapter analyses and compares different software solutions that use different cloud IT infrastructures for the provision of IaaS.

Chapter 3

Cloud Infrastructure Platforms

Physical and virtual resources (*e.g.*, servers, storage and networks) present a challenge for IaaS providers when building a cloud infrastructure as their organization and availability must be ensured to allow a rapid and dynamical provision of resources to the end application. This resource orchestration is done by software tool-kits composed of a Cloud Infrastructure Manager (CIM) with remote and secure interfaces for creating, controlling and monitoring virtualized resources on a IaaS cloud, and a Virtual Infrastructure Manager (VIM) that provides primitives to schedule and manage virtual machines across multiple physical hosts. Below the VIM, there are the hypervisors that provide simple primitives (*e.g.*, start, stop, suspend) to manage the virtual machines life-cycle of one physical host (node).

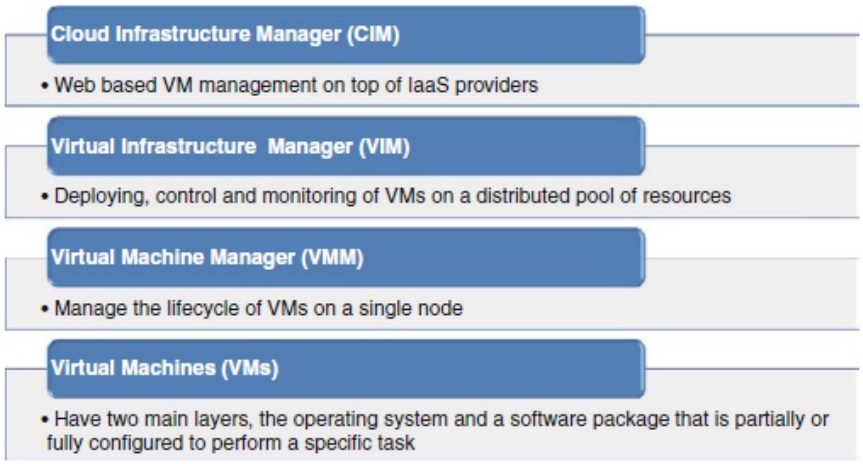


Figure 3.1: VM management layers of a cloud infrastructure platform [37].

Through these management stacks, illustrated on Figure 3.1, physical resources are abstracted into a virtualized pool of resources accessible by the cloud consumer through virtual machines. Thus, CIM and VIM as well as the hypervisors (virtual machine managers) are the essential set of tools to deploy cloud IaaS platforms from the organizations IT physical infrastructures.

The software tool-kit that implements CIM and VIM aggregates resources from multiple nodes, presenting a uniform view to the user and applications. The features available to manage VM can either be basic or advanced depending on the type of the software, *i.e.*, if it is focussed on the cloud infrastructure management or on the virtual infrastructure management (presenting more advanced features to manage virtual machines). From a general point of view, software tool-kits can offer the following main features [25]: (*i*) Virtualization Support; (*ii*) Self-Service, On-Demand Resource Provisioning; (*iii*) Multiple back-end Hypervisors; (*iv*) Storage Virtualization; (*v*) Interface to Public Clouds; (*vi*) Virtual Networking; (*vii*) Dynamic Resource Allocation; (*viii*) Reservation and Negotiation Mechanism; and (*ix*) High Availability and Data Recovery.

The first commercial cloud infrastructure that contributed to the popularization of the IaaS paradigm was arguably initiated by Amazon's public Elastic Compute Cloud (EC2) [60] back in 2006, offering virtual machines for US $0.10/h, using both a simple Web interface and a programmer-friendly API. Although the initial ecosystem was focussed around the utilization of public clouds, the appearance of open source tools enabled organizations to build their own IaaS clouds utilizing their internal infrastructure, increasing the appearance and the deployment of private clouds [61].

Due to the lack of standardization, the architecture, features and interface compatibility (either with the underlying hypervisors or the manager interface from other clouds) varies. This way in the following sub-chapters are presented, characterized and compared four different open source and one proprietary (used by LunaCloud) solutions for the deployment of cloud infrastructure platforms: (*i*) OpenNebula [62]; (*ii*) OpenStack [63]; (*iii*) CloudStack [64]; (*iv*) Eucalyptus [65]; and (*v*) PACI [66]. The comparison of the referred tools will also by important for the identification of common features that will enable the development of a management solution that meets the scope of this thesis.

3.1 OpenNebula

OpenNebula started as a research project by Distributed Systems Architecture (DSA) research group [67] and it was released in 2008 as a completely open source software solution. It was originally conceived to manage local virtual infrastructures, but the inclusion of remote interfaces made it also available to

build public and hybrid clouds, turning OpenNebula into one of the most feature-rich open source solutions for the deployment of cloud computing IaaS platforms [68][25]. Recently, it is developed, maintained and distributed by the Cloud Computing (C12G) Labs organization [69] that follows a benevolent dictator governance model[1].

3.1.1 Architecture

OpenNebula is at the moment on the 4.0 release. Its architecture is modular allowing integration with different storage, network infrastructure configurations and hypervisor technologies. Figure 3.2 presents a diagram with the the components organized in three layers: (*i*) Drivers layer; (*ii*) Core layer; and (*iii*) Tools layer. These components communicate via a set of API that support system and cloud end user interfaces. The API are not presented in Figure 3.2 for simplification purposes (they are illustrated in Figure 3.3).

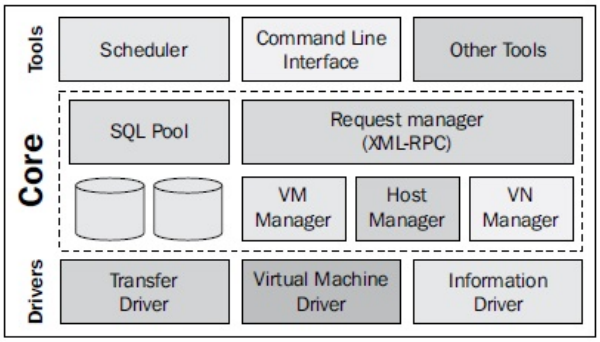


Figure 3.2: OpenNebula architecture [68].

The core layer is composed by modules that manage the virtual machines, the hosts and the virtual networks as well as a request manager to interact between the user requests and the resource managers via Remote Procedure Call (RPC). The RPC calls are transported over the HyperText Transfer Protocol (HTTP) and encoded in XML. Structured Query Language (SQL) databases, *e.g.*, SQLite [70] and MySQL [71], are also part of the core components and they are responsible for the storage of all the monitoring information collected from physical hosts, VM instances, VM configurations, available disk images and virtual networks. The OpenNebula core is written in a highly optimized C++ programming language to improve the system performance and scalability.

The lowest layer provides modules containing drivers to communicate with the underlying host OS components (the hypervisors, virtual networks, file systems,

[1]Focuses on the interest of the project and strategically leads it to ensure that it meets the needs of the users and the community

host information). These drivers, which are used to connect OpenNebula's core to the hosts physical system, are divided in the following tree categories:

- Transfer drivers - used to manage shared disk images, *e.g.*, Network File System (NFS) or internet Small Computer System Interface (iSCSI), and non-shared disk images, *e.g.*, simple copies over Secure Shell (SSH), on the current storage system;
- VM drivers - used to interact with the specific supported hypervisors to manage the virtual machine instances on the current hosts;
- Information drivers - used to retrieve the current status of virtual machine instances and hosts. They are copied and remotely executed via SSH in every physical host, according to the specifications of the host hypervisor.

On the other hand, the top layer provides tools to remotely interact with the end users, channelling the requests to the core management systems. The tools that can be attached to this layer are end-user interface-related. They can be Web interfaces, CLI interfaces or other standard types of interfaces to enable hybrid and/or public cloud interactions (*e.g.*, EC2 tools compatibility). OpenNebula also comes with a scheduler (separated from the core management function) to deploy VM on host nodes following specific user requirements and resource-aware policies. An ecosystem catalogue is also provided in the community wiki Web page [72], containing tools, extensions and plug-ins that enhance the functionality provided by the standard tool-kit or enable its integration with existing products.

3.1.2 Interfaces

OpenNebula provides a set of interfaces to interact with multiple end-user tools and provider infrastructure configurations. Through them, it is possible to manage different node hypervisor technologies at the same time as well as ensure the compatibility and interaction between tools required for the provision of public clouds or drivers to interact with other cloud infrastructures. OpenNebula's interfaces are divided into two categories: (*i*) End-user cloud interfaces; and (*ii*) System interfaces. Figure 3.3 exposes the provided interfaces and API in OpenNebula's architecture as well as some of the supported tools.

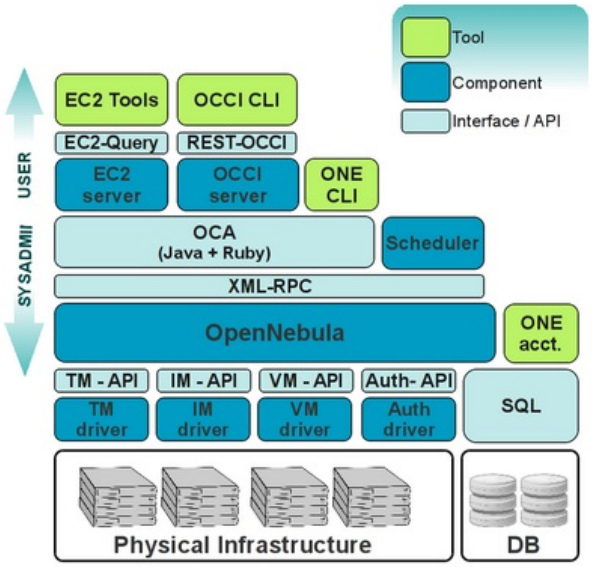


Figure 3.3: OpenNebula interfaces [73].

Cloud interfaces are useful as they provide partners or external users with access to the infrastructure. Providers convert private or hybrid clouds into public clouds through the exposure of REpresentational State Transfer (REST) Web API. OpenNebula provides support for the integration and development of RESTful cloud interfaces and includes software modules that implement two standard cloud interfaces:

- The EC2 Query API - implements the functionalities offered by the Amazon's AWS EC2 API [74] to interact with EC2 Query tools [75] and, thus, perform VM management tasks;
- The OGF OCCI API - is an OGF [76] Open Cloud Computing Interface (OCCI) API [77] that addresses the interoperability and extensibility. It enables the use of a OCCI server as a Web service to create, control and monitor cloud resources on an OpenNebula platform [78].

System interfaces, on the other hand, are utilized to enable the interconnection between different modules of the OpenNebula's architecture as well as to provide compatibility capabilities with third party technologies. OpenNebula offers the following system interfaces:

- XML-RPC interface - exposes all the functionalities to interface with the OpenNebula deamon. Using the XML-RPC interface it is possible to control

and manage any resource (*e.g.*, VM, networks, images, users, hosts and clusters). The XML-RPC interface is the primary interface for OpenNebula;

- OpenNebula Cloud API (OCA) - provides a simplified way to interface the OpenNebula core, exposing the same functionalities as the XML-RPC interface via Java and Ruby programming language bindings. This interface allows the integration of developed advanced IaaS tools that need full access to the software main functionalities;
- Drivers interfaces - allow the integration between OpenNebula and the cloud infrastructures. Drivers interfaces are utilized for storage (permitting other programs implementations to interface special storage back end and file systems), virtualization (allowing the integration with different hypervisors), monitoring (enabling the inclusion of additional external probes to monitor information) and authorization (use of external programs to authorize and authenticate user requests). OpenNebula provides drivers for VMware [43], KVM [44] and XEN [45] hypervisors;
- DataBase (DB) interfaces - enables the use of MySQL [71] or SQLite [70] as DB tools.

3.1.3 Platform Deployment

OpenNebula's installation follows a cluster-like architecture with a front-end, a set of hosts where the virtual machines will be executed (one host can provide a cluster of virtual machines), storage (data-stores) and physical networks that connect all components represented in Figure 3.4.

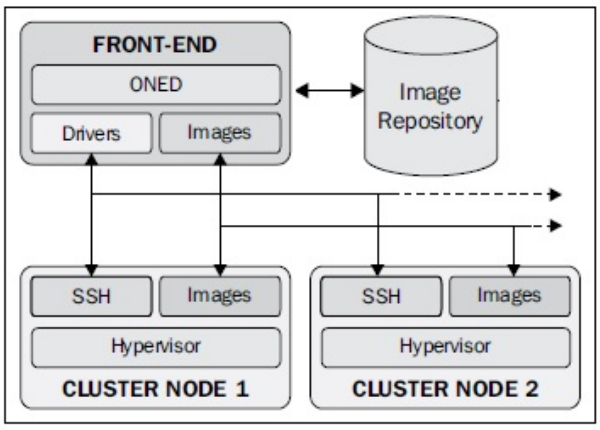


Figure 3.4: OpenNebula system (modified from [68]).

The front-end server, which can also be utilized to execute virtual machines in smaller configurations, is where the software is installed. It contains the manage-

ment daemon (called ONED), the scheduler, the monitoring and account daemon, a Web interface server (called Sunstone) and the cloud API servers (EC2-query and/or OCCI). These components communicate together through an XML-RPC interface and can also be installed in different machines for security or performance reasons. The front-end and the host machines also need to have the Ruby 1.8.7 software installed (or a more recent version). The front-end machine must be able to communicate with all the other hosts and has access to the network storage mounts.

The hosts are managed directly by the OpenNebula daemons running on the front-end platform, using SSH for communication. There is no need to install any particular OpenNebula package on the hosts. The only software requirements are the SSH server, a compatible hypervisor and the Ruby 1.8.7 software. The image repository and storage is used to manage the virtual machine image files and has to be accessible through the front end using any suitable technology such as: (*i*) Network-Attached Storage (NAS); (*ii*) Storage Area Network (SAN); (*iii*) Direct Attached Storage (DAS); or (*iv*) any GNU/Linux distributed-network File System (FS). The Image Repository has to be large enough to store all the VM images of the provider infrastructures. When a VM is initiated, it can be configured to work with cloned copies of master images or directly use an image available in the repository. Shared storage between hosts is optional.

The OpenNebula front-end daemons require network connection with the hosts to manage and monitor the hypervisors and move image files. To offer network connectivity to the VM across the different hosts, the default configuration connects the virtual machine network interface to a bridge in the physical host. When a new virtual machine is launched, OpenNebula will connect its network interfaces to the bridge or physical device specified in the virtual network definition, allowing the VM to have access to different networks, public or private. To ensure that an user can only interact with his VM and, thus, restrict the network access to the owned Virtual Machines, the following secure mechanisms are implemented:

- Firewall - Firewall rules are applied, but networking isolation is ignored;
- 802.1Q [79] - restrict network access through VLAN tagging, which also requires support from the hardware switches;
- Ebtables - restrict network access through Ebtables rules. No special hardware configuration required;
- Open vSwitch - restrict network access with Open virtual Switch (vSwitch) [80];

- VMware - uses the VMware networking infrastructure to provide an isolated and 802.1Q compatible network for virtual machines launched with the VMware hypervisor [43].

3.1.4 Authentication and Authorization

OpenNebula comes by default with an internal user/password authentication and an Access Control List (ACL) authorization system. The authentication and authorization is handled by the OpenNebula Core and can also be delegated to an external module. Any interface to OpenNebula (CLI, Sunstone, Ruby or Java OCA) communicates with the core using XML-RPC containing the user's session string (which is authenticated comparing the username and password with the registered users). Each operation generates an authorization request that is checked against the registered ACL rules granting permission or rejecting the request. OpenNebula also includes a user and group management system installation that can be classified in four types:

- Administrators - an admin user belongs to the oneadmin group and can perform any operation;
- Regular users - a regular user accesses most OpenNebula functionalities;
- Public users - a public user accesses only the basic functionalities (and public interfaces) that are open to public users;
- Service users - a service user account is used by the OpenNebula services (*e.g.*, EC2 and OCCI API, Sunstone) to proxy authorization and authentication requests.

It is possible to customize other authentication and authorization mechanisms such as the Rivest r., Shamir a. and Adleman l. (RSA) keypairs over SSH, X509 certificates [81], Lightweight Directory Access Protocol (LDAP) or Active Directory (AD).

3.2 OpenStack

OpenStack is a project managed since 2012 by the OpenStack foundation [63]. It was initiated in 2010 through the collaboration and the code contribution of Rackspace cloud files platform [82] and the National Aeronautics and Space Administration (NASA) open-source cloud computing project named Nebula [83]. The OpenStack project aims "to produce the ubiquitous Open Source Cloud Computing platform that will meet the needs of public and private clouds regardless of size, by being simple to implement and massively scalable." [84].

3.2.1 Architecture

The OpenStack architecture is continuously changing between releases. The first release named Austin combined only an object store and a compute module (with basic management of VM, networks and authentication/authorization procedures) and a simple support for a virtual machine image registry service as well as a preview Web control panel. Over the following years some functions provided by the legacy components become deprecated and migrated to new project components (with new and enhanced features) providing a more modular and balanced architecture as well as a more complete IaaS platform implementation. The current software release of OpenStack is named Grizzly and its architecture provides the following seven interrelated and independently developed components (divided into computing, networking and storage functions): (*i*) Compute module named Nova; (*ii*) Object storage named Swift; (*iii*) Block storage named Cinder (*iv*) Image service module named Glance; (*v*) Network service module named Neutron (with the previous name of Quantum); (*vi*) Identity module named Keystone; and (*vii*) Dashboard named Horizon.

All of the constituent modules are written in Python programming language and a simplified architecture with basic interactions of the referred components is presented in Figure 3.5.

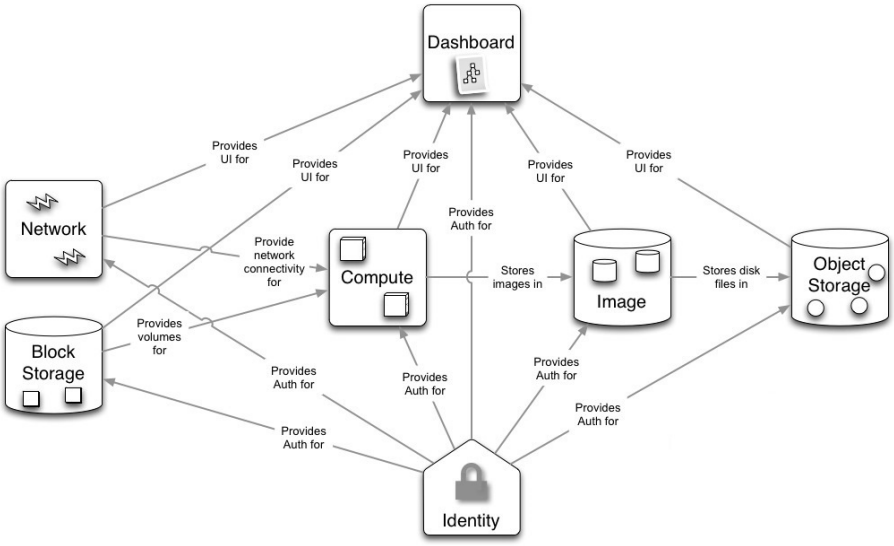


Figure 3.5: OpenStack conceptual architecture [85].

The compute functionalities are implemented through the combination of Nova and Glance. Nova implements both cloud management and virtual resource management functions, being the core of the OpenStack cloud. It manages the communication with the hypervisor and the VM, the authentication and author-

ization as well as standard network functionalities for the VM (IP forwarding, bridges and VLAN). Nova also provides interaction with external cloud administration tools (*e.g.*, Web Dashboard, CLI) and contains a scheduler which distributes the VM through the hosts (defines where a new instance must be created). The communication between Nova components is made via RPC (both request and response) over an Advanced Message Queue Protocol (AMQP). The OpenStack image service, Glance, provides functionalities for discovering, registering and retrieving virtual machine images. Furthermore, it allows users to query the VM image metadata stored in a SQL DB (either MySQL [71] or SQLite [70] can be used) and, then, to retrieve the actual image using HTTP requests. The VM images made available through Glance can be stored in a variety of locations. The image service supports the following back end stores:

- OpenStack Object Storage - OpenStack Object Storage (code-named Swift) is the highly-available object storage project within OpenStack;
- File System - The default back end that Glance uses to store virtual machine images is the FS back end. It writes image files to the local file system;
- Simple Storage Service (S3) - This back end allows Glance to store virtual machine images in Amazon's S3 service [86];
- HTTP - Glance can read virtual machine images that are available via HTTP somewhere on the Internet (this store is read only);
- Rados Block Device (RBD) - This back end stores images inside of a Ceph storage cluster using Ceph's RBD interface [87];
- GridFS [88] - This back-end stores images inside of MongoDB [89]

The network functionalities of OpenStack can be managed with the Neutron component instead of the Nova network manager. This module provides flexible networking models to suit the needs of different applications or user groups. The original OpenStack Nova network implementation assumed a very basic isolation model through Linux VLAN and Internet Protocol (IP) tables. OpenStack Networking introduces the concept of a plugin, which enables the utilization of basic Linux VLAN and IP tables, as well as more advanced technologies, *e.g.*, Layer 2 (L2) in Layer 3 (L3) tunneling or OpenFlow [90]. The current set of available plugins include: (*i*) Open vSwitch [80]; (*ii*) Cisco products [91] ; (*iii*) Linux Bridge; and (*iv*) Open Networking Foundation (ONF) OpenFlow [90]. Typically, the plugins require access to a database for persistent storage, similar to other OpenStack services. OpenStack Networking also includes additional agents that might be required depending on the intended deployment:

- Plugin agent - runs on each hypervisor to perform the local vSwitch configuration. Depending on the plugin used, a different type of agent or no agent at all, as some plugins do not require an agent, is launched;

- Dynamic Host Configuration Protocol (DHCP) agent - Provides DHCP services to tenant networks (is the same across all plugins);

- L3 agent - Provides L3/ Network Address Translation (NAT) forwarding for external network access to VM on tenant networks (is the same across all plugins).

The storage functions in OpenStack are performed by the object storage component Swift and the block storage component Cinter. Swift offers cloud storage software based on the Cloud Files of Rackspace. The Swift architecture is highly distributed to prevent any single point of failure as well as to scale horizontally. It includes:

- A proxy server to accept incoming requests (via the OpenStack Object API or just raw HTTP), *i.e.*, data upload, metadata update or container creation as well as data download or container listing to Web browsers;

- Account servers to manage accounts defined with the object storage service;

- Container servers to manage a mapping of containers (*i.e.*, folders) within the object store service;

- Object servers to manage actual objects (*i.e.*, files) on the storage nodes.

Swift also ensures consistency and availability through the implementation of periodic processes which run to perform housekeeping tasks on the large datastores like replication services, auditors, updaters and reapers. On the other hand, Cinter is similar to the Amazon's Elastic Block Store (EBS) [92]. It provides persistent block storage to guest VM and uses a SQL central database that is shared by all Cinder services. The block storage is appropriate for performance sensitive scenarios such as database storage, expandable file systems or to provide servers with access to raw block level storage. It enables snapshot management for backing up data stored on block storage volumes. Snapshots[2] can be restored or used to create a new block storage volume. In addition to local Linux server storage, it can use other storage platforms.

[2]Backups of the state of a system at a particular point in time.

3.2.2 Interfaces

OpenStack is, opposed to OpenNebula, an aggregation of different projects rather than a single project and, as a result, the platform interfaces have a more complex representation. Figure 3.6 illustrates the complexity of the global architecture of OpenStack as well as the interactions between different project components through their underlying modules. Every component has its own internal architecture that exposes interfaces as well as specific modules to (*i*) interact with other project component interfaces and cloud control tools; (*ii*) manage messaging distribution; (*iii*) communicate with the hypervisor nodes; and (*iv*) enable aggregation of network plugins. As in OpenNebula, the provided interfaces can be grouped into two categories: end-user cloud interfaces and system interfaces.

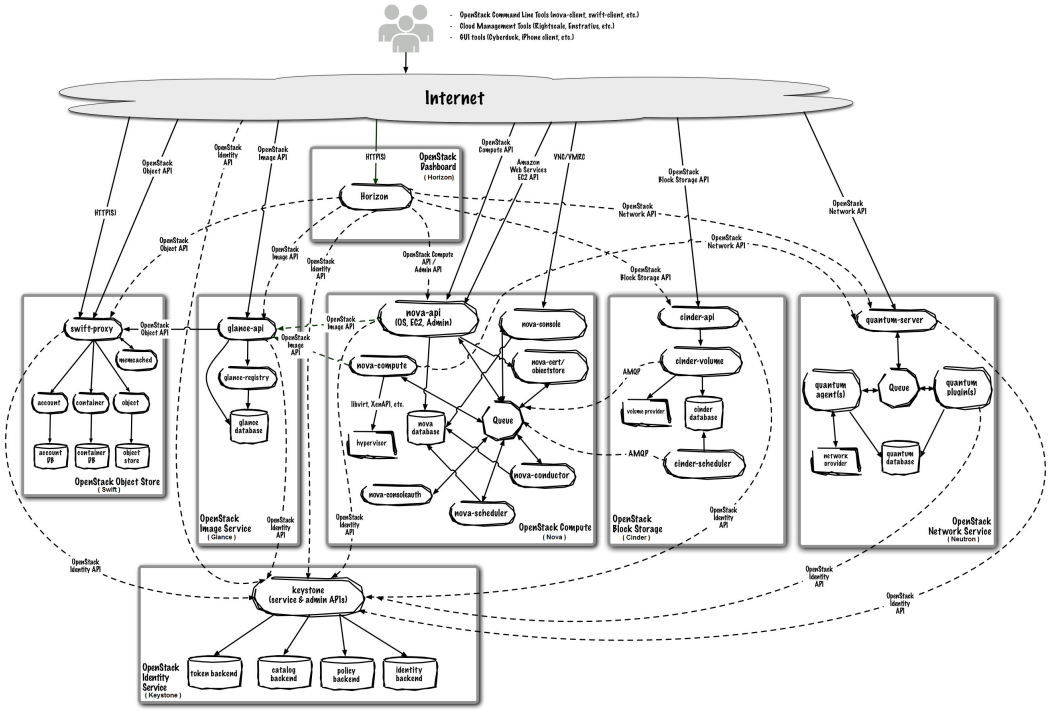


Figure 3.6: OpenStack Grizzly global architecture (modified from [93]).

OpenStack provides command line, Web based and API based instance management through the Nova component. It has a native RESTful API, a console interface for issuing commands through a Virtual Network Computing (VNC) connection to run instances and a compatible EC2 API. The native API interacts with the provided dashboard Horizon and using a Web Server Gateway Interface (WSGI) framework is also possible to provide an OCCI interface to support the OGF standard [94]. On the other hand, the EC2 API provides support for Amazon's EC2 tools. It is also possible to emulate the Amazon's S3 API

through the installation of a middleware on top of the object storage block Swift [95].

The offered system interfaces implemented in OpenStack enable the interconnection between the project components, the communication between internal component modules and third party technologies. The following list of interfaces are supported:

- OpenStack native API - includes the identity API (Keystone), compute API (Nova), image service API (Glance), block storage service API (Cinder) and object storage API (Swift). They interact through REST calls to, respectively, provide authorization tokens to access the compute API, VM launch based on the images uploaded as well as establishment of secure connections to server instances through SSH, management (*i.e.*, create, update and delete) of virtual image metadata records and the upload/download of raw image data as well as the image loading to the compute API, the management of volumes and snapshots for the block storage service and the management of accounts, containers and objects in the object store system;
- AMQP Broker - handles the interaction between services. It can be implemented using RabbitMQ [96] or Qpid [97]. The AMQP broker stands between any two Nova, Cinder and Neutron internal components and allow them to communicate via RPC over AMQP;
- Driver Interfaces - allow OpenStack to be compatible with third-party technology solutions and different infrastructure setups. Cinder uses by default a Logical Volume Manage (LVM) on a local volume group, but also provides drivers to add support for other storage devices. A list of all supported volume drivers can be obtained from the OpenStack Block Storage Service Administration Guide [98]. On the other hand, Nova interacts with the underlying hypervisors through drivers using the internal Nova-compute module. OpenStack is compatible with KVM [44], XEN [45], VMware [43], Hyper-V [99] and other hypervisors. A complete list can be obtained from the OpenStack Compute Administration Guide [100];
- Database interfaces - enables the use of MySQL [71] or SQLite [70] as DB tools through HTTP with the open source SQL toolkit and the Object-Relational Mapper (ORM) SQLAlchemy [101].

3.2.3 Platform Deployment

OpenStack installation depends on the purpose of the intended IaaS platform. The installation of all the referred project components is not required for the implementation of a standard platform. Unlike the OpenNebula deployment, the

OpenStack suite installation is divided between the chosen OpenStack project components. A straight-forward installation involves a three node set-up (controller, network and compute nodes) - see Figure 3.7.

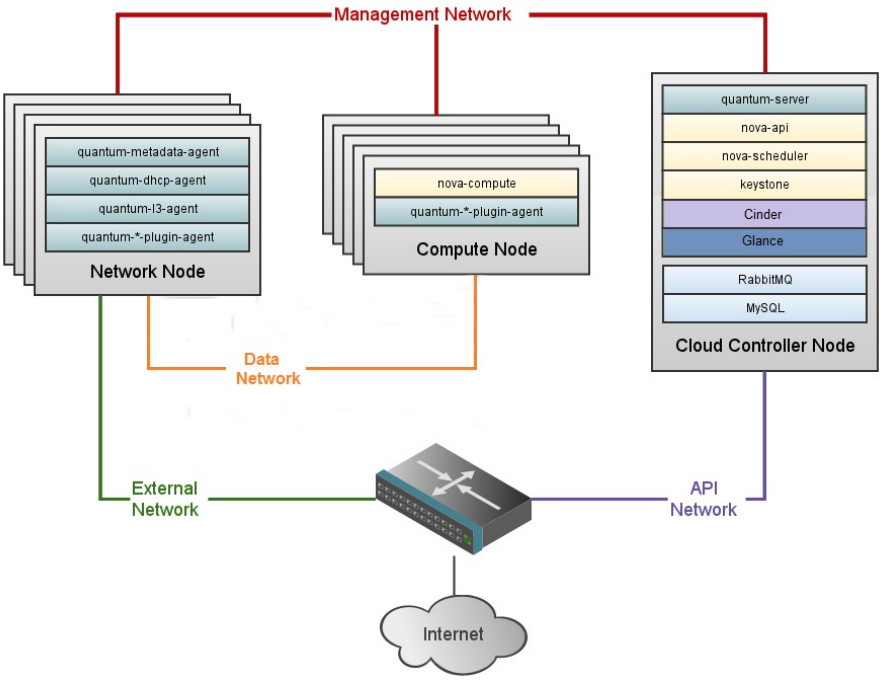


Figure 3.7: OpenStack nodes, networks and components (modified from [102]).

The cloud controller provides the central management system for multi-node deployment. Typically, the cloud controller manages authentication and sends messages to all systems through a message queue. It includes the image service Glance, the block storage service Cinder, the identity service Keystone, the dashboard Horizon, portions of the compute service Nova (API servers, scheduler, conductor, console authentication and VNC service) and the API endpoint for the OpenStack network service Neutron. The compute node (or compute nodes as there can be more than one server node) has the resources to provide virtual machines. It includes the Nova internal module compute for interaction with the hypervisor, the network service agent (*e.g.*, the Open vSwitch plug-in agent) and a compatible hypervisor. The network node provides the bulk of the OpenStack services (*e.g.*, DHCP, L2 switching, L3 routing, floating IP and metadata connectivity) as well as the Neutron component project functionalities.

A standard network set-up can have up to four distinct physical data centre networks, which can also be combined and re-used, *e.g.*, the Management, Data, and API networks are commonly in the same network. The management network is used for internal communication between OpenStack components. The IP

addresses should be reachable only within the platform infrastructure. The data network is used for VM data communication within the cloud deployment and the IP addressing requirements depend on the OpenStack networking plug-in in use. In some deployment scenarios, the external network provides VM Internet access and the IP addresses are accessible through the Internet. The API network exposes all OpenStack API, including the Networking API to tenants (running virtual machines). The IP addresses on this network are reachable through the Internet.

3.2.4 Authentication and Authorization

The authentication and authorization process is performed by the identity service Keystone. Keystone provides the authentication services and maintains user attribute information, which is then used by the other OpenStack services to grant authorization. Users have credentials for authentication and can be a member of one or more groups, *e.g.*, a cloud administrator might be able to list all instances in the cloud, whereas a user can only see those in their current group. The policy applied by the service can also be modified by the configuration of a policy.json file. Keystone supports different plug-ins for back-end authentication decisions and storing information. These range from internal storage choices to external systems and currently include: (*i*) In-memory Key-Value Store; (*ii*) SQL database; (*iii*) Pluggable Authentication Modules (PAM); (*iv*) LDAP and (*v*) X509 credentials.

3.3 CloudStack

CloudStack is a top level project of Apache Software Foundation (ASF) [103]. The project was initiated by a company named Cloud.com [104] that released the first version of CloudStack in May 2010. Most of the software released was under the GNU General Public License version 3 (GPLv3), but a small portion was kept proprietary. Meanwhile, Cloud.com was purchased by Citrix [105] that released the remaining CloudStack proprietary software under the GPLv3 and managed the project for almost one year. In April 2012 Citrix re-licensed CloudStack under the Apache Software License version 2.0 (ASLv2), submitted it to the Apache Incubator [106] and ceased its involvement in the project.

CloudStack is an open source software platform that pools computing resources to build public, private and hybrid IaaS clouds and can be used to deploy, manage and configure cloud computing environments.

3.3.1 Architecture

CloudStack is focussed on the automation and management of various data centres. As a result, the available documentation gives more relevance to the overall cloud platform architecture and features (*e.g.*, data centre network organization of pods and clusters) than to a detailed description of individual software functions (layers). The current Apache CloudStack release is 4.1 (although release 4.2 in imminent). The software is entirely written in Java and it is organized in the following layers: (*i*) Interface; (*ii*) Business logic; (*iii*) Orchestration engine; and (*iv*) Controllers - see Figure 3.8.

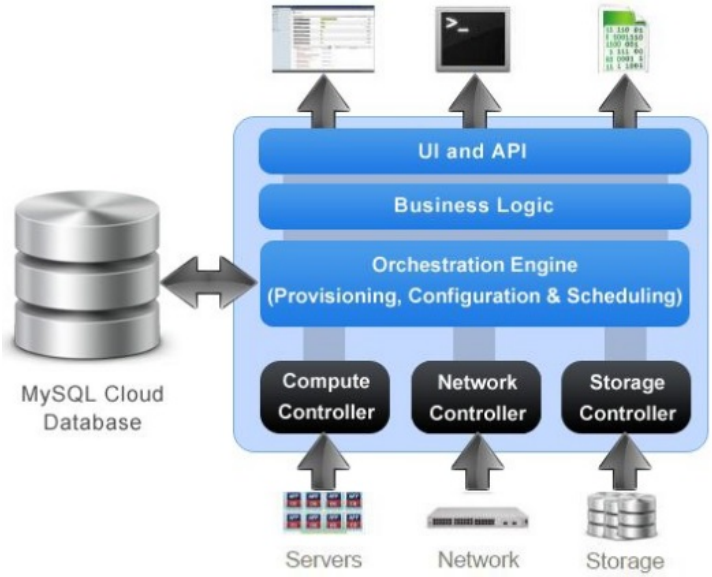


Figure 3.8: CloudStack layers [107].

The interface layer is composed by the User Interface (UI) application and external API, allowing to configure, request, provision and enable programmatic access to the IT infrastructure.

The business layer contains management functions that provide access control and authentication, process workflow to enhance the High Availability (HA) of the service and availability and/or applicability of the required resources such as networks and storage.

The orchestration engine layer is responsible for the management of the virtual infrastructure. It manages the virtual machines, storage, network and it interacts with the data centre Cloud DB (which stores the configuration information of a cluster) to manage and deploy the templates, images and snapshots. It automates the distribution of compute, network and storage resources for the controllers and supplies the business layer with the necessary information for the definition of

policies on load balancing, data security and compliance. This layer also has an asynchronous job manager to manage the scheduling and prioritising of requested tasks.

The controllers enable the communication with the underlying hypervisors as well as network and storage resources. CloudStack uses by default a Virtual Router as network service provider. This Virtual Router implements the following features: (*i*) Remote access through VPN; (*ii*) Firewall protection; (*iii*) Source and static NAT; (*iv*) Load balancing; (*v*) Port forwarding; and (*vi*) DHCP and Domain Name System (DNS). The storage resources are divided into two types:

- Primary storage - is shared among all the hosts of a cluster and is used to host the guest VM instances. It supports SAN (*e.g.*, iSCSI), NAS (*e.g.*, NFS) or DAS, *e.g.*, the linux EXTended file system (EXT);
- Secondary storage - is used to store templates, ISO images and snapshots to be deployed in the IT infrastructure. Every data centre (zone) has at least one secondary storage server that is shared by all the pods (containing the clusters). Unlike primary storage, it only uses NFS like the OpenStack Object storage Swift, NFS storage or Linux NFS servers.

3.3.2 Interfaces

CloudStack provides three different API interfaces: (*i*) Platform API; (*ii*) Agent API; and (*iii*) Plug-in API. The platform API is a native REST end-user cloud interface through which administrators and users control the infrastructure platform. This API is divided into an end-user API, an Operation, Administration, Maintenance & Provision (OAM&P) API, an AWS API, compatible with EC2, and a pluggable service API engine to enable direct management access to third party plug-ins.

The remaining Agent and Plug-in API are system interfaces. The Agent API is JavaScript Object Notation (JSON) based and is used to provide interaction between the CloudStack components and the resource server, where the hypervisor and the hardware resources are located. In the current CloudStack version, the supported hypervisors are VMware [43], KVM [44], Citrix XenServer [108] and Citrix Cloud Platform [109]. The plug-in API allows the insertion of code directly into CloudStack deployments to add or modify the default behaviour of CloudStack. This Java-implemented API defines adapters which expose the functionalities required by CloudStack to implement cloud operations [110]. Through this API it is possible to aggregate third party technologies, *e.g.*, Citrix NetScaler [111]. Figure 3.9 illustrates the integration of the software components with the available plug-ins as well as the platform API.

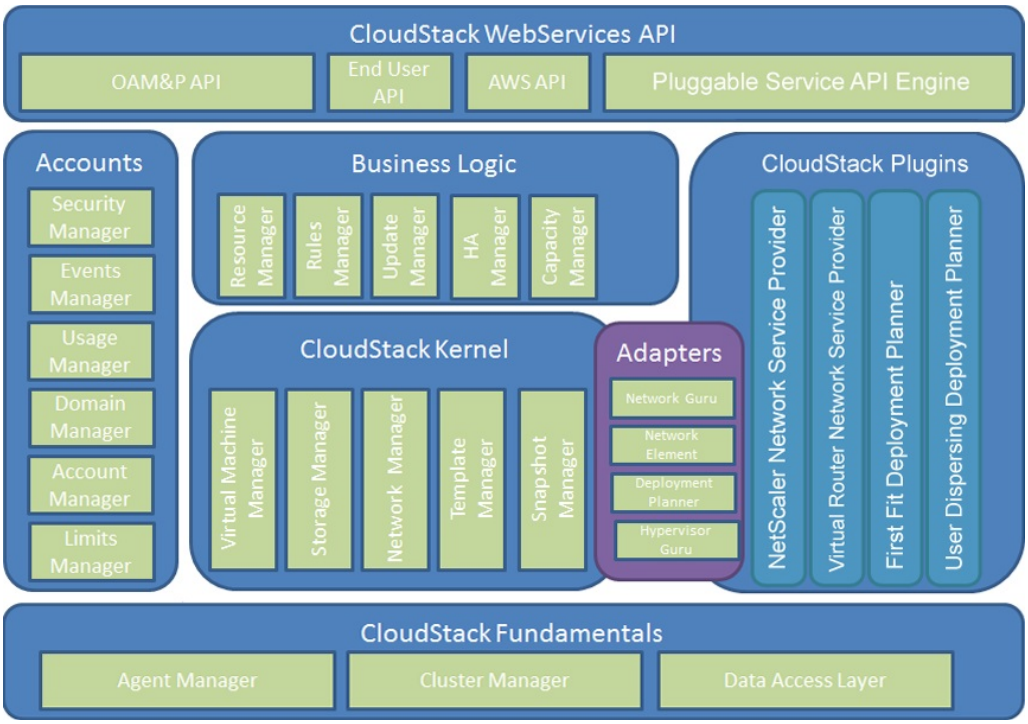


Figure 3.9: CloudStack interfaces (modified from [107]).

3.3.3 Platform Deployment

A CloudStack installation consists of two parts: (*i*) the Management Server; and (*ii*) the managed cloud infrastructure. This infrastructure is hierarchically organized. At the top there is the zone, which is equivalent to a single data centre. It consists of one or more pods isolated by a L3 router and secondary storage. A pod is usually one rack of hardware that includes a L2 switch and one or more hardware servers that provide the VM clusters. The clusters consist of one or more hosts with the same hypervisor and the primary storage. On the other hand, the host is a single compute node within a cluster. The hosts are where the actual cloud services run in the form of guest virtual machines.

The Management Server is where the CloudStack software is installed and it runs on a dedicated server or VM. It controls allocation of virtual machines to hosts and assigns storage and IP addresses to the virtual machine instances. The Management Server runs in a Tomcat [112] container and requires a MySQL database [71] for persistence.

The minimum production installation consists of one machine running the CloudStack Management Server and another machine to act as the cloud infrastructure (in this case, a very simple infrastructure consisting of one host running hypervisor software). A more complex and extended installation consists of a

highly-available multi-node Management Server installation and multiple hosts using any of several advanced networking set-ups. Figure 3.10 illustrates a basic CloudStack deployment architecture.

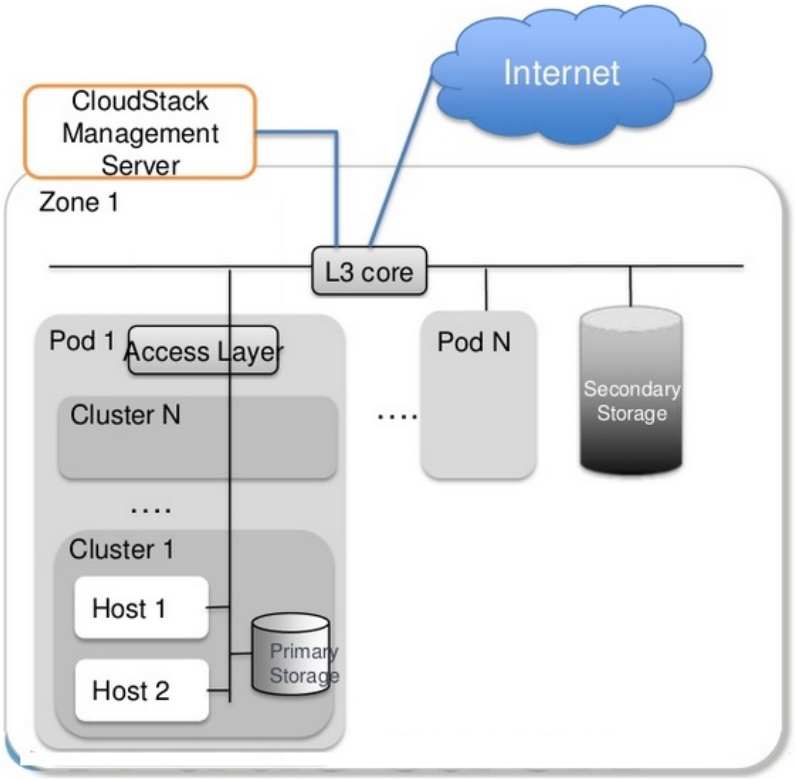


Figure 3.10: CloudStack deployment architecture [113].

CloudStack offers two types of networking scenarios:

- Basic network for AWS type networking - Provides a single network where guest isolation can be achieved through L3 means such as security groups (IP address source filtering);
- Advanced network for more sophisticated network topologies - Provides the most flexibility in defining guest networks.

Each physical network can carry one or more types of network traffic. The basic zone traffic types are:

- Guest - generated when end users run a VM. The guest VM instances communicate with each other over a network that can be referred to as the guest network. Each pod in a basic zone is a broadcast domain and, therefore, each pod has a different IP range for the guest network;

- Management - generated when internal resources communicate with each other. This includes communication between hosts, system VM (VM used by CloudStack to perform various tasks in the cloud) and any other component that communicates directly with the CloudStack Management Server;

- Public - generated when a cloud VM accesses the Internet. Public IP must be allocated for this purpose;

- Storage - generated when data is replicated in the secondary storage. It includes traffic such as VM templates and snapshots, which is sent between the secondary storage VM and secondary storage servers. CloudStack uses a separate Network Interface Controller (NIC) named storage NIC for storage network traffic. The use of this NIC allows fast template and snapshot copying.

3.3.4 Authentication and Authorization

Authentication and authorization is performed by the access control component of the management server. It cross-checks the authorization of the users requesting the action. The user must be authenticated and then mapped to the domain, project or other groups to which he/she belongs. The requested action has an authentication token that authorizes the user to perform the action and specifies the permissions. The actions performed are recorded into logs. In addition to the credentials authentication, CloudStack supports the usage of SSH keys to log in to the cloud infrastructure for additional security as well as of LDAP servers.

3.4 Eucalyptus

Eucalyptus is an open-source software managed by Eucalyptus Systems [65] (following the same governance model approach as OpenNebula) to build private and hybrid IaaS cloud computing platforms compatible with the Amazon Web Services cloud solutions. Eucalyptus started as a research project in the Computer Science Department at the University of California, Santa Barbara, in the fall of 2007. It was initially conceived as a local infrastructure solution to test the integration of the Virtual Grid application Development Software (VGrDS) project, from the National Science Foundation (NSF), with the AWS, which was the most appealing public cloud choice to execute large-scale scientific workflows generated by the NFS's Linked Environment for Atmosphere Discovery (LEAD) weather forecasting system. The first public release of Eucalyptus occurred in May 2008 [114].

3.4.1 Architecture

Eucalyptus possesses a modular architecture composed by software components organized in three layers: (*i*) Cloud layer; (*ii*) Cluster layer; and (*iii*) Node layer. This disposition enables the IT infrastructure fast expansion and consequentially the high availability of the platform resources. Eucalyptus current release 3.3 is comprised of five software components : (*i*) Cloud controller; (*ii*) Walrus; (*iii*) Cluster controller; (*iv*) Storage controller; and (*v*) Node controller. Additionally, a sixth optional component, VMware broker, is also available but only through a subscription service. Figure 3.11 illustrates the software organization of Eucalyptus.

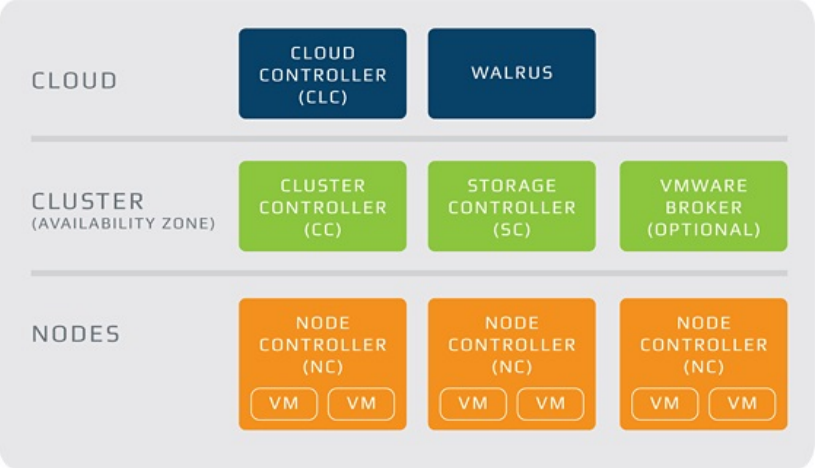


Figure 3.11: Eucalyptus software modules organization [115].

The cloud layer is the platform front end and is formed by the Cloud Controller and the Walrus components. Additionally, software modules associated with administration services can also be included in this layer, *e.g.*, the AWS compatible Identity and Access Management (IAM) API for the enterprise cloud. The Cloud Controller is the entry point into the cloud for administrators, developers, project managers and end users. It is a Java program that queries other software components for information about resources and makes high-level scheduling decisions and requests to the Cluster Controllers. The Cloud Controller is also responsible for exposing and managing the underlying virtualized resources and to handle authentication, accounting and quota management. Walrus, also written in Java, is the Eucalyptus equivalent of AWS S3 [86] and OpenStack Swift object storage services. It allows users to store persistent data, organized as buckets, and objects and it is used to create, delete, and list buckets, put/get and delete objects or to set access control policies in the cloud. There are no data type restrictions for Walrus and it can contain images, volume snapshots

and application data.

The cluster layer is the Eucalyptus middle end and is composed by the Cluster Controller, Storage Controller and, optionally, the VMware Broker [116]. The Cluster Controller is written in C and acts as the front end for a cluster within a Eucalyptus cloud. It gathers information about Node Controllers and schedules virtual machine executions to them. The Cluster Controller also interacts with the Linux host OS and the virtual machine networks, managing the iptables rules (to implement firewalls and address translations), the Linux routing table entries, the DHCP server and the Linux network interface state. The Node controllers associated with a Cluster Controller are in the same sub-network. The Storage Controller provides functionalities similar to the Amazon EBS [92] and OpenStack Cinder and is able to interface with various storage systems, *e.g.*, NFS, iSCSI or SAN devices. It is written in Java and communicates with the Cluster Controller, Walrus and Node Controller to manage the block volumes and to request snapshots of the instances within its specific cluster. The VMware Broker enables Eucalyptus to deploy virtual machines on VMware infrastructure elements, mediating all the interactions between Cluster Controller and VMware hypervisors (ESX/ESXi) [116].

The node layer is formed by the Node Controllers and is the back end of a Eucalyptus platform. The Node Controllers are written in C. They control virtual machine activities (*e.g.*, execution, inspection and termination of VM instances), the host OS, the hypervisors and are also responsible for the management of the virtual network endpoints.

3.4.2 Interfaces

Eucalyptus is an IaaS platform created to test software integration with the Amazon public cloud. Thus, as an AWS compatible platform, Eucalyptus offers both a variety of user interface tools as well as the option to use third party AWS compatible interfaces (*e.g.*, EC2, S3 and EBS API) that interoperate with AWS and Eucalyptus. As Figure 3.12 illustrates, the Eucalyptus cloud interfaces are implemented at the cloud level by the Cloud Controller and Walrus software components. The first (Cloud Controller) exposes a Web interface and a HTTP Simple Object Access Protocol (SOAP) and Query API on top of the Amazon EC2 compatible layer. The Web interface enables account management (*e.g.*, user sign-up and account configuration) and supports basic queries (*e.g.*, listing of images and instances), whereas the SOAP and Query API allow full control of executions, network and storage through command line tools (*e.g.*, Euca2ools [117]) or programming libraries compatible with the AWS EC2 API. The second (Walrus) exposes a SOAP and a REST API on top of the Amazon S3 compatible layer. Using this API, Walrus can store data from all the VM running in the

Eucalyptus cloud and also provide a simple HTTP put/get Storage as a Service solution for users outside the Eucalyptus cloud.

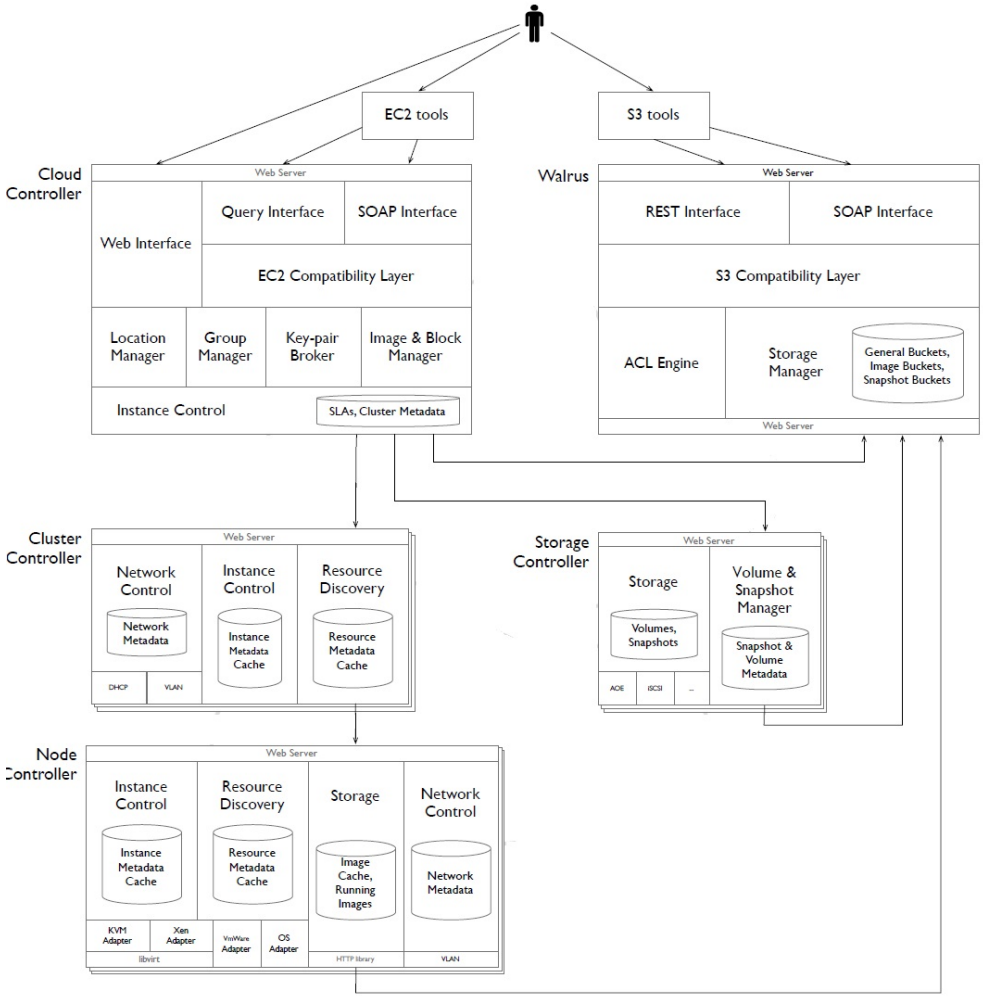


Figure 3.12: Eucalyptus software component interfaces (modified from [118]).

Internally, Eucalyptus uses an EBS that provides a logical communication channel between Web service containers that run in the Web services layer of every software component. These containers take care of messaging integrity, security and dispatch while the Web service layer is responsible for the container life-cycle. This system abstracts communication and location functions, enabling all services to use a common framework that handles secure message routing and delivery regardless of physical location. The Eucalyptus software components uses SOAP and REST API as system interfaces to communicate with other software components running in different physical machines implementing the following steps:

1. The sending service component puts the message on the ESB;
2. The ESB calls the Web service stack to implement the communication;
3. The Web service stack connects to the network and sends the message to the Web server of the desired Eucalyptus service component;
4. The receiving Web server hands the message to the receiving Web service stack to perform the message validation;
5. The receiving Web service stack puts the message on the ESB;
6. The receiving service component gets the message from the ESB.

When the services are located on the same machine they communicate exactly in the same way (they do not need to be specialized for local and remote communication) as the implementation of the ESB short-cuts the message path to bypass the Web service stack and to deliver the message directly when two services are co-located and they are both written in Java (as they run in the same Java virtual machine). If the services are written in different languages, *e.g.*, C and Java, then they must use the full procedure and communicate through local Linux networking via sockets. On the other hand, the communication with the hypervisors is done using drivers that provide support to XEN [45] and KVM [44] (support to VMware hypervisors [43] is also available through the implementation of the optional software component VMware Broker [116]).

3.4.3 Platform Deployment

Like the previous software solutions, Eucalyptus installation varies with the purpose and complexity of the desired IaaS platform (deployment simplicity, performance or high availability). All Eucalyptus components need to be installed on 64-bit architecture physical machines (not virtual machines) with synchronized clocks. It is possible to install the Cloud Controller, Walrus, Cluster Controller and Storage Controller on one machine and a Node Controller on one or more machines. Alternatively, one can install each component on an independent physical server which gives each component maximal local resource usage. Placing all cloud and cluster components on a single machine simplifies the administration as there is only one machine to monitor and control. However, each component deploys as an independent Web service. If these components share a single physical server, the physical resources that can be given to each service may become a performance bottleneck. On the other hand, separating service components in different physical machines that share the same internal network can decrease messaging efficiency (*e.g.*, software services written in Java that share the same

Java environment) but increase the responsiveness of the overall Eucalyptus system due to the availability of physical resources (*e.g.*, large memory footprint). Figure 3.13 represents an Eucalyptus deployment structure where all standard software components run in different physical machines.

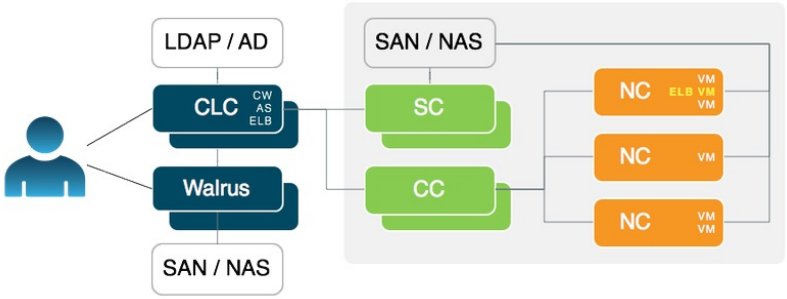


Figure 3.13: Eucalyptus generalized deployment structure [118].

The Cloud Controller must have Transmission Control Protocol (TCP)/IP connectivity to all other Eucalyptus components except for Node Controllers, which may reside on their own private networks. In addition, Walrus needs to be able to route network traffic directly to the Node Controllers for image delivery. The Cluster Controller physical machine can act as a software IP gateway between VM instances and the public network when elastic IP and security groups are implemented. This means that the physical server on which the Cluster Controller is deployed should have fast, dedicated network access to both the Node Controller network and the public network (where the Cloud Controller, Walrus and Storage Controller are located). The machine on which the Storage Controller is deployed must always have TCP/IP connectivity to the Cloud Controller and to the chosen SAN device (SAN integration drivers are only provided through subscription service). When SAN is not implemented the Storage Controller requires only TCP/IP connectivity to the Node Controllers in the cluster to provide network access to the dynamic block volumes residing on the Storage Controller's storage. The optional VMware Broker runs in the same machine as the Cluster Controller (when the infrastructure platform has more than one cluster it runs on the Cluster controller of the cluster that will use the VMware components (vCenter Server or ESX/ESXi). All Node Controllers must have network connectivity to either a SAN or the Storage Controller and Walrus.

3.4.4 Authentication and Authorization

Eucalyptus manages access control through an authentication, authorization, and accounting system. This system manages user identities, enforces access controls over resources and provides reporting on resource usage as a basis for auditing

and managing cloud activities. The user identity organizational model and the scheme of authorizations used to access resources are based on and compatible with the AWS IAM system, with some Eucalyptus extensions provided for a private cloud environment. Authentication can also be performed by integrating Eucalyptus with an existing LDAP or Active Directory. In this case, the user, group and account information and the Eucalyptus administrator console login authenticate using the LDAP/AD service. Eucalyptus-specific information about user, group and account is stored within the local database of Eucalyptus, including certificates, secret keys and attached policies. Each user has a unique set of credentials. These credentials are used to authenticate access to resources. There are three types of credentials:

- An X.509 certificate, used to authenticate requests to the SOAP API service;
- A secret access key, used to authenticate requests to the REST API service;
- A login password, used to authenticate the Eucalyptus administrator console access.

Eucalyptus also has two special identities for the convenience of administration and use of the system:

- The Eucalyptus account - each user has unrestricted access to all of the cloud's resources, similar to the super user on a typical Linux system. This account is automatically created when the system starts for the first time and cannot be removed from the system;
- The admin user of an account - each account, including the Eucalyptus account, has a user named admin. This user is created automatically by the system when an account is created. The admin of an account has full access to the resources owned by the account. It can not be removed from an account and it can delegate resource access to other users in the account by using policies.

3.5 Parallels Automation for Cloud Infrastructure

Parallels is a hosting and cloud services company that released in November 2011 the Parallels Automation for Cloud Infrastructure (PACI) [119]. PACI allows the deployment of cloud services on top of the Parallels Automation 5.3 software. While the Parallels Automation is a global software solution to implement cloud services, PACI is only a service module of Parallels Automation that is focused

on the provision of IaaS to SME. As opposed to the previous open source IaaS platform solutions, PACI (as well as all Parallels software services) is proprietary. PACI, is currently on its third release (as part of Parallels Automation 5.5), offers virtualization, automated operations and billing, customer self-service dashboards and an on line store to service providers.

3.5.1 Architecture

PACI has a distributed and modular architecture that is composed by four core services (Customer Self-service, Provision and Orchestration, Billing and Infrastructure Management layers) and two interfaces that enable integration with other systems and services as represented on Figure 3.14 in red and in orange respectively. Through the services interface are attached the business Storefront and Marketplace where a Web store or a provided business infrastructure exposes to customers the service provider offerings.

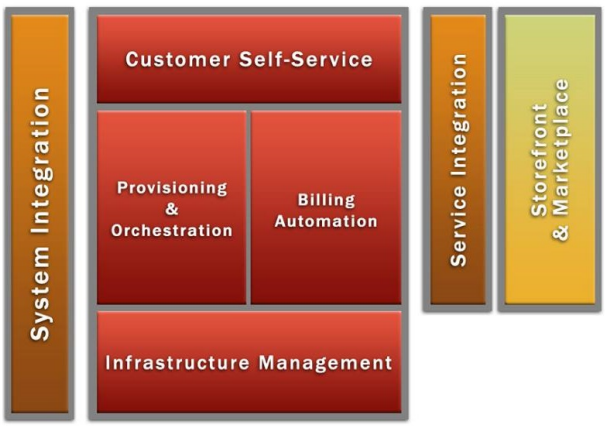


Figure 3.14: PACI architecture [120].

The Customer Self-Service layer implements the usual IaaS dashboards or control panels for the administrators, re-sellers and end-users. Through them it is possible to execute tasks over the provided services (add, modify or remove services), manage accounts and users, run reports and obtain support. Provisioning and Orchestration as well as Billing Automation stack on top of the Infrastructure Management layer and inter-operate in order to enable providers to manage and commercialize raw infrastructure resources. Provisioning and Orchestration services are responsible for the resources provision and management, service monitoring and management and to report to the Billing Automation Block. The Billing Automation service processes the payment and taxation, rating, billing and interact with the Storefront and Marketplace services managing the product catalogue and to deal with new orders, marketing and promotions.

Finally, the Infrastructure Management layer is responsible for the coordination of tasks related to the management of the physical and virtual infrastructure resources. It manages the networks, virtual environment, the storage and servers as well as implement security procedures, motorisation and the provision of the servers.

The presented services that are illustrated in Figure 3.14 are implemented through a combination of four main technologies: (*i*) PACI modules, *e.g.*, instances management and instances management DB; (*ii*) Parallels Operations Automation (POA), which is the Operations Support System (OSS) responsible for the centralized service management, service provisioning and the global infrastructure management; (*iii*) Parallels Business Automation (PBA), which is the Business Support System (BSS) that performs the resource usage accounting and billing; and (*iv*) Parallels Server Bare Metal (PSBM), which provides hybrid server virtualization, allowing hardware virtualization and OS virtualization to run side-by-side on the same physical server.

3.5.2 Interfaces

As a cloud interface PACI provides a RESTful API that enables the programmatic access to the PACI server through HyperText Transfer Protocol Secure (HTTPS) , as illustrated on Figure 3.15. This API allows getting Information about and performing actions on the resources through external cloud management programs. On the other hand, the OSS POA located in the Parallels Automation layer provides compatibility with an extensive list of third-party products [121] and external billing integration systems through a public API.

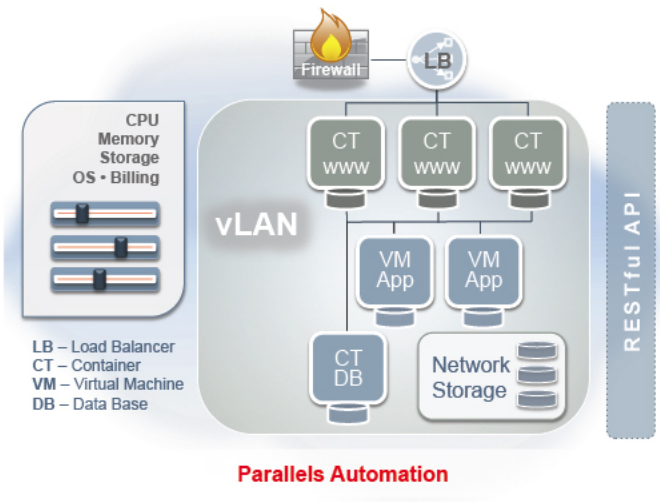


Figure 3.15: PACI RESTful interface [119].

3.5.3 Platform Deployment

PACI is deployed by combining the POA and PBA services with a cloud infrastructure module. POA services include a management node which runs the POA management software and system DB, a cluster of physical or virtual UI and branding servers that provide both high availability and load balancing functions regarding the access and interaction with the system's control panel, a private proxy responsible for transferring XML-RPC or HTTPS requests from the protected internal network to the front network and, finally, DNS servers which can also be implemented externally but without direct management from the POA management node. The PBA services are composed by an application server containing the PBA core software and UI, a DB server and a on line store server. On the other hand, the cloud infrastructure model consists of an Instance Manager (IM) node which acts as the control centre for all operations related to the Cloud Infrastructure module, an IM database node to store all information related to PACI, at least two PSBM nodes to deploy virtual machines or virtual containers, two storage nodes for each PSBM node to store backups and images, a Microsoft Provisioning System (MPS) node and an Active Directory domains controller to run the AD service. Figure 3.16 illustrates the deployment of a PACI IaaS platform.

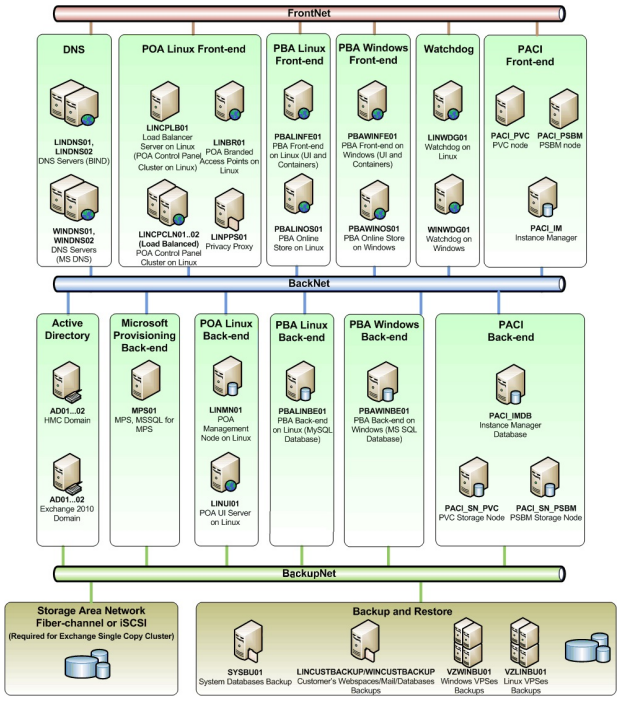


Figure 3.16: PACI deployment structure [122].

The deployed services nodes are organized as front end and back end services.

The front end nodes are exposed to the front and back networks in order to expose the infrastructure services to the Internet while being managed by the respective core service nodes from a secure network. These nodes are typically the ones that interact with the end users, *e.g.*, nodes containing the virtual machines or virtual containers, servers containing the control panels from POA and PBA, DNS servers and third-party services like Watchdog. The infrastructure can also implement system backups through a third network (Backup net), contributing for the high availability of the system and the provided services.

3.5.4 Authentication and Authorization

PACI utilizes an user authentication and authorization strategy based on groups, permissions, roles and authentication databases. Users are objects characterized by the roles delegated to them in a certain scope. Users can be members of groups. Users and groups can be retrieved either from local databases or from databases on external computers in the network. The information on those databases is stored on the physical server in the form of authentication databases. Roles are sets of abstract privileges that can be assigned to a user or a group to form a permission. Permissions enable users or groups to perform certain operations in different scopes, *e.g.*, virtual environments, physical servers, logical units.

3.6 Conclusion

This chapter summarises the main features of the five IaaS platforms studied. Table 3.1 presents a global comparison regarding authentication, hypervisors, management, interfaces, network, storage and governance.

Table 3.1: IaaS platforms comparison.

	IaaS Frameworks				
Features	**OpenNebula**	**OpenStack**	**CloudStack**	**Eucalyptus**	**PACI**
Authorization/ Authentication	Password, SSH RSA keypair, X509, LDAP	In-memory Key-Value Store, PAM, LDAP, X509	Password, LDAP, SSH RSA keypair	Password, access Key, X509, LDAP	Password, LDAP
Hypervisors	XEN, KVM, VMware vSphere	KVM, LXC, UML, VMWare vSphere, Xen, PowerVM, Hyper-V	VMware vSphere, KVM, Citrix Xen	XEN, KVM, VMware vSphere	Parallels hypervisor, KVM
Management	Centralized	Scattered	Centralized	Centralized	Centralized
Interfaces	Native: XML-RPC API; Additional: AWS EC2, OCCI, OCA	Native: RESTful API; Additional: AWS EC2, S3, EBS and OCCI	Native: Query API; Additional: AWS EC2, Plug-in API	AWS API	RESTful API
Network	Virtual router, Contextualization	Nova-network, Newtron	Virtual router	Cluster Controller	POA
Storage	Volume Storage	Volume and Object storage (Glance, Swift, Cinder)	Volume Storage	Volume and Object storage (Walrus, Storage Controller)	System DB
Governance Model	Benevolent Dictator	Foundation	Technical Meritocracy	Benevolent Dictator	Proprietary

The IaaS platforms studied contemplate both open source and proprietary software frameworks with different architectures and cloud deployment focuses. This diversity is caused by the absence of well defined architectural standards for the commoditization of IaaS systems (converting functionalities with economic value, distinguishable in terms of attributes,uniqueness or brand, to simple commodities in the eyes of the market or consumers). Every IaaS platform tends to provide distinct functionalities and be compatible with specific third-party services in order to monopolize the market and impose the its technologies as standards.

OpenNebula is the only non-US IaaS platform studied. This platform is ready to manage virtual resources from public and hybrid clouds. It presents a layered architecture, that enables a centralized management of the data-centre, and provides a well structured documentation. At the top of the stack, it exposes multiple API that enable the communication with AWS and OCCI-based solutions.

OpenStack is a perfect example of an IaaS platform that tries to monopolize the market. This platform is highly dynamic, presenting several new functionalities with each software release. However, it is fragmented into multiple software modules (OpenStack projects) with dedicated interface libraries. This is the consequence for being a foundation that serves essentially the needs of the vendors (the enterprises that form the OpenStack Foundation). This fragmentation hardens the installation process, the management of the platform and increases the complexity of the system. On the other hand, it interacts with several third-party applications, uses RESTful interfaces and offers OCCI and AWS interface libraries.

Apache CloudStack uses a modular architecture. This platform is focussed on the automation and management of various data centres in a centralized manner, being organized by zones, pods and clusters. It uses a Query API as well as an API translator so that applications written for CloudStack can also run in AWS. However, CloudStack does not provide OCCI support.

Eucalyptus is the only open source IaaS platform that is not licensed under Apache 2.0 (GPL v3.0). Its main purpose is to enable the deployment of private and hybrid open-source clouds that behave and are compatible with the Amazon Web Services. It presents a modular architecture exposing interfaces exclusively for AWS, which excludes the communication with other IaaS platforms and Open Cloud Computing Interfaces like OCCI. Due to the nature of this IaaS platform (deployment of private clouds compatible with AWS only) and the absence of dedicated user application programming interfaces, it will be discarded from future studies.

The Parallels Automation for Cloud Infrastructure (PACI) includes various proprietary (Parallels) products (*e.g.*, POA, PBA, PSBM) that are combined to enable the creation, management, monitoring and billing of a public or hybrid (if the same PACI platform is used) cloud IaaS platform. This platform exposes an open interface (RESTful API) to enable the development of third-party applications for the interaction with the system. However, PACI is essentially a "closed" platform without software modules that support directly the interaction with other IaaS platforms. This approach is common among proprietary solutions where the user lock-in phenomenon is purposely originated to monetize new products and paid support services.

The following chapter analyses in detail the corresponding client interface API of the OpenNebula, OpenStack, CloudStack and PACI, in order to identify both their common and distinctive interface features.

Chapter 4

Interface Libraries Comparison

This chapter compares the client interface libraries provided by OpenNebula, OpenStack, CloudStack and PACI IaaS platforms in terms of programmable components, available operations, including the parameters and respective attributes.

4.1 Interface Types

This section describes the RESTful, REST-like (Query) and RPC interface libraries used natively by the OpenStack, PACI, CloudStack and OpenNebula IaaS platforms respectively.

4.1.1 RESTful API

A RESTful API is a Web API conforming to the REST architectural style introduced in 2000 by Roy Fielding in his academic dissertation entitled "Architectural Styles and the Design of Network-based Software Architectures" [123]. These interface libraries expose a set of data and functions to facilitate interactions between computer programs and allow them to exchange information following four basic design principles:

- Expose a directory structure-like Uniform Resource Identifier (URI);
- Are stateless;
- Use HTTP access methods explicitly;
- Transfer resources through different media-types representations (usually JSON or XML).

RESTful API use URI to address resources (sources of specific information). The generic URI syntax is defined by the Request For Comments (RFC) 3986 [124] and it consists of a hierarchical sequence of components referred to as the scheme, authority, path, query and fragment:

```
URI = scheme "://" authority "/" path [ "?" query ] [ "#" fragment ]
```

The most common schema are HTTP and HTTPS. The authority corresponds to the IP address or host name and port of the server containing the Web API, the path represents the path to the RESTful API resource within the server. The query component starts after the question mark ("?") character and is terminated by a number sign ("#") character or by the end of the URI. It can be used to send additional (optional) parameters, *e.g.*, add a filter to retrieve a particular information. The fragment identifier component of a URI allows indirect identification of a secondary resource by reference to a primary resource and additional identifying information.

The stateless constraint dictates that a Web server is not required to memorize the state of the client application. As a result, each client must include the contextual information that it considers relevant in each interaction with the Web server. Web servers expect clients to manage the complexity of communicating their application state so that the Web server can service a larger number of clients [125].

REST API libraries embrace all aspects of the HyperText Transfer Protocol, version 1.1 (HTTP/1.1) [126], including request methods, response codes and message headers. Each HTTP method has specific, well-defined semantics within the context of a REST API library resource model:

- GET is used to retrieve data from a resource;
- HEAD is used to retrieve the metadata associated with a resource;
- PUT is used to add or update a resource;
- DELETE removes a resource from its parent;
- POST is used to submit data to a resource.

The result of a request is obtained by the Status-Line part of an HTTP response message. HTTP defines forty standard status codes divided into the following five categories:

- 1xx - Communicates transfer protocol-level information;
- 2xx - Indicates that the client's request was accepted successfully;

- 3xx - Indicates that the client must take some additional action in order to complete their request;
- 4xx - Client Error;
- 5xx - Server Error.

REST API libraries often employ a text-based format to represent a resource state as a set of meaningful fields. This resource representation typically reflects the current state of the resource and its attributes at the time of the client application request. The last set of constraints of a RESTful Web service design has to do with the exchanged data format (both in the request/response payload or in the HTTP body). The most commonly data exchange formats are XML and JSON.

4.1.2 XML-RPC API

The XML-RPC library uses the HTTP as a transport protocol to execute remote procedure calls encoded in XML in a remote machine [127]. An RPC is initiated by the client, which sends a request message (the calling process) to a known remote server (the called process) to execute a specified procedure with supplied parameters. A server API is made available at a particular Uniform Resource Locator (URL), *e.g.*, `http://server.org:8080/rpcapi/`.

To consume the server-side available procedures, the following steps are necessary:

- The client program makes a procedure call using the XML-RPC client, specifying a method name, parameters and a target server;
- The XML-RPC client takes the method name and parameters and then packages them as XML (this procedure is known by marshalling). Then the client issues an HTTP POST request, containing the request information to the target server;
- The target HTTP server receives the POST request and passes the XML content to an XML-RPC listener;
- The XML-RPC listener parses the XML (also known as un-marshalling) to get the method name and parameters and, then, invokes the appropriate method, passing the received parameters;
- The method returns a response to the XML-RPC process, which, in turn, packages the response as XML;
- The Web server returns this XML response to the client;

- The XML-RPC client parses the XML response, extracts and sends the return value to the client program.

4.1.3 Query API

Query interfaces use standard components of the HTTP protocol to invoke the library actions. However, unlike the RESTful interfaces, they do not use the HTTP message components to fully describe API operations. The HTTP envelope serves merely as a way of delivering parameters (simple name and value pairs), specifying both the operation and the data. With Query interfaces the operations are performed by rewriting the parameters in a GET request URI or in the body of a POST request.

Although Query interfaces differ from RESTful interfaces, they only use standard HTTP message components to perform operations. This type of interfaces are also known as REST-like interfaces.

4.2 OpenNebula Interface

OpenNebula uses a XML-RPC API to expose a set of operations, with a name and a set of input values, for end-users to interact with the OpenNebula system. The returned response uses always three pre-defined output values. The first and the third output values are fixed and correspond, respectively, to the status of the invoked operation and to an error code. The second value is used to indicate failure or to return information about the performed operation. The information returned by one..info operations is XML-formatted. The version 4.2 of OpenNebula interface libraries provides a large list of components that can be consulted at [128].

The access to the available operation provided by each component uses a session string generated by the user authentication. The study of the OpenNebula API will be focussed on the components that are common with the other IaaS platforms under analysis and covers: (*i*) Virtual Machine Management; (*ii*) Template Management; (*iii*) Image Management; (*iv*) Virtual Network Management; and (*v*) Data-store Management. These components and their respective operations will be described in the next sections.

4.2.1 Server Management

The Server (virtual machine) Management component is used to perform management operation to specific virtual machines in OpenNebula. The provided operations are: (*i*) Allocate VM; (*ii*) VM Actions; (*iii*) Save Disk; (*iv*) Attach Disk; (*v*) Detach Disk; (*vi*) Attach NIC; (*vii*) Detach NIC; (*viii*) Change VM Ownership; (*ix*) Rename VM; (*x*) Create Snapshot; (*xi*) Revert Snapshot; (*xii*)

Delete Snapshot; (*xiii*) Resize VM; (*xiv*) Update VM; (*xv*) VM Information; (*xvi*) VM Pool Information, (*xvii*) VM Monitoring; and (*xviii*) VM Pool Monitoring. There are other operations that are specific to cloud administrators that will not be considered in this description like: (*i*) VM Deploy; (*ii*) VM Reschedule and Unreschedule actions; (*iii*) VM Permissions Mode (chmod); (*iv*) VM Migrate; (*v*) VM Recover and (*vi*) VM Pool Accounting.

4.2.1.1 Allocate VM

The Allocate VM operation allocates a new virtual machine in OpenNebula. This operation invokes the one.vm.allocate method with the TEMPLATE and Boolean data type parameters. If the operation is successful the allocated resource ID parameter is returned.

- The TEMPLATE parameter is a string containing the template attributes to be applied to the allocated VM. These attributes are organized by Capacity, OS and Boot, Disk, Network, I/O Devices, Context, Placement and Raw sections. The structure and parameters of each section are described in the Virtual Machine Definition File 4.2 [129];
- The Boolean data type parameter sets the VM creation on pending (if false) or on hold (if true).

4.2.1.2 VM Actions

The VM Actions operation submits an action to be performed on a virtual machine. This operation invokes the one.vm.action method with the action and ID parameters. If the operation is successful, it returns the VM ID parameter.

- The action parameter is a string that indicates the name of the action to be performed on the VM. The available action names are: (*i*) SHUTDOWN; (*ii*) SHUTDOWN-HARD; (*iii*) HOLD; (*iv*) RELEASE; (*v*) STOP; (*vi*) SUSPEND; (*vii*) RESUME; (*viii*) BOOT; (*ix*) DELETE; (*x*) DELETE-RECREATE; (*xi*) REBOOT; (*xii*) REBOOT-HARD; (*xiii*) POWEROFF; (*xiv*) POWEROFF-HARD; (*xv*) UNDEPLOY; and (*xvi*) UNDEPLOY-HARD.

4.2.1.3 Save Disk

The Save Disk operation sets the disk to be saved in the given image. This operation invokes the one.vm.savedisk method with the ID, DISK ID, NAME, TYPE and Boolean data type parameters. If the operation is successful, it returns the new Image ID parameter.

- The Name parameter indicates the name for the new image where the disk will be saved;
- The Type parameter indicates the image type;
- The Boolean data type parameter sets if the disk is saved immediately (true) or if the operation will be performed when the VM shuts down (false).

4.2.1.4 Attach Disk

The Attach Disk operation attaches a new disk to a specific virtual machine. This operation invokes the one.vm.attach method with the ID and DISK parameters. If the operation is successful, it returns the VM ID parameter.

- The DISK parameter is a string containing pre-defined attributes from the Disk section template [129].

4.2.1.5 Detach Disk

The Detach Disk operation detaches a disk from a specific virtual machine. This operation invokes the one.vm.detach method with the ID and DISK ID. If the operation is successful, it returns the VM ID parameter.

4.2.1.6 Attach NIC

The Attach NIC operation attaches a new network interface to the virtual machine. This operation invokes the one.vm.attachnic method with the ID and NIC parameters. If the operation is successful, it returns the VM ID parameter

- The NIC parameter is a string containing pre-defined attributes from the NIC section template [129].

4.2.1.7 Detach NIC

The Detach NIC operation detaches a network interface from a virtual machine. This operation invokes the one.vm.detachnic method with the ID and NIC ID parameters. If the operation is successful, it returns the VM ID parameter.

4.2.1.8 Change VM Ownership

The Change VM Ownership operation changes the ownership of a virtual machine. This operation invokes the one.vm.chown method with the ID, USER ID and GROUP ID parameters. If the operation is successful, it returns the resource ID parameter.

4.2.1.9 Rename VM

The Rename VM operation renames a virtual machine. This operation invokes the one.vm.rename method with the ID and NAME parameters. If the operation is successful, it returns the VM ID parameter.

- The NAME parameter is a string with the new name for the VM.

4.2.1.10 Create Snapshot

The Create Snapshot operation creates a new virtual machine snapshot. This operation invokes the one.vm.snapshotcreate method with the ID and NAME parameters. If the operation is successful, it returns the new SNAPSHOT ID parameter.

- The NAME parameter is a string containing the new snapshot name (it may be empty).

4.2.1.11 Revert Snapshot

The Revert Snapshot operation reverts a virtual machine to a specified snapshot. This operation invokes the one.vm.snapshotrevert method with the ID and SNAPSHOT ID parameters. If the operation is successful, it returns the VM ID parameter.

4.2.1.12 Delete Snapshot

The Delete Snapshot operation deletes a virtual machine snapshot. This operation invokes the one.vm.snapshotdelete method with the ID and SNAPSHOT ID parameters. If the operation is successful, it returns the VM ID parameter.

4.2.1.13 Resize VM

The Resize VM operation changes the capacity of the virtual machine. This operation invokes the one.vm.resize method with the ID and TEMPLATE parameters. If the operation is successful, it returns the VM ID parameter.

- The TEMPLATE parameter with the new capacity elements (CPU, VCPU and MEMORY).

4.2.1.14 Update VM

The Update VM operation replaces the user template contents. This operation invokes the one.vm.update method with the ID and TEMPLATE and parameters. If the operation is successful, it returns the resource ID parameter.

- The TEMPLATE parameter contains the new user template contents for the VM.

4.2.1.15 VM Information

The VM Information operation retrieves information about the virtual machine. This operation invokes the one.vm.info method with the ID parameter. If the operation is successful, it returns the information parameter.

- The information parameter is a string XML-formatted containing the information of a specific VM.

4.2.1.16 VM Pool Information

The VM Pool Information operation retrieves information for all or part of the virtual machines associated with an user. This operation invokes the one.vmpool.info method with the filter, range start ID, range end ID and state parameters. If the operation is successful, it returns the information parameter.

- The filter parameter is a filter flag that can be set to: -3 (or lower values) to show connected user's resources, -2 to show all resources, -1 to show connected user's and his group's resources and 0 (or higher values) to show Unique IDentifier (UID) User's Resources;
- The range start ID and range end ID parameters filter the result by a VM ID range;
- The state parameter filters the result by the VM state;
- The information parameter is a string XML-formatted containing information about all VM.

4.2.1.17 VM Monitoring

The VM Monitoring operation returns the virtual machine monitoring records. This operation invokes the one.vm.monitoring with the ID parameters. If the operation is successful, it returns the monitoring parameter.

- The monitoring parameter is a string with a list of XML element records of a specific VM.

4.2.1.18 VM Pool Monitoring

The VM Pool Monitoring operation returns all the virtual machine monitoring records. This operation invokes the one.vmpool.monitoring with the filter parameter. If the operation is successful, it returns the monitoring parameter.

- The monitoring parameter returns a list of VM elements. Each VM element contains the complete xml of the VM with the updated information returned by the poll action.

The operations and parameters used by the Server Management component from OpenNebula interface are resumed in Table 4.1.

Table 4.1: Server Management Table (operations *vs* parameters).

	Operations																	
Parameters	**Allocate VM**	**Actions**	**Save Disk**	**Attach Disk**	**Detach Disk**	**Attach NIC**	**Detach NIC**	**Change VM Ownership**	**Rename VM**	**Create Snapshot**	**Revert Snapshot**	**Delete Snapshot**	**Resize VM**	**Update VM**	**VM Information**	**VM Pool Information**	**VM Monitoring**	**VM Pool Monitoring**
Sent Parameters																		
TEMPLATE	x		x										x	x				
NAME									x	x								
Boolean data type	x		x															
ID		x	x	x	x	x	x	x	x	x	x	x	x	x	x		x	
action		x																
DISK ID			x		x													
TYPE			x															
DISK				x														
NIC						x												
NIC ID							x											
USER ID								x										
GROUP ID								x										
SNAPSHOT ID											x	x						
Received Parameters																		
SNAPSHOT ID										x								
ID	x	x		x	x	x	x	x	x		x	x	x	x				
information															x	x		
monitoring																	x	x
IMAGE ID			x															

4.2.2 Template Management

The Template Management component provides methods to manage the template repository in OpenNebula using the following list of operations: (*i*) Allocate Template; (*ii*) Clone Template; (*iii*) Delete Template; (*iv*) Instantiate Template; (*v*) Update Template; (*vi*) Change Template Ownership; (*vii*) Rename Template; (*viii*) Template Information; and (*ix*) Template Pool Information.

4.2.2.1 Allocate Template

The Allocate Template operation allocates a new template in OpenNebula. This operation invokes the one.template.allocate method with the TEMPLATE parameters. If the operation is successful, it returns the allocated resource ID parameter.

4.2.2.2 Clone Template

The Clone Template operation clones an existing virtual machine template. This operation invokes the one.template.clone method with the ID and NAME parameters. If the operation is successful, it returns the new template ID parameter.

- The ID parameter indicates the identifier of the template;
- The NAME parameter defines the new cloned template name.

4.2.2.3 Delete Template

The Delete Template operation deletes the given template from the repository. This operation invokes the one.template.delete method with the ID parameter. If the operation is successful, it returns the resource ID parameter.

4.2.2.4 Instantiate Template

The Instantiate Template operation instantiates a new virtual machine from a template. This operation invokes the one.template.instantiate method with the ID, VM NAME, EXTRA TEMPLATE and Boolean data type parameters. If the operation is successful, it returns the new VM ID parameter.

- The VM NAME parameter indicates the name for the new VM instance;
- The EXTRA TEMPLATE parameter contains an extra template to be merged with the one being instantiated;
- The Boolean data type parameter sets the VM creation on pending (if false) or on hold (if true).

4.2.2.5 Update Template

The Update Template operation replaces the template contents. This operation invokes the one.template.update method with the ID, TEMPLATE and TYPE parameters. If the operation is successful, it returns the resource ID parameter.

- The TYPE parameter specifies the update type, indicating if the template is going to be partial or completely modified.

4.2.2.6 Change Template Ownership

The Change Template Ownership operation changes the ownership of a template. This operation invokes the one.template.chown method with the ID, USER ID and GROUP ID parameters. If the operation is successful, it returns the resource ID parameter.

- The USER ID parameter identifies the new template owner;
- The GROUP ID parameter identifies the new group for the template.

4.2.2.7 Rename Template

The Rename Template operation renames a template. This operation invokes the one.template.rename method with the ID and NAME parameters. If the operation is successful, it returns the VM ID parameter.

4.2.2.8 Template Information

The Template Information operation retrieves information from the template. This operation invokes the one.template.info method with the ID parameter. If the operation is successful, it returns the information parameter.

- The information parameter is a string XML-formatted containing the information of a specific VM template.

4.2.2.9 Template Pool Information

The Template Pool Information operation retrieves information about all or part of the public and owned templates in the repository. This operation invokes the one.templatepool.info method with the filter, range-start-ID and range-end-ID parameters. If the operation is successful, it returns the information parameter.

- The information parameter is a string XML-formatted containing the information of a list of VM templates.

The operations and parameters used by the Template Management component from OpenNebula interface are resumed in Table 4.2.

Table 4.2: Template Management Table (Parameters *vs* Operations).

	Operations								
Parameters	**Allocate Template**	**Clone Template**	**Delete Template**	**Instantiate Template**	**Update Template**	**Change Template Ownership**	**Rename Template**	**Template Information**	**Template Pool Information**
Sent Parameters									
TEMPLATE	x				x				
ID		x							
EXTRA TEMPLATE		x		x					
NAME							x		
VM NAME				x					
Boolean data type				x					
USER ID						x			
GROUP ID						x			
filter									x
range start ID									x
range end ID									x
TYPE					x				
Received Parameters									
ID	x		x	x	x	x	x	x	
information								x	x
VM ID				x			x		

4.2.3 Image Management

The Image Management component provides methods to manage the image repository in OpenNebula using the following list of operations: (*i*) Allocate Image; (*ii*) Clone Image; (*iii*) Delete Image; (*iv*) Enable Image; (*v*) Persistent; (*vi*) Change Image Type; (*vii*) Update Image; (*viii*) Change Image Ownership; (*ix*) Rename Image; (*x*) Image Information; and (*xi*) Image Pool Information.

4.2.3.1 Allocate Image

The Allocate Image operation allocates a new image in OpenNebula. This operation invokes the one.image.allocate method with the TEMPLATE and DATASTORE ID parameters. If the operation is successful, it returns the allocated resource ID parameter.

- The template parameter is a string XML-formated containing the template attributes of the image [130].

4.2.3.2 Clone Image

The Clone Image operation clones an existing image. This operation invokes the one.image.clone method with the ID and NAME parameters. If the operation is successful, it returns the new image ID parameter.

4.2.3.3 Delete Image

The Delete Image operation deletes the given image from the repository. This operation invokes the one.image.delete method with the ID parameter. If the operation is successful, it returns the resource ID parameter.

4.2.3.4 Enable Image

The Enable/Disable Image operation enables or disables an image. This operation invokes the one.image.enable method with the ID and Boolean data type parameters. If the operation is successful, it returns the image ID parameter.

- The Boolean data type parameter enables (if true) or disables (if false) the image.

4.2.3.5 Persistent

The Persistent operation sets the image as persistent or non-persistent. This operation invokes the one.image.persistent method with the ID and Boolean data type parameters. If the operation is successful, it returns the image ID parameter.

- The Boolean data type parameter defines the image as persistent (if true) or as non-persistent (if false).

4.2.3.6 Change Image Type

The Change Image Type operation changes the type of an image. This operation invokes the one.image.chtype method with the ID and image-type parameters. If the operation is successful, it returns the image ID parameter.

- The image-type parameter identifies the image type. The available types are: OS, CDROM, DATABLOCK, KERNEL, RAMDISK and CONTEXT. The default type OS is attributed in case this parameter is omitted.

4.2.3.7 Update Image

The Update Image operation replaces the image template contents. This operation invokes the one.image.update method with the ID, TEMPLATE and TYPE parameters. If the operation is successful, it returns the resource ID parameter.

4.2.3.8 Change Image Ownership

The Change Image Ownership operation changes the ownership of an image. This operation invokes the one.image.chown method with the ID, USER ID and GROUP ID parameters. If the operation is successful, it returns the resource ID parameter.

4.2.3.9 Rename Image

The Rename Image operation renames an image. This operation invokes the oe.image.rename method with the ID and NAME parameters. If the operation is successful, it returns the VM ID parameter.

4.2.3.10 Image Information

The Image Information operation retrieves information about a specific image. This operation invokes the one.image.info method with the ID parameter. If the operation is successful, it returns the information parameter.

- The information parameter is a string XML-formatted containing the information of a specific image.

4.2.3.11 Image Pool Information

The Image Pool Information retrieves information for all or part of the images in the repository. This operation invokes the one.imagepool.info with the filter, range start ID and range end ID parameters. If the operation is successful, it returns the information parameter.

- The information parameter is a string XML-formatted containing the information of a list of images.

The operations and parameters used by the Image Management component from OpenNebula interface are resumed in Table 4.3.

Table 4.3: Image Management Table (Parameters *vs* Operations).

Parameters	*Operations*										
	Allocate Image	**Clone Image**	**Delete Image**	**Enable/Disable Image**	**Persistent**	**Change Image Type**	**Update Image**	**Change Image Ownership**	**Rename Image**	**Image Information**	**Image Pool Information**
Sent Parameters											
TEMPLATE	x						x				
DATASTORE ID	x										
ID		x	x	x	x	x	x	x	x	x	
NAME		x							x		
image-type						x					
Boolean data type				x	x						
USER ID								x			
GROUP ID								x			
filter											x
range start ID											x
range end ID											x
TYPE							x				
Received Parameters											
ID	x	x	x	x	x	x	x	x			
VM ID									x		
information										x	x

4.2.4 Network Management

The VN Management component provides methods to manage virtual networks in OpenNebula using the following list of operations: (*i*) Allocate VN; (*ii*) Delete VN; (*iii*) Add VN Leases; (*iv*) Remove VN Leases; (*v*) Hold VN Lease; (*vi*) Release VN Lease; (*vii*) Update VN; (*viii*) Change VN Ownership; (*ix*) Rename VN; (*x*) VN Information; and (*xi*) VN Pool Information.

4.2.4.1 Allocate VN

The Allocate VN operation allocates a new virtual network in OpenNebula. This operation invokes the one.vn.allocate method with the TEMPLATE and CLUSTER ID parameters. If the operation is successful, it returns the allocated resource ID parameter.

- The TEMPLATE parameter is a string that specifies the template attributes of the virtual network [131];
- The CLUSTER ID parameter indicates the ID of the cluster. If the value is defined as -1 the virtual network will not be added to any cluster.

4.2.4.2 Delete VN

The Delete VN operation deletes the given virtual network from the pool. This operation invokes the one.vn.delete method with the ID parameter. If the operation is successful, it returns the resource ID parameter.

4.2.4.3 Add VN Leases

The Add VN Leases operation adds a new lease to the virtual network (only available for FIXED networks). This operation invokes the one.vn.addleases method with the ID and LEASE parameters. If the operation is successful, it returns the resource ID parameter.

- The LEASE parameter specifies the template of the lease to add.

4.2.4.4 Remove VN Leases

The Remove VN Leases operation removes a lease from the virtual network (also is only available for FIXED networks). This operation invokes the one.vn.rmleases method with the ID and LEASE parameters. If the operation is successful, it returns the resource ID parameter.

- The LEASE parameter specifies the template of the lease to removed.

4.2.4.5 Hold VN Lease

The Hold VN Lease operation holds a virtual network lease on use. This operation invokes the one.vn.hold method with the ID and LEASE parameters. If the operation is successful, it returns the resource ID parameter.

- The LEASE parameter specifies the template of the lease to hold.

4.2.4.6 Release VN Lease

The Release VN Lease operation releases a virtual network lease on hold. This operation invokes the one.vn.release method with the ID and LEASE parameters. If the operation is successful, it returns the resource ID parameter.

- The lease_template parameter specifies the template of the lease to release.

4.2.4.7 Update VN

The Update VN operation replaces the virtual network template contents. This operation invokes the one.vn.update method with the ID, TEMPLATE and TYPE parameters. If the operation is successful, it returns the resource ID parameter.

- The TEMPLATE parameter specifies the contents of the new template;
- The TYPE parameter specifies if the update operation is to replace the whole template (value='0') or to merge the new template with the existing one (value='1').

4.2.4.8 Change VN Ownership

The Change VN Ownership operation changes the ownership of a virtual network. This operation invokes the one.vn.chown method with the ID, USER ID and GROUP ID parameters. If the operation is successful, it returns the resource ID parameter.

4.2.4.9 Rename VN

The Rename VN operation renames a virtual network. This operation invokes the one.vn.rename method with the ID and NAME parameters. If the operation is successful, it returns the VM ID parameter.

4.2.4.10 VN Information

The VN Information operation retrieves information about the virtual network. This operation invokes the one.vn.info method with the ID parameter. If the operation is successful, it returns the information parameter.

- The information parameter is a string XML-formatted containing the information of a specific virtual network.

4.2.4.11 VN Pool Information

The VN Pool Information operation retrieves information about all or part of the virtual networks in the pool. This operation invokes the one.vnpool.info method with the filter, range start ID and range end ID parameters. If the operation is successful, it returns the information parameter.

- The information parameter is a string XML-formatted containing the information of a list of virtual networks.

The operations and parameters used by the Virtual Network Management component from OpenNebula interface are resumed in Table 4.4.

Table 4.4: Virtual Network Management Table (Parameters *vs* Operations).

	Operations										
Parameters	**Allocate**	**Delete**	**Add Lease**	**Remove Lease**	**Hold**	**Release**	**Update**	**Change Ownership**	**Rename**	**Information**	**VN Pool Information**
Sent Parameters											
TEMPLATE	x						x				
LEASE			x	x	x	x					
NAME									x		
TYPE							x				
USER ID								x			
ID		x	x	x	x	x	x	x	x	x	
GROUP ID								x			
CLUSTER ID	x										
filter											x
range start ID											x
range end ID											x
Received Parameters											
ID	x	x	x	x	x	x	x	x			
information										x	x
VM ID									x		

4.2.5 Data-Store Management

The Data-Store (DS) Management Component provides the following list of operations to manage the data-store types used with OpenNebula: (*i*) Allocate Data-store; (*ii*) Delete Data-store; (*iii*) Update Data-store; (*iv*) Change Data-store Ownership; (*v*) Data-store Information; and (*vi*) Data-store Pool Information.

4.2.5.1 Allocate Data-store

The Allocate Data-store operation allocates a new data-store in OpenNebula. This operation invokes the one.datastore.allocate method with the TEMPLATE parameter. If the operation is successful, it returns the resource ID parameter.

- The TEMPLATE parameter is a string that specifies the template of the data-store. The template attributes vary with the type of the desired data-store: LVM [132], vmware [133], ceph [134], system [135], file system [136], iSCSI [137] and kernel&file [138].

4.2.5.2 Delete Data-store

The Delete Data-store operation deletes the given data-store from the pool. This operation invokes the one.datastore.delete parameter with the ID parameter. If the operation is successful, it returns the resource ID parameter.

4.2.5.3 Update Data-store

The Update Data-store operation replaces the data-store template contents. This operation invokes the one.datastore.update method with the ID, TEMPLATE and TYPE parameters. If the operation is successful, it returns the resource ID parameter.

- The TEMPLATE parameter specifies a new data-store template;
- The TYPE parameter specifies if the update operation is to replace the whole template (value='0') or to merge the new template with the existing one (value='1').

4.2.5.4 Change Data-store Ownership

The Change Data-store Ownership operation changes the ownership of a data-store. This operation invokes the one.datastore.chown method with the ID, USER ID and GROUP ID parameters. If the operation is successful, it returns the resource ID parameter.

4.2.5.5 Data-store Information

The Data-store Information operation retrieves information from an OpenNebula data-store. This operation invokes the one.datastore.info method with the ID parameter. If the operation is successful, it returns the information parameter.

- The information parameter is a string XML-formatted containing the information of a specific data-store.

4.2.5.6 Data-store Pool Information

The Data-store Pool Information operation retrieves information from all or part of the data-stores in the pool. This operation invokes the one.datastorepool.info method without additional parameters. If the operation is successful, it returns the information parameter.

- The information parameter is a string XML-formatted containing the information of a list of data-stores.

The operations and parameters used by the Data-store Management component from OpenNebula interface are resumed in Table 4.5.

Table 4.5: Data-store Management Table (Parameters *vs* Operations).

	Operations					
Parameters	**Allocate DS**	**Delete DS**	**Update DS**	**Change DS Ownership**	**DS Information**	**DS Pool Information**
Sent Parameters						
TEMPLATE	x		x			
TYPE			x			
USER ID				x		
ID		x	x	x	x	
GROUP ID				x		
Received Parameters						
ID	x	x	x	x		
information					x	x

4.3 OpenStack Interface

The OpenStack API is a RESTful HTTP service that supports both JSON and XML data serialization request and response formats. The request format is specified in the Content-Type header of the request. The response format is specified either through the Accept header or by adding an .xml or .json extension to the request URI. JSON will be used by default. The API uses both URI and Multipurpose Internet Mail Extensions (MIME) type versioning schemas. In the URI schema, the first element of the path contains the target version identifier, *e.g.*:

```
https://servers.api.openstack.org/v1.0/224532/servers/123.
```

The MIME type versioning schema uses HTTP content negotiation where the Accept or Content-Type headers contains a MIME type that identifies the version (application/vnd.openstack.compute.v2+xml). A version MIME type is always linked to a base MIME type (application/xml or application/json) and allows the creation of permanent links because the version schema is not specified in the URI path, *e.g.*:

```
https://api.servers.openstack.org/224532/servers/123.
```

If conflicting versions are specified using both an HTTP header and a URI, the URI takes precedence.

The OpenStack interface libraries of the current release - Grizzly - are provided by the Nova, Cinder, Glance, Neutron, Keystone and Horizon modules. The complete list of components provided by these modules is available at [139]. This analysis will be focussed on the common functionalities between the OpenStack Nova, Glance, Neutron and Cinder interface libraries and the API of the other

IaaS platforms under comparison. The OpenStack Nova API manages the following components: (*i*) Servers; (*ii*) Server Addresses; (*iii*) Server Actions; and (*iv*) Flavours. The OpenStack Glance, Neutron and Cinder interface libraries, on the other hand, are responsible for the provision of Images, Networking and Volume management components respectively. To simplify the comparison with other interface libraries, the referred components will be listed and describe in the following sections as: (*i*) Server Management (aggregating the Servers, Server Addresses and Server Actions); (*ii*) Flavour Management; (*iii*) Image Management; (*iv*) Network Management and (*v*) Volume Management.

4.3.1 Server Management

The Servers Management component is used to perform actions, obtain information and list addresses for a specific server through the following list of operations: (*i*) List Servers; (*ii*) Create Server; (*iii*) Get Server Details; (*iv*) Update Server; (*v*) Delete Server; (*vi*) List Addresses; (*vii*) List Addresses by Network; (*viii*) Change Administrator Password; (*ix*) Reboot Server; (*x*) Rebuild Server; (*xi*) Resize Server; (*xii*) Confirm Resized Server; (*xiii*) Revert Resized Server; and (*xiv*) Create Image.

4.3.1.1 List Servers

The List Servers operation lists the servers associated with an account. This operation does not require a request body and the list can be filtered by the imageRef, flavorRef, name, status, marker, limit and change-Since query parameters. The response returns a servers parameter.

- The imageRef parameter specifies the image reference as an ID or full URL;
- The flavorRef parameter specifies the flavour reference as an ID or full URL;
- The name parameter specifies the name of the server;
- The status parameter specifies the server status values. These values can be: ACTIVE, BUILD, DELETED, ERROR, HARD_REBOOT, PASSWORD, REBOOT, REBUILD, RESCUE, RESIZE, REVERT_RESIZE, SHUTOFF, SUSPENDED, UNKNOWN and VERIFY_RESIZE;
- The marker parameter specifies the ID of the last item in the previous list;
- The limit parameter specifies the page size limit;
- The change-Since parameter specifies the time since the last change occurred;

- The servers parameter contains the id, tenant-id, user-id, name, status, progress, hostId, updated, created, accessIPv4, accessIPv6, imageRef, flavorRef, metadata, addresses and link attributes for each listed server.

4.3.1.2 Create Server

The Create Server operation asynchronously provisions a new server. This operation sends the required imageRef, flavorRef and name parameters as well as the optional metadata, personality and networks parameters. The response message returns the Location, id, link and adminPass parameters.

- The metadata parameter specifies the Metadata key and value pairs;
- The personality parameter specifies the file path and contents (text only) attributes to inject into the server at launch;
- The networks parameter specifies the list of networks and their attributes;
- The Location parameter contains a full URL to the created server.

4.3.1.3 Get Server Details

The Get Server Details operation returns the details of a specific server by its ID. This operation does not require a request body as the id parameter is included in the URI path. The response returns a server parameter.

- The server parameter contains the id, tenant-id, user-id, name, status, progress, hostId, updated, created, accessIPv4, accessIPv6, imageRef, flavorRef, metadata, addresses and link attributes for the specified server.

4.3.1.4 Update Server

The Update Server operation is used to update the editable attributes of a specified server. This operation sends the name, accessIPv4 and accessIPv6 parameters as well as the id parameter in the URI path. The response returns the server parameter.

4.3.1.5 Delete Server

The Delete Server operation deletes a specified cloud server instance from the system. This operation does not require a request body as the id parameter is included in the URI path. It also does not returns a response body. The success or failure of this operation is returned in the response header in the Status-Line.

4.3.1.6 List Addresses

The List Addresses operation lists networks and addresses for a specified tenant and server. This operation does not require a request body as the id parameter is included in the URI path. The response returns the addresses parameter.

- The addresses parameter include the list of public and private elements that provide information about the IP addresses through the version and addr attributes.

4.3.1.7 List Addresses by Network

The List Addresses by Network operation lists addresses for a specified tenant, server and network. This operation does not require a request body sending the id and network_label parameters in the URI path. The response returns the network parameter.

- The network_label parameter specifies the network label, such as public or private;
- The network parameter contains a list of IP elements with the version and addr attributes for the IP addresses defined in the network and the ID element that identifies the network type (public or private).

4.3.1.8 Change Administrator Password

The Change Administrator Password operation changes the administrator password for a specified server. This operation sends a adminPass parameter in the request body and the id parameter in the URI path. It does not return a response body. The success or failure of this operation is returned in the response header in the Status-Line.

4.3.1.9 Reboot Server

The Reboot Server operation enables a soft or hard reboot of the specified server. With a soft reboot, the operating system is signalled to restart. A hard reboot is the equivalent of power cycling the server. This operation sends the type parameter in the request body and the id parameter in the URI path. It does not return a response body. The success or failure of this operation is returned in the response header in the Status-Line.

- The type parameter specifies the type of reboot. The available values are SOFT and HARD.

4.3.1.10 Rebuild Server

The Rebuild Server operation removes all data on the server and replaces it with the specified image. This operation sends the name, imageRef, accessIPv4, accessIPv6, adminPass, metadata and personality parameters in the request body and the id parameter in the URI path. The Response returns the server container with the parameters described in the List Servers operation.

4.3.1.11 Resize Server

The Resize Server operation converts an existing server to a different flavour, scaling the server up or down. This operation sends the flavorRef parameter in the request body and the id parameter in the URI path. It does not return a response body. The success or failure of this operation is returned in Status-Line response header.

4.3.1.12 Confirm Resized Server

The Confirm Resized Server operation confirms the success of the operation, *i.e.*, after it has been verified that the newly resized server is functioning properly. This operation send the confirmResize parameter in the request body and the id parameter in the URI path. It does not return a response body. The success or failure of this operation is returned in the Status-Line response header.

4.3.1.13 Revert Resized Server

The Revert Resized Server operation is used to revert the resize and roll back to the original server. This operation sends the revertResize parameter in the request body and the id parameter in the URI path. It does not return a response body. The success or failure of this operation is found in the Status-Line response header.

4.3.1.14 Create Image

The Create Image operation creates a new image for the given server that can later be used to rebuild or create servers. This operation sends the name and metadata parameters in the request body and the id parameter in the URI path. It does not return a response body. The success or failure of this operation is returned in the Status-Line response header.

The operations and parameters used by the Server Management component from OpenStack interface are resumed in Table 4.6.

Table 4.6: Server Management Table (Parameters *vs* Operations).

	Operations													
Parameters	**List Servers**	**Create Server**	**Get Server Details**	**Update Server**	**Delete Server**	**List Addresses**	**List addresses by Network**	**Change Administrator Password**	**Reboot Server**	**Rebuild Server**	**Resize Server**	**Confirm Resize Server**	**Revert Resized Server**	**Create Image**
Sent Parameters														
imageRef	x	x								x				
flavorRef	x	x									x			
name	x	x		x						x				x
status	x													
marker	x													
limit	x													
change-since	x													
id			x	x	x	x	x	x	x	x	x	x	x	x
accessIPv4				x						x				
accessIPv6				x						x				
metadata		x								x				x
personality		x								x				
networks		x												
adminPass								x		x				
network_label							x							
confirmResize												x		
revertResize													x	
type									x					
Received Parameters														
location		x												
adminPass		x												
network							x							
link		x												
addresses						x								
servers	x													
server			x	x						x				
id		x												

4.3.2 Flavour Management

In OpenStack the virtual hardware templates are called "flavors" which are available hardware configurations for the servers. Each flavour has a unique combination of virtual CPU cores, disk space and memory capacity. The Flavour Management component is used to obtain the list of available flavours and to provide information about a specified flavour through the following operations: (*i*) List Flavours and (*ii*) Get Flavour Details.

4.3.2.1 List Flavours

The List Flavours operation lists information for all available flavours in the response body. This operation does not require a request body and the list can be filtered by the minDisk, minRam, marker and limit query parameters. The response returns the flavors parameter.

- The minDisk parameter specifies the minimum number of gigabytes of disk storage;
- The minRam parameter specifies the minimum amount of RAM in megabytes;
- The marker parameter identifies the Universally Unique IDentifier (UUID) of the flavor to set a marker;
- The limit parameter specifies the integer limit value of flavours returned;
- The flavors parameter contains the id, links and name attributes of each flavour.

4.3.2.2 Get Flavour Details

The Get Flavour Details operation returns details of the specified flavour in the response body. This operation does not require a request body as the id parameter is included in the URI path. The response returns the flavor parameter.

- The flavor parameter contains the the disk, id, links, name, ram and vcpus attributes of a specified flavour.

The operations and parameters used by the Flavour Management component from OpenStack interface are resumed in Table 8.

Table 4.7: Flavour Management Table (Parameters *vs* Operations).

	Operations	
Parameters	**List Flavours**	**Get Flavour Details**
Sent Parameters		
minDisk	x	
minRam	x	
marker	x	
limit	x	
id		x
Received Parameters		
flavors	x	
flavor		x

4.3.3 Image Management

The Image Management component is used to manage OpenStack images used to create or rebuild a server. This component allows the following operations: (*i*) Create Image; (*ii*) List Images; (*iii*) Get Image Details; (*iv*) Update Image; (*v*) Delete Image; (*vi*) Upload Image; (*vii*) Download Image; (*viii*) Add Image Tag; and (*ix*) Delete Image Tag.

4.3.3.1 Create Image

The Create Image operation creates a virtual machine image. This operation sends the required name parameter as well as the optional id, visibility and tags parameters. The response returns the Location parameter.

- The id parameter specifies the image ID;
- The name parameter specifies the image name file;
- The visibility parameter indicates if the image is public or private; by default it is public;
- The tags parameter attributes a tag to the image;
- The Location parameter contains a full URL to the created image.

4.3.3.2 List Images

The List Images operation lists a subset of a larger collection of images. This operation does not require a request body and the list can be filtered by the visibility, member_status, owner, size_min, size_max, sort_key, sort_dir, name, status, marker, and limit query parameters. The response returns the images parameter.

- The images parameter is composed by the id, name, deleted, container_format, created_at disk_format, updated_at, min_disk, protected, min_ram, checksum, owner, is_public, deleted_at and size attributes as well as the properties element with the kernel_id and ramdisk_id attributes for each image.

4.3.3.3 Get Image Details

The Get Image Details operation returns information about a specified image. This operation does not require a request body as the id parameter is rewritten in the URI. The response returns the corresponding image parameter.

- The image parameter is composed by the id, name, deleted, container_format, created_at disk_format, updated_at, min_disk, protected, min_ram, checksum, owner, is_public, deleted_at and size attributes as well as the properties element with the kernel_id and ramdisk_id attributes of a specific image.

4.3.3.4 Update Image

The Update Image operation updates a specified image. This operation sends the new id, name, visibility and tags parameters in the request body and the old id parameter in the URI. The response returns the updated image parameter.

4.3.3.5 Delete Image

The Delete Image deletes a specified image. This operation does not require a request body and sends the id parameter in the URI. It does not return a response body. The success or failure of this operation is returned in the response header in the Status-Line.

4.3.3.6 Upload Image

The Upload Image operation is used to upload binary image data. This operation does not require a request body and sends the id parameter and file location in the URI. It does not return a response body. The success or failure of this operation is returned in the response header in the Status-Line.

4.3.3.7 Download Image

The Download Image operation is used to download binary image data. This operation does not require a request body and sends the id parameter and file location in the URI. It does not return a response body. The success or failure of this operation is returned in the response header in the Status-Line.

4.3.3.8 Add Image Tag

The Add Image Tag operation adds a specified tag to a specified image. This operation does not require a request body and sends the id and tag parameters in the URI. It does not return a response body. The success or failure of this operation is returned in the response header in the Status-Line.

4.3.3.9 Delete Image Tag

The Delete Image Tag operation deletes a specified tag from a specified image. This operation does not require a request body and sends the id and tag parameters in the URI. It does not return a response body. The success or failure of this operation is returned in the response header Status-Line parameter.

The operations and parameters used by the Image Management component from OpenStack interface are resumed in Table 4.8.

Table 4.8: Image Management Table (Parameters vs Operations).

	Operations								
Parameters	**Create Image**	**List Images**	**Get Image Details**	**Update Image**	**Delete Image**	**Upload Image**	**Download Image**	**Add Image Tag**	**Delete Image Tag**
Sent Parameters									
name	x	x		x					
id	x	x	x	x	x	x	x	x	x
new id				x					
visibility	x			x					
tags	x			x				x	x
marker		x							
limit		x							
status		x							
member_status		x							
owner		x							
size_min		x							
size_max		x							
sort_key		x							
sort_dir		x							
file						x	x		
Received Parameters									
images		x							
image			x	x					
location	x								

4.3.4 Network Management

The Network Management component provides virtual networking services among devices that are managed by the OpenStack Compute service using the following list of operations: (*i*) List Networks; (*ii*) Show Network; (*iii*) Create Network; (*iv*) Update Network; (*v*) Delete Network; (*vi*) List Subnets; (*vii*) Show Subnet; (*viii*) Create Subnet; (*ix*) Update Subnet; (*x*) Delete Subnet; (*xi*) List Ports; (*xii*) Show Port; (*xiii*) Create Port (*xiv*) Update Port; and (*xv*) Delete Port;

4.3.4.1 List Networks

The List Networks operation lists the available networks to which the specified tenant has access. This operation does not require a request body and rewrites the tenant_id parameter in the URI. The response returns the networks parameter.

- The tenant_id parameter specifies the ID of the VM;
- The networks parameter contains the status, subnets, name, admin_state_up, tenant_id, id and shared list of attributes for every listed network.

4.3.4.2 Show Network

The Show Network operation shows information about the specified network. This operation does not require a request body and sends the tenant_id and network_id parameters in the URI. The response returns the network parameter.

- The network_id parameter contains the ID of a specific network;
- The network parameter contains the status, subnets, name, admin_state_up, tenant_id, id and shared list of attributes for the specific network.

4.3.4.3 Create Network

The Create Network operation creates a network. This operation sends the name, admin_state_up and shared parameters in the request body and the tenant_id parameter in the URI. The response returns the network parameter.

- The name parameter contains the network name;
- The admin_state_up parameter is a Boolean value that indicates the administrative status of the network;
- The shared parameter indicates whether the network is shared across all tenants. Only administrative users can change this value.

4.3.4.4 Update Network

The Update Network operation updates the specified network. This operation sends the name, admin_state_up and shared parameters in the request body as well as the tenant_id and network_id parameters in the URI. The response returns the network parameter.

4.3.4.5 Delete Network

The Delete Network operation deletes the specified network and its associated resources. This operation does not require a request body and sends the tenant_id and network_id parameters in the URI. The success or failure of this operation is returned in the response header Status-Line parameter.

4.3.4.6 List Subnets

The List Subnets operation lists the subnets to which the specified tenant has access. This operation does not require a request body and sends the tenant_id parameter in the URI. The response returns the subnets parameter.

- The subnets parameter contains , for each subnet, the allocation_pools, cidr, dns_nameservers, enable_dhcp, gateway_ip, host_routes, id, ip_version, name, network_id and tenant_id attributes.

4.3.4.7 Show Subnet

The Show Subnet operation returns data about a specific subnet. This operation does not require a request body and sends the tenant_id and subnet_id parameters in the URI. The response returns the subnet parameter.

- The subnet_id parameter contains the ID of the desired subnet. The value is obtained from the id attribute from the subnet or subnets parameters;
- The subnet parameter contains the allocation_pools, cidr, enable_dhcp, gateway_ip, id, ip_version, name, network_id and tenant_id attributes of the requested subnet.

4.3.4.8 Create Subnet

The Create Subnet operation creates a subnet on a specified network. This operation sends the allocation_pools, cidr, enable_dhcp, gateway_ip, ip_version, name and network_id parameters in the request body as well as the tenant_id parameter in the URI. The response returns the subnet parameter.

- The allocation_pools parameter specifies the start and end addresses for the allocation pools;
- The cidr parameter containing the Classless Inter-Domain Routing (CIDR) notation;
- The enable_dhcp parameter is a Boolean that indicates if DHCP is enabled (when set to True), disabled (when set to False);
- The gateway_ip parameter specifies the gateway IP address;
- The ip_version parameter contains the IP version (IPv4 or IPv6);
- The name parameter contains the subnet name.

4.3.4.9 Update Subnet

The Update Subnet operation updates a specified subnet. This operation sends the gateway_ip and name parameters in the request body as well as the tenant_id and subnet_id parameters in the URI. The response returns the subnet parameter.

4.3.4.10 Delete Subnet

The Delete Subnet operation removes a subnet from a OpenStack network. This operation does not require a request body and sends the tenant_id and subnet_id parameters in the URI. The success or failure of this operation is returned in the response header Status-Line parameter.

4.3.4.11 List Ports

The List Ports operation lists the ports to which the tenant has access. This operation does not require a request body and sends the tenant_id parameter in the URI. The response returns the ports parameter.

- The ports parameter contains the status, name, admin_state_up, network_id, tenant_id, binding:vif_type, device_owner, binding:capabilities, port_filter, mac_address, fixed_ips, id, device_id and security_groups attributes for each available port.

4.3.4.12 Show Port

The Show Port operation Shows information on a specified port. This operation does not require a request body and sends the tenant_id and port_id parameters in the URI. The response returns the port parameter.

- The port parameter contains the status, name, admin_state_up, network_id, tenant_id, binding:vif_type, device_owner, binding:capabilities, port_filter, mac_address, fixed_ips, id, device_id and security_groups attributes for the specified port.

4.3.4.13 Create Port

The Create Port operation creates a port on a specified network. This operation sends the status, name, admin_state_up, mac_address, fixed_ips, security_groups and network_id parameters in the request body as well as the tenant_id parameter in the URI. The response returns the port parameter.

- The status parameter specifies the status of the port (UP or DOWN);
- The admin_state_up parameter specifies the administrative state of the router;
- The mac_address parameter indicates the MAC address of the port;
- The fixed_ips parameter indicates the IP address and subnet ID of the fixed ports;
- The security_groups parameter indicates the ID of any attached security group.

4.3.4.14 Update Port

The Update Port operation updates a specified port. This operation sends the status, name, admin_state_up, mac_address, fixed_ips, security_groups and network_id parameters in the request body as well as the tenant_id and port_id parameters in the URI. The response returns the port parameter.

4.3.4.15 Delete Port

The Delete Port operation deletes a specified port. This operation does not require a request body and sends the tenant_id and port_id parameters in the URI. The success or failure of this operation is returned in the response header Status-Line parameter.

The operations and parameters used by the Network Management component from OpenStack interface are resumed in Table 4.9.

Table 4.9: Network Management Table (Parameters *vs* Operations).

	Operations														
Parameters	**List Networks**	**Show Network**	**Create Network**	**Update Network**	**Delete Network**	**List Subnets**	**Show Subnet**	**Create Subnet**	**Update Subnet**	**Delete Subnet**	**List Ports**	**Show Port**	**Create Port**	**Update Port**	**Delete Port**
Sent Parameters															
subnet_id							x		x	x					
tenant-id	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
port_id												x		x	x
network_id		x		x	x			x					x	x	
Admin_state_up			x	x									x	x	
name			x	x				x	x				x	x	
cidr								x							
ip_version								x							
mac_address													x	x	
fixed_ips													x	x	
allocation_pools								x							
enable_dhcp								x							
security_groups													x	x	
status													x	x	
shared			x	x											
gateway_ip								x	x						
Received Parameters															
networks	x														
network		x	x	x											
subnets						x									
subnet							x	x	x						
ports											x				
port												x	x	x	

4.3.5 Volume Management

The Volume Management component is used to provide the following operations for managing volumes in OpenStack: (*i*) Create Volume; (*ii*) List Volumes; (*iii*) List Volume Details; (*iv*) Show Volume; (*v*) Update Volume; and (*vi*) Delete Volume.

4.3.5.1 Create Volume

The Create Volume operation creates an OpenStack volume. This operation sends the name, description, size, volume_type, availability_zone and metadata parameters in the request body as well as the tenant_id parameter in the URI. The response returns the id, links and name parameters.

4.3.5.2 List Volumes

The List Volumes operation lists summary information for all Cinder volumes accessible to the tenant who submits the request. This operation sends the ten-

ant_id parameter in the URI. The response returns the id, links and name parameters for each available volume.

4.3.5.3 List Volume Details

The List Volume Details operation lists detailed information for all Cinder volumes accessible to the tenant who submits the request. This operation sends the tenant_id parameter in the URI. The response returns the volumes parameter.

- The volumes parameter contains the status, attachment, links, availability_zone, os-vol-host-attr:host, source_volid, snapshot_id, id, description, name, created_at, volume_type, os-vol-tenant-attr:tenant_id, size and metadata parameters for each available volume.

4.3.5.4 Show Volume

The Show Volume operation shows information about a specified volume. This operation sends the tenant_id and volume_id parameters in the URI. The response returns the volume parameter.

- The volume parameter contains the status, attachment, links, availability_zone, os-vol-host-attr:host, source_volid, snapshot_id, id, description, name, created_at, volume_type, os-vol-tenant-attr:tenant_id, size and metadata parameters for the specified volume.

4.3.5.5 Update Volume

The Update Volume operation updates a specific volume. This operation sends the name and description parameters in the request body as well as the tenant_id and volume_id parameters in the URI. The response returns the volume parameter.

4.3.5.6 Delete Volume

The Delete Volume operation deletes a specified volume from the pool. This operation sends the tenant_id and volume_id parameters in the URI. The success or failure of this operation is returned in the response header Status-Line parameter.

The operations and parameters used by the Volume Management component from OpenStack interface are resumed in Table 4.10.

Table 4.10: Volume Management Table (Parameters *vs* Operations).

	Operations					
Parameters	**Create Volume**	**List Volumes**	**List Volume Details**	**Show Volume**	**Update Volume**	**Delete Volume**
Sent Parameters						
availability_zone	x					
description	x				x	
tenant_id	x	x	x	x	x	x
volume_id				x	x	x
volume_type	x					
metadata	x					
name	x				x	
size	x					
Received Parameters						
name	x	x				
links	x	x				
id	x	x				
volumes			x			
volume				x	x	

4.4 CloudStack Interface

CloudStack uses a Query API to interface with the CloudStack system. This way, all CloudStack API requests are submitted by rewriting the URL of the HTTP GET/POST request messages with the required operation and parameters. A CloudStack API request presents the following syntax (HTTP/HTTPS):

```
Base URL + API Path + Operation + Signature
```

- Base URL + API Path - the CloudStack API URL, *e.g.*, `http://www.cloud.com:8080/client/api`;
- Operation - the Web services operation to be executed, *e.g.*, start a virtual machine or create a disk volume which is invoked after the question mark character;
- Signature - the additional required or optional operation parameters, *e.g.*, the signature so that CloudStack can verify the caller has been authenticated and authorized to invoke the operation.

CloudStack supports both JSON and XML response formats. The default response is XML-formatted. To change the response format to JSON it is necessary to add &response=json to the URL query string [140].

The current CloudStack interface libraries version 4.1.0 provide a variety of operation groups that are listed at [141]. The Virtual Machine, ISO, Template, Volume, Network and Firewall operation groups provide the user with functionalities common to the ones found in the other IaaS platform API under analysis.

In this section, these groups of operation are aggregated as following: (*i*) Server Management (containing the operations from the Virtual Machine group); (*ii*) Template Management; (*iii*) Image Management (containing the operations from the ISO group); (*iv*) Network Management (containing the operations from Network and Firewall groups); and (*v*) Volume Management.

4.4.1 Server Management

The Server Management component is used to manage a specific virtual machine through the following list of operations: (*i*) Deploy Virtual Machine; (*ii*) Destroy Virtual Machine; (*iii*) Reboot Virtual Machine; (*iv*) Start Virtual Machine; (*v*) Stop Virtual Machine; (*vi*) Reset Password For Virtual Machine; (*vii*) Reset SSH Key For Virtual Machine; (*viii*) Update Virtual Machine; (*ix*) List Virtual Machines; (*x*) Get VM Password; (*xi*) Restore Virtual Machine; (*xii*) Change Service For Virtual Machine; (*xiii*) Add NIC To Virtual Machine; (*xiv*) Remove NIC From Virtual Machine; (*xv*) Update Default NIC For Virtual Machine.

With the exception of the Get VM Password operation, the other Server Management operations return the same response that is composed by the id, account, cpunumber, cpuspeed, cpuused, created, displayname, domain, domainid, forvirtualnetwork, group, groupid, guestosid, haenable, hostid, hostname, hypervisor, instancename, isodisplaytext, isoid, isoname, keypair, memory, name, networkkbsread, networkkbswrite, password, passwordenabled, project, projectid, publicip, publicipid, rootdeviceid, rootdevicetype, serviceofferingid, serviceofferingname, state, templatedisplaytext, templateid, templatename, zoneid, zonename, jobid, jobstatus, nic, securitygroup and tag parameters. This common response will be referred as standard response parameter to simplify the extended list of parameters.

4.4.1.1 Deploy Virtual Machine

The Deploy Virtual Machine operation creates and automatically starts a virtual machine based on a service offering, disk offering, and template. This operation sends the serviceofferingid, templateid and zoneid required parameters and optionally the account, diskofferingid, displayname, domainid, group, hypervisor, ip6address, ipaddress, iptonetworklist, keyboard, keypair, name, networkids, projectid, securitygroupids, securitygroupnames, size, startvm and userdata parameters. The response returns the standard response parameter.

- The serviceofferingid parameter specifies the ID of the service offering for the virtual machine;
- The templateid parameter specifies the ID of the template for the virtual machine;

- The zoneid parameter specifies the zone ID for the virtual machine;
- The account parameter specifies an optional account for the virtual machine;
- The diskofferingid parameter specifies the ID of the disk offering for the virtual machine;
- The displayname parameter specifies an optional user generated name for the virtual machine;
- The domainid parameter specifies an optional domain ID for the virtual machine;
- The group parameter specifies an optional group for the virtual machine;
- The hypervisor parameter specifies the hypervisor on which to deploy the virtual machine;
- The ip6address parameter specifies the IPv6 address for a default VM's network;
- The ipaddress parameter specifies the IPv4 address for a default VM's network;
- The iptonetworklist parameter specifies the IP to network mapping;
- The keyboard parameter specifies an optional keyboard device type for the virtual machine. Valid values can be one of: de,de-ch,es,fi,fr,fr-be,fr-ch,is,it,jp,nl-be,no,pt,uk and us;
- The keypair parameter specifies the name of the SSH key pair used to login to the virtual machine;
- The name parameter specifies the host name for the virtual machine;
- The networkids parameter specifies the list of network ID used by virtual machine;
- The projectid parameter specifies the project ID;
- The securitygroupids parameter specifies a comma separated list of security groups ID that are going to be applied to the virtual machine;
- The securitygroupnames parameter specifies a comma separated list of security groups names that are going to be applied to the virtual machine;
- The size parameter specifies the arbitrary size for the DATADISK volume;

- The startvm parameter specifies if thenetwork offering supports specifying IP ranges (if true, default value) or not (if false);
- The userdata parameter specifies an optional binary data that can be sent to the virtual machine upon a successful deployment;

4.4.1.2 Destroy Virtual Machine

The Destroy Virtual Machine operation destroys a virtual machine (once destroyed, only the administrator can recover it). This operation sends the id' parameter with the virtual machine identifier. The response returns the standard response parameter.

4.4.1.3 Reboot Virtual Machine

The Reboot Virtual Machine operation reboots a virtual machine. This operation sends the id parameter with the virtual machine identifier. The response returns the standard response parameter.

4.4.1.4 Start Virtual Machine

The Start Virtual Machine operation starts a virtual machine. This operation send the id parameter with the virtual machine identifier. The response returns the standard response parameter.

4.4.1.5 Stop Virtual Machine

The Stop Virtual Machine operation stops a virtual machine. This operation sends the id parameter with the virtual machine identifier as well as the optional forced parameter to force stop the virtual machine. The response returns the standard response parameter.

4.4.1.6 Reset Password For Virtual Machine

The Reset Password For Virtual Machine operation resets the password for a virtual machine when the virtual machine is in a stopped state. This operation sends the id parameter with the virtual machine identifier. The response returns the standard response parameter.

4.4.1.7 Reset SSH Key For Virtual Machine

The Reset SSH Key For Virtual Machine operation resets the virtual machine SSH Key when the virtual machine is in a stopped state. This operation sends the id parameter with the virtual machine identifier, the keypar parameter with the name of the SSH key pair used to login into the virtual machine and, optionally,

the account, domainid and projectid parameters to associate an optional account, domain or project to the SSH key. The response returns the standard response parameter.

4.4.1.8 Update Virtual Machine

The Update Virtual Machine operation updates the properties of a virtual machine. The virtual machine has to be stopped and restarted for the new properties to take effect. This operation sends the id parameter with the virtual machine identifier and, optionally, the displayname with the user generated name, the group parameter to define the virtual machine group, the haenable parameter to enable or disable the high availability for the virtual machine, the ostypeid parameter that identifies the ID of the OS type that best represents the virtual machine and the userdata parameter containing an optional binary data to be sent to the virtual machine upon a successful deployment. The response returns the standard response parameter.

4.4.1.9 List Virtual Machines

The List Virtual Machines operation enumerates the virtual machines associated with an account. This operation sends the account, details, domainid, forvirtualnetwork, groupid, hostid, hypervisor, id, isoid, isrecursive, keyword, listall, name, networkid, page, pagesize, podid, projectid, state, storageid, tags, templateid, vpcid and zoneid parameters. The response returns the standard response parameter.

- The details parameter provides a comma separated list of host details requested. If no parameter is passed, the details will be defaulted to all;
- The forvirtualnetwork parameter lists the resources by network type;
- The hosted parameter lists the host identifier;
- The id parameter lists the identifier of the virtual machine;
- The isoid parameter lists the virtual machines by ISO;
- The isrecursive parameter lists all resources from the parent specified by the domainId parameter;
- The keyword parameter lists the resources by keyword;
- The listall parameter lists only the resources belonging to the caller (when set to false) or lists the resources that the caller is authorized to see (when the parameter's value is true). The default value is false;

- The networkid parameter lists by network ID;
- The page parameter sets the number of pages;
- The pagesize parameter sets the size of the page;
- The podid parameter lists the pod ID;
- The state parameter lists the state of the virtual machine;
- The storageid parameter lists the storage identifier holding the virtual machine's volumes;
- The tags parameter lists resources by tags (key/value pairs);
- The templateid parameter lists virtual machines by template;
- Tee vpcid parameter lists virtual machines by virtual private cloud;

4.4.1.10 Get VM Password

The Get VM Password operation returns an encrypted password for the virtual machine. This operation sends the id parameter with the virtual machine identifier and returns the encryptedpassword parameter with the encrypted password of the virtual machine.

4.4.1.11 Restore Virtual Machine

The Restore Virtual Machine operation restores a virtual machine to the original template or specific snapshot. This operation sends the virtualmachineid parameter. The response returns the standard response parameter.

4.4.1.12 Change Service For Virtual Machine

The Change Service For Virtual Machine operation changes the service offering for a virtual machine when the virtual machine is in a stopped state. This operation sends the id and the serviceofferingid parameters. The response returns the standard response parameter.

4.4.1.13 Add NIC To Virtual Machine

The Add NIC To Virtual Machine operation adds a virtual machine to the specified network by creating a NIC. This operation sends the networkid, virtualmachineid and, optionally, ipaddress parameters. The response returns the standard response parameter.

4.4.1.14 Remove NIC From Virtual Machine

The Remove NIC From Virtual Machine operation removes the virtual machine from a specified network by deleting a NIC. This operation sends the nicid and the virtualmachineid parameters. The response returns the standard response parameter.

4.4.1.15 Update Default NIC For Virtual Machine

The Update Default NIC For Virtual Machine operation changes the default NIC on a virtual machine. This operation sends the nicid and the virtualmachineid parameters. The response returns the standard response parameter.

The operations and parameters used by the Server Management component from CloudStack interface are resumed in Table 4.11.

4.4.2 Template Management

The Template Management component of CloudStack enables the management of templates using the following list of operations: (*i*) Create Template; (*ii*) Update Template; (*iii*) Copy Template; (*iv*) Delete Template; (*v*) List Templates; (*vi*) Update Template Permissions; (*vii*) List Template Permissions; and (*viii*) Extract Template.

The Update Template, Copy Template and List Templates return the same id, account, accountid, bootable, checksum, created, crossZones, details, displaytext, domain, domainid, format, hostid, hostname, hypervisor, isextractable, isfeatured, ispublic, isready, name, ostypeid, ostypename, passwordenabled, project, projectid, removed, size, sourcetemplateid, sshkeyenabled, status, templatetag, templatetype, zoneid, zonename, tags, jobid and jobstatus response parameters. This common response will be referred as standard response parameter to simplify the extended list of parameters.

4.4.2.1 Create Template

The Create Template operation creates a template of a virtual machine when the VM is in a stopped state. This operation sends the required displaytext, name and ostypeid parameters as well as the optional bits, details, isfeatured, ispublic, passwordenabled, requireshvm, snapshotid, templatetag, url, virtualmachineid and volumeid parameters. The response to this operation returns the createtemplaterespose parameter.

- The displaytext parameter displays the text of the template;
- The name parameter indicates the name of the template;

Table 4.11: Server Management Table (Parameters *vs* Operations).

Parameters	*Operations*														
	Deploy VM	**Destroy VM**	**Reboot VM**	**Start VM**	**Stop VM**	**Reset VM password**	**Reset VM SSH Key**	**Update VM**	**List VM**	**Get VM Password**	**Restore VM**	**Change VM Service**	**Add VM NIC**	**Remove VM NIC**	**Update VM NIC**
Sent Parameters															
account	x						x		x						
details									x						
diskoffering	x														
displayname	x							x							
domainid	x						x		x						
forvirtualnetwork									x						
forced					x										
group	x							x							
groupid									x						
haenable								x							
hostid	x								x						
hypervisor	x								x						
id		x	x	x	x	x	x	x	x	x		x			
ipaddress	x												x		
ip6address	x														
iptonetworklist	x														
isoid									x						
isrecursive									x						
keyboard	x														
keyword									x						
keypair	x						x								
listall									x						
name	x								x						
networkids	x								x				x		
nicid														x	x
ostypeid								x							
page									x						
pagesize									x						
podid									x						
projectid	x						x		x						
securitygroupids	x														
securitygroupnames	x														
state									x						
size	x														
startvm	x														
storageid									x						
serviceofferingid	x											x			
tags									x						
tempalteid	x								x						
userdata	x							x							
vpcid									x						
virtualmachineid											x		x	x	x
zoneid	x								x						
Received Parameters															
encriptedpassword										x					
standard response	x	x	x	x	x	x	x	x	x		x	x	x	x	x

- The ostypeid parameter indicates the ID of the OS type that best represents the OS of the template;
- The bits parameter indicates if the OS architecture is 32 bit or 64 bit;
- The details parameter indicates the template details in key/value pairs;

- The isfeatured parameter indicates if the template is a featured template (true) or not (false);
- The ispublic parameter indicates if the template is a public (true) or not (false);
- The passwordenabled parameter indicates if the template supports the password reset feature (true) or not (false). By default the value is false;
- The requireshvm parameter indicates if the template requires a Hardware VM (true) or not (false);
- The snapshotid parameter identifies the ID of the snapshot the template is being created from;
- The templatetag parameter indicates the tag for the template;
- The url parameter is only used for baremetal hypervisors and indicates the Common Internet File System (CIFS) server directory where the template is stored;
- The virtualmachineid parameter indicates the VM ID for a baremetal VM;
- The volumeid parameter indicates the ID of the disk volume the template is being created from;
- The createtemplaterespose parameter is composed by the id, clusterid, clustername, created, disksizeallocated, disksizetotal, disksizeused, ipaddress, name, path, podid, podname, state, tags, type, zoneid, zonename, jobid and jobstatus attrbutes.

4.4.2.2 Update Template

The Update Template operation updates the attributes of a template. This operation sends the required id parameter as well as the optional bootable, displaytext, format, name, ostype, passwordenabled and sortkey parameters. The response to this operation returns the standard response parameter.

- The id parameter indicates the ID of the image file;
- The bootable parameter indicates if the image is bootable (true) or not (false);
- The format parameter indicates the image format;
- The name parameter indicates the image file name;
- The sortkey parameter indicates the sort key of the template.

4.4.2.3 Copy Template

The Copy Template operation copies a template from one zone to another. This operation sends the id, destzoneid and sourcezoneid parameters. The response to this operation returns the standard response parameter.

- The destzoneid parameter indicates the ID of the zone the template is being copied to;
- The sourcezoneid parameter indicates the ID of the zone the template is currently hosted on.

4.4.2.4 Delete Template

The Delete Template operation deletes a template from the system. All virtual machines using the deleted template will not be affected. This operation sends the required id parameter as well as the optional zoneid parameter. The response to this operation returns the deletetemplaterespose parameter.

- The zoneid parameter indicates the template zone ID;
- The deletetemplateresponse parameter contaisn the displaytext and success attributes;

4.4.2.5 List Templates

The List Templates operation list all public, private and privileged templates. This operation sends the required templatefilter parameter as well as the optional account, domainid, hypervisor, id, isrecursive, keyword, llistall, name, page, pagesize, projectid, tags and zoneid. The response to this operation returns the standard response parameter.

- The templatefilter parameter indicates the filter values;
- The account parameter lists resources by account;
- The domainid parameter lists only the resources belonging to the specified domain;
- The hypervisor parameter identifies the hypervisor under search;
- The keyword parameter lists the resources by keyword;
- The listall parameter lists only resources belonging to the caller (if set to false) or list resources that the caller is authorized to see (if set to true). Default value is false;

- The projectid parameter lists objects by project;
- The tags parameter lists resources by tags (key/value pairs).

4.4.2.6 Update Template Permissions

The Update Template Permissions updates a template visibility permissions. A public template is visible to all accounts within the same domain. A private template is visible only to the owner of the template. A privileged template is a private template with account permissions added. This operation sends the required id parameter and the optional acounts, isextractable, isfeatured, ispublic, op and projectids parameters. The response to this operation returns the updatetemplatepermresponse parameter.

- The accounts parameter holds a comma delimited list of accounts;
- The isextractable parameter indicates if the template/ISO is extractable (if true) or not (if false);
- The isfeatured parameter indicates if is a featured template/ISO (if true) or not (if false);
- The ispublic parameter indicates if the template/ISO is public (if true) or private (if false);
- The op parameter represents the permission operator (add, remove, reset);
- The projectids parameter holds a comma delimited list of projects;
- The updatetemplatepermresponse parameter contains the displaytext and success attributes.

4.4.2.7 List Template Permissions

The List Template Permissions operation lists the template visibility and all accounts that have permissions to view the template. This operation sends the id parameter and returns the listtemplatepermresponse parameter.

- The listtemplatepermresponse parameter contains the id, account, domainid, ispublic and projectids attributes.

4.4.2.8 Extract Template

The Extract Template operation extracts a template. This operation sends the required id and mode parameters as well as the optional url and zoneid parameters. The response to this operation returns the extracttemplateresponse parameter.

- The extracttemplateresponse parameter contains the id, accountid, created, extractId, extractMode, name, state, status, storagetype, uploadpercentage, url, zoneid and zonename attributes.

The operations and parameters used by the Template Management component from CloudStack interface are resumed in Table 4.12.

4.4.3 Image Management

CloudStack supports the Image image format for defining guest virtual machines. The Image Management component manages ISO images through the following operations: (*i*) Attach ISO; (*ii*) Detach ISO; (*iii*) List ISO; (*iv*) Update ISO; (*v*) Delete ISO; (*vi*) Copy ISO; (*vii*) Update ISO Permissions; (*viii*) List ISO Permissions; and (*ix*) Extract ISO.

The List ISO, Update ISO and Copy ISO operations return the same id, account, accountid, bootable, checksum, created, crossZones, details, displaytext, domain, domainid, format, hostid, hostname, hypervisor, isdynamicallyscalable, isextractable, isfeatured, ispublic, isready, name, ostypeid, ostypename, passwordenabled, project, projectid, removed, size, sourcetemplateid, sshkeyenabled, status, templatetag, templatetype, zoneid, zonename, tags, jobid and jobstatus response parameters. This common response will be referred as standard response parameter to simplify the extended list of parameters.

4.4.3.1 Attach ISO

The Attach ISO operation attaches an ISO image to a virtual machine. This operation sends the id and virtualmachineid parameters and returns the attachisoresponse parameter.

- The attachissoresponse parameter contains the id, account, cpunumber, cpuspeed, cpuused, created, diskioread, diskiowrite, diskkbsread, diskkbswrite, displayname, displayvm, domain, domainid, forvirtualnetwork, group, groupid, guestosid, haenable, hostid, hostname, hypervisor, instancename, isdynamicallyscalable, isodisplaytext, isoid, isoname, keypair, memory, name, networkkbsread, networkkbswrite, password, passwordenabled, project, projectid, publicip, publicipid, rootdeviceid, rootdevicetype, serviceofferingid, serviceofferingname, servicestate, state, templatedis-

Table 4.12: Template Management Table (Parameters *vs* Operations).

	Operations							
Parameters	**Create Template**	**Update Template**	**Copy Template**	**Delete Template**	**List Templates**	**Update Template Permissions**	**List Template Permissions**	**Extract Template**
Sent Parameters								
account					x	x		
bits	x							
bootable		x						
details	x							
destzoneid			x					
displaytext	x	x						
domainid					x			
format		x						
hypervisor					x			
id		x	x	x	x	x	x	x
isexecutable						x		
isfeatured	x					x		
ispublic	x					x		
isrecursive					x			
keyword					x			
keypair								
listall					x			
mode								x
name	x	x			x			
op						x		
ostypeid	x	x						
page					x			
pagesize					x			
passwordenable	x	x						
projectid					x	x		
requireshvm	x							
sortkey		x						
sourcezoneid			x					
snapshotid	x							
tags					x			
tempaltetag	x							
templatefilter					x			
url	x							x
virtualmachineid	x							
volumeid	x							
zoneid				x	x			x
Received Parameters								
createtemplateresponse	x							
deletetemplateresponse				x				
updatetemplatepermresponse						x		
listtemplatepermresponse							x	
extracttemplateresponse								x
standard response		x	x		x			

playtext, templateid, templatename, zoneid, zonename, affinitygroup, nic, securitygroup, tags, jobid and jobstatus attributes.

4.4.3.2 Detach ISO

The Detach ISO operation detaches any ISO file currently attached to a virtual machine. This operation sends the virtualmachineid parameter and returns the detachisoresponse parameter.

- The detachisorespose parameter contains the same list of attributes as the attachisoresponse parameter.

4.4.3.3 List ISO

The List ISO operation lists all available ISO files. This operation sends the optional account, bootable, domainid, hypervisor, id, isofilter, ispublic, isready, isrecursive, keyword, listall, name, page, pagesize, projectid, tags and zoneid parameters and returns the standard response parameter.

4.4.3.4 Update ISO

The Update ISO operation updates an ISO file. This operation sends the required id parameter and the optional bootable, displaytext, format, isdynamicallyscalable, isrouting, name, ostype, passwordenabled and sortkey parameters. The response returns the standard response parameter.

- The bootable parameter indicates if the image is bootable (if true) or not (if false);
- The isdynamicallyscalable is a boolean parameter that indicates if the ISO image contains Xen Server or VMWare tools in order to support dynamic scaling of VM CPU or memory (true) or not (false);
- The isrouting parameter indicates if the template type is routing *i.e.*, if the template is used to deploy a router.

4.4.3.5 Delete ISO

The Delete ISO operation deletes an ISO file. This operation sends the required id parameter and the optional zoneid parameter and returns the deleteisoresponse parameter.

- The deleteisoresponse parameter contains the displaytext and success attributes.

4.4.3.6 Copy ISO

The Copy ISO operation copies an ISO from one zone to another. This operation sends the id, destzoneid and sourcezoneid parameters and returns the standard response parameter.

4.4.3.7 Update ISO Permissions

The Update ISO Permissions updates the ISO file permissions. This operation sends the required id parameter and the optional accounts, isextractable, isfeatured, ispublic, op and projectids parameters. The response message return the updateissopermresponse parameter.

- The updateisopermresponse parameter contains the displaytext and success attributes.

4.4.3.8 List ISO Permissions

The List ISO Permissions lists the visibility and all accounts that have permissions to view the referred ISO image. This operation sends the id parameter and returns the listisopermresponse parameter.

- The listisopermresponse parameter contains the id, account, domainid, ispublic and projectids attributes.

4.4.3.9 Extract ISO

The Extract ISO operation extracts an ISO image. This operation sends the required id and mode parameters and the optional url and zoneid parameters. The response message returns the extractisoresponse parameter.

- The extractisoresponse parameter ontains the id, accountid, created, extractId, extractMode, name, state, status, storagetype, uploadpercentage, url, zoneid and zonename attributes.

The operations and parameters used by the Image Management component from CloudStack interface are resumed in Table 4.13.

Table 4.13: Image Management Table (Parameters *vs* Operations).

	Operations								
Parameters	**Attach ISO**	**Detach ISO**	**List ISO**	**Update ISO**	**Delete ISO**	**Copy ISO**	**Update ISO Permissions**	**List ISO Permissions**	**Extract ISO**
Sent Parameters									
account			x				x		
bootable			x	x					
destzoneid						x			
displaytext				x					
domainid			x						
format				x					
hypervisor			x						
id	x		x	x	x	x	x	x	x
isextractable							x		
isfeatured							x		
ispublic			x				x		
isready			x						
isrecursive			x						
isdinamicallyscalable				x					
isofilter			x						
isrouting				x					
keyword			x						
listall			x						
mode									x
name			x	x					
op							x		
ostypeid				x					
page			x						
pagesize			x						
passwordenable				x					
projectid			x				x		
sortkey				x					
sourcezoneid						x			
tags			x						
url									x
virtualmachineid	x	x							
zoneid			x		x				x
Received Parameters									
attachisoresponse	x								
detachisoresponse		x							
standard response			x	x		x			
deleteisoresponse					x				
updateisopermresponse							x		
listisopermresponse								x	
extractisoresponse									x

4.4.4 Network Management

The Network Management component is used to manage port forwarding and firewall rules through the following list of operations: (*i*) List Port Forwarding Rules; (*ii*) Create Port Forwarding Rule; (*iii*) Delete Port Forwarding Rule; (*iv*) Update Port Forwarding Rule; (*v*) Create Firewall Rule; (*vi*) Delete Firewall Rule; (*vii*) List Firewall Rules; (*viii*) Create Egress Firewall Rule; (*ix*) Delete Egress Firewall Rule; (*x*) List Egress Firewall Rules; (*xi*) Create Network; (*xii*)

Delete Network; (*xiii*) List Networks; (*xiv*) Restart Network; and (*xv*) Update Network;

The List Port Forwarding Rules, Create Port Forwarding Rule, Update Port Forwarding Rule, Create Firewall Rule, List Firewall Rules, Create Egress Firewall Rule and List Egress Firewall Rules return the same id, cidrlist, ipaddress, ipaddressid, privateendport, privateport, protocol, publicendport, publicport, state, virtualmachinedisplayname, virtualmachineid, virtualmachinename and tags response parameters. This common response will be referred as firewall response parameter to simplify the extended list of parameters.

The Create Network, List Networks and Update Network return the same id, account, acltype, broadcastdomaintype, broadcasturi, canusefordeploy, cidr, displaytext, dns1, dns2, domain, domainid, gateway, ip6cidr, ip6gateway, isdefault, ispersistent, issystem, name, netmask, networkdomain, networkofferingavailability, networkofferingdisplaytext, networkofferingid, networkofferingname, physicalnetworkid, project, projectid, related, restartrequired, specifyipranges, state, subdomainaccess, traffictype, type, vlan, vpcid, zoneid, zonename, service and tags response parameters. This common response will be referred as network response parameter to simplify the extended list of parameters.

4.4.4.1 List Port Forwarding Rules

The List Port Forwarding Rules operation lists all port forwarding rules for an IP address. This operation sends the account, domainid, id, ipaddressid, isrecursive, keyword, listall, page, pagesize, projectid and tags parameters and returns the firewall response parameter.

4.4.4.2 Create Port Forwarding Rule

The Create Port Forwarding Rule operation creates a port forwarding rule. This operation sends the required ipaddressid, privateport, protocol, publicport and virtualmachineidparameters as well as the optional cidrlist, networkid, openfirewall, privateendport and publicendport parameters. The response to this operation returns the firewall response parameter.

4.4.4.3 Delete Port Forwarding Rule

The Delete Port Forwarding Rule operation deletes a port forwarding rule. This operation sends the id parameter with the identifier of the port forwarding rule and returns the displaytext and success parameters.

4.4.4.4 Update Port Forwarding Rule

The Update Port Forwarding Rule operation updates the private port and the virtual machine information of a port forwarding rule. This operation sends the required ipaddressid, privateport, protocol and publicport parameters as well as the optional privateip and virtualmachineid parameters. The response to this operation returns the the firewall response parameter.

4.4.4.5 Create Firewall Rule

The Create Firewall Rule operation creates a firewall rule for a given IP address. This operation sends the required ipaddressid and protocol parameters as well as the optional cidrlist, endport, icmpcode, icmptype, startport and type parameters. The response to this operation returns the firewall response parameter.

4.4.4.6 Delete Firewall Rule

The Delete Firewall Rule operation deletes a firewall rule. This operation sends the id parameter with the identifier of the firewall rule and returns the displaytext and success parameters.

4.4.4.7 List Firewall Rules

The List Firewall Rules operation lists all firewall rules for an IP address. This operation sends the optional account, domainid, id, ipaddressid, isrecursive, keyword, listall, page, pagesize, projectid and tags parameters and returns the firewall response parameter.

4.4.4.8 Create Egress Firewall Rule

The Create Egress Firewall Rule operation creates a egress firewall rule for a given network. This operation sends the required networkid and protocol parameters as well as the optional cidrlist, endport, icmpcode, icmptype, startport and type parameters. The response to this operation returns the firewall response parameter.

4.4.4.9 Delete Egress Firewall Rule

The Delete Egress Firewall Rule operation deletes an egress firewall rule. This operation sends the id parameter with the identifier of the firewall rule and returns the displaytext and success parameters.

4.4.4.10 List Egress Firewall Rules

The List Egress Firewall Rules operation lists all egress firewall rules for a network identifier. This operation sends the optional account, domainid, id, ipaddressid, isrecursive, keyword, listall, networkid, page, pagesize, projectid and tags parameters and returns the firewall response parameter.

4.4.4.11 Create Network

The Create Network operation creates a CloudStack network. This operation sends the required displaytext, name, networkofferingid, zoneid parameters and the optional account, acltype, domainid, endip, endipv6, gateway, ip6cidr, ip6gateway, netmask, networkdomain, physicalnetworkid, projectid, startip, startipv6, subdomainaccess, vlan and vpcid parameters. The response returns the network response parameter.

- The displaytext parameter specifies the display text of the network;
- The networkofferingid parameter specifies the network offering ID;
- The account parameter specifies the account who will own the network;
- The acltype parameter specifies the access control type;
- The endip parameter specifies the ending IP address in the network IP range. If not specified, will be defaulted to startIP;
- The endipv6 parameter specifies the ending IPv6 address in the IPv6 network range;
- The gateway parameter specifies the gateway of the network;
- The ip6cidr parameter specifies the CIDR of IPv6 network (must be at least /64);
- The ip6gateway parameter specifies the gateway of the IPv6 network;
- The netmask parameter specifies the netmask of the network;
- The physicalnetworkid parameter specifies the network Physical Network ID;
- The startip parameter specifies the initial IP address in the network IP range;
- The startipv6 parameter specifies the initial IPv6 address in the IPv6 network range;

- The subdomainaccess parameter defines if subdomains can use their parent dedicated networks domain(s);
- The vlan parameter specifies the ID or VID of the network;
- The vpcid parameter specifies the ID of the Virtual Private Cloud (VPC)) the network belongs to.

4.4.4.12 Delete Network

The Delete Network operation deletes a CloudStack network. This operation sends the id parameter of the network and returns the displaytext and success parameters.

4.4.4.13 List Networks

The List Networks operation lists all available networks. This operation sends the account, acltype, canusefordeploy, domainid, forvpc, id, isrecursive, issystem, keyword, listall, page, pagesize, physicalnetworkid, projectid, restartrequired, specifyipranges, supportedservices, tags, traffictype, type, vpcid and zoneid parameters. The response returns the network response parameter.

- The canusefordeploy parameter lists the networks available for VM deployment;
- The domainid parameter lists the resources belonging to the specified domain;
- The forvpc parameter specifies the network as belonging to a VPC;
- The physicalnetworkid parameter lists networks by physical network ID;
- The restartrequired parameter lists networks by restartRequired;
- The specifyipranges parameter indicates whether to show (true) or not (false) the IP ranges of the networks;
- The supportedservices parameter lists the networks supporting certain services;
- The traffictype parameter indicates the type of traffic;
- The type parameter indicates the type of the network (Isolated or Shared);
- The vpcid parameter lists networks by VPC.

4.4.4.14 Restart Network

The Restart Network operation restarts a CloudStack network. This operation sends the id and cleanup parameters and returns the restartnetworkresponse parameters.

- The restartnetworkparameter contains the id, account, allocated, associatednetworkid, associatednetworkname, domain, domainid, forvirtualnetwork, ipaddress, issourcenat, isstaticnat, issystem, networkid, physicalnetworkid, project, projectid, purpose, state, virtualmachinedisplayname, virtualmachineid, virtualmachinename, vlanid, vlanname, vpcid, zoneid, zonename, tags, jobid and jobstatus attributes.

4.4.4.15 Update Network

The Update Network operation updates a CloudStack network. This operation sends the network id parameter and the optional changecidr, displaytext, name, networkdomain and networkofferingid parameters. The response returns the network response parameter.

- The changecidr parameter forces the update even if the CIDR type is different;
- The displaytext parameter specifies the new display text for the network;
- The name parameter specifies the new name for the network;
- The networkdomain parameter identifies the network domain;
- The networkofferingid identifies the network offering ID.

The operations and parameters used by the Network Management component from CloudStack interface are resumed in Table 4.14.

Table 4.14: Network Management Table (Parameters *vs* Operations).

	Operations														
Parameters	**List Port Forwarding Rules**	**Create Port Forwarding Rule**	**Delete Port Forwarding Rule**	**Update Port Forwarding Rule**	**Create Firewall Rule**	**Delete Firewall Rule**	**List Firewall Rules**	**Create Firewall Rule**	**Delete Egress Firewall Rule**	**List Egress Firewall Rules**	**Create Network**	**Delete Network**	**List Networks**	**Restart Network**	**Update Network**
Sent Parameters															
account	x						x			x	x		x		
acltype											x		x		
canusefordeploy													x		
changecidr															x
cidrlist		x			x			x							
cleanup														x	
displaytext											x				x
domainid	x						x			x	x		x		
endip											x				
endipv6											x				
endport					x			x							
forvpc													x		
gateway											x				
icmpcode					x			x							
icmptype					x			x							
id	x		x		x	x	x		x	x		x	x	x	x
ipaddressid	x	x		x	x		x			x					
ipv6cidr											x				
ipv6gateway											x				
isrecursive	x						x			x			x		
issystem													x		
keyword	x						x			x			x		
listall	x						x			x			x		
name											x				x
netmask											x				
networkid		x						x		x					
networkdomain											x				x
networkofferingid											x				x
openfirewall		x													
page	x						x			x			x		
pagesize	x						x			x			x		
physicalnetworkid											x		x		
privateendport		x		x											
privateip				x											
privateport		x		x	x										
projectid	x						x			x	x		x		
protocol		x		x	x			x							
publicendport		x		x											
publicport		x		x	x										
restartrequired													x		
specifyipranges													x		
startip											x				
startipv6											x				
startport					x			x							
state				x											
subdomainaccess											x				
supportedservices													x		
tags	x			x			x			x			x		
traffictype													x		
type					x			x					x		
virtualmachinedisplayname				x											
virtualmachineid		x		x											
virtualmachinename				x											
vlan											x				
vpcid											x		x		
zoneid													x		
Received Parameters															
firewall response	x	x		x	x		x	x		x					
network response											x		x		x
restartnetworkresponse														x	
displaytext			x			x			x			x			
success			x			x			x			x			

4.4.5 Volume Management

The Volume Management component is used to manage CloudStack volumes through the following list of operations: (*i*) Attach Volume; (*ii*) Upload Volume; (*iii*) Detach Volume; (*iv*) Create Volume; (*v*) Delete Volume; (*vi*) List Volumes; (*vii*) Extract Volume; (*viii*) Migrate Volume; and (*ix*) Resize Volume.

With the exception of the Delete Volume and the Extract Volume operations, the other volume Management operations return the same response that is composed by the id, account, attached, created, destroyed, deviceid, diskofferingdisplaytext, diskofferingid, diskofferingname, domain, domainid, hypervisor, isextractable, name, project, projectid, serviceofferingdisplaytext, serviceofferingid, serviceofferingname, size, snapshotid, state, status, storage, storagetype, type, virtualmachineid, vmdisplayname, vmname, vmstate, zoneid, zonename, tags, jobid and jobstatus parameters. This common response will be referred as standard response parameter to simplify the extended list of parameters.

4.4.5.1 Attach Volume

The Attach Volume operation attaches a disk volume to a CloudStack virtual machine. This operation sends the id and virtualmachineid parameters as well as the optional deviceid parameter. The response returns the standard response parameter.

4.4.5.2 Upload Volume

The Upload Volume operation uploads a data disk. This operation sends the format, name, url, zoneid parameters as well as the optional account, checksum and domainid parameters. The response returns the standard response parameter.

4.4.5.3 Detach Volume

The Detach Volume operation detaches a disk volume from a CloudStack virtual machine. This operation sends the deviceid, id and virtualmachineid parameters. The response returns the standard response parameter.

4.4.5.4 Create Volume

The Create Volume operation creates a disk volume from a disk offering for further attachment. This operation sends the name parameter as well as the optional account, diskofferingid, domainid, projectid, size, snapshotid and zoneid parameters. The response returns the standard response parameter.

4.4.5.5 Delete Volume

The Delete Volume operation deletes a detached disk volume. This operation sends the id parameter and returns the displaytext and success parameters.

4.4.5.6 List Volumes

The List Volumes operation list the available CloudStack volumes. This operation sends the optional account, domainid, hostid, id, isrecursive, keyword, listall, name, page, pagesize, podid, projectid, tags, type, virtualmachineid and zoneid parameters. The response returns the standard response parameter.

4.4.5.7 Extract Volume

The Extract Volume operation extracts a specific volume. This operation sends the id, mode, zoneid parameters as well as the optional url parameter. The response returns the extractvolumeresponse parameter.

- The extractvolumeresponse parameter contains the id, accountid, created, extractId, extractMode, name, state, status, storagetype, uploadpercentage, url, zoneid and zonename attributes.

4.4.5.8 Migrate Volume

The Migrate Volume operation migrates a volume to other storage pool. This operation sends the storageid and volumeid parameters. The response returns the standard response parameter.

4.4.5.9 Resize Volume

The Resize Volume operation resizes a specific volume. This operation sends the optional diskofferingid, id, shrinkok and size parameters. The response returns the standard response parameter.

The operations and parameters used by the Volume Management component from CloudStack interface are resumed in Table 4.15.

Table 4.15: Volume Management Table (Parameters *vs* Operations).

	Operations								
Parameters	**Attach Volume**	**Update Volume**	**Detach Volume**	**Create Volume**	**Delete Volume**	**List Volumes**	**Extract Volume**	**Migrate Volume**	**Resize Volume**
Sent Parameters									
account		x		x		x			
checksum		x							
deviceid	x		x						
diskofferingid				x					x
domainid		x		x		x			
format		x							
hostid						x			
id	x		x		x	x	x		x
isrecursive						x			
keyword						x			
listall						x			
mode							x		
name		x		x		x			
page						x			
pagesize						x			
podid						x			
projectid				x		x			
shrinkok									x
size				x					x
snapshotid				x					
storageid								x	
tags						x			
type						x			
url		x					x		
viirtualmachineid	x		x			x			
volumeid								x	
zoneid		x		x		x	x		
Sent Parameters									
standard response	x	x	x	x		x		x	x
extractvolumeresponse							x		
displaytext					x				
success					x				

4.5 PACI Interface

This interface is accessible through a base URL with the following syntax:

```
https://{ip_address | hostname}:port/paci/version
```

The request URL is composed by the IP address or hostname and listening port of the PACI server, followed by the path containing the PACI string and the version of the REST API. The PACI RESTful API allows the end user to manage the system through lists of components available at [142]. This analysis will be focussed on the common functionalities between PACI API and the API of the other IaaS platforms under comparison. In the case of PACI interface library,

the common functionalities include the management of: (*i*) Servers; (*ii*) Firewall; (*iii*) Application Template Installations; and (*vi*) Images.

4.5.1 Server Management

The Server Management component is used to obtain information and perform actions on individual servers through the following list of operations: (*i*) List Servers; (*ii*) Start/Stop a Server; (*iii*) Create Server; (*iv*) Create Server From Image; (*v*) Clone Server; (*vi*) Modify Server Configuration; (*vii*) Reset Server Administrator Password; (*viii*) Obtain Server Information; (*ix*) Obtain Server History; (*x*) Delete Server; (*xi*) Set Backup Schedule; (*xii*) Cancel Backup Schedule; (*xiii*) List Backup Schedule; and (*xiv*) Restore a Server.

4.5.1.1 List Servers

The List Servers operation is used to obtain the list of servers owned by the current user. This operation does not require a request body and sends the subscription-id query parameter in the URL. The response returns the ve-info parameter for each listed server.

- The ve-info parameter contains individual server information in the description, state, name and subscription-id attributes.

4.5.1.2 Start/Stop a Server

Start/Stop a Server is used to start or stop a specified server. This operation does not require a request body sending the ve-name and the action parameters in the URI path. The response returns a text message describing the state of the operation, *i.e.*, if the server is initiated or stopped.

- The ve-name parameter specifies the server name;
- The action parameter must be substituted by the action start or stop.

4.5.1.3 Create Server

The Create Server operation is used to request the creation of a new server. This operation send the name, description, subscription-id, cpu, ram-size, bandwidth, no-of-public-ip, no-of-public-ipv6, ve-disk, template-info, os-info, backup-schedule and admin parameters. The response returns a pwd-response parameters.

- The name parameter is a string type and specifies the server name;

- The description parameter is a string type and specifies the description of the new server;
- The subscription-id specifies the Subscription ID number (type: int) for which to create the new server and must be included and populated when the customer account to which the user belongs has more than one subscription. It can be omitted otherwise;
- The cpu parameter specifies information about the server CPU unit and includes the number and power attributes do define the number of cores and their clock frequency in MHz;
- The ram-size parameter specifies the quantity of RAM in megabytes the server will contain;
- The bandwidth parameter specifies the attributed network bandwidth in Kib/s;
- The no-of-public-ip specifies the number of desired public IPv4 addresses;
- The no-of-public-ipv6 specifies the number of desired public IPv6 addresses;
- The ve-disk specifies information about the server's hard disk and contains the local, primary and size attributes to define if it is a local or network hard disk, if it should be a system disk and the storage capacity in GiB;
- The template-info parameter provides information about the OS template and includes the name attribute of the OS template name;
- The os-info parameter provides information about the OS and includes the technology and type attributes to define the virtualization technology to use (virtual machine or container) and the OS type;
- The backup-schedule is an optional parameter to provide information about the backup schedule. It contains the name attribute to define de schedule type from a provided list.
- The admin parameter specifies the administrator credentials using the login and password attributes;
- The pwd-response parameter contains the message and password attributes.

4.5.1.4 Create Server From Image

The Create Server From Image operation is used to request the creation of a server from an existing image. This operation does no require a request body and sends the ve-name, image-name and the subscription-id parameters in the

URI path. The response returns a text message describing the status of the operation.

- The image-name parameter specifies the image name;

4.5.1.5 Clone Server

The Clone Server operation is used to request the creation of an exact copy of an existing server. This operation does not require a request body and sends the ve-name and new-server-name parameters in the URI path. The response returns the pwd-response parameter.

- The new-server-name parameter specifies the name of the cloned server;

4.5.1.6 Modify Server Configuration

The Modify Server Configuration operation is used to request the modification of an existing server configuration. This operation sends the reconfigure-ipv4, reconfigure-ipv6, primary-disk-size parameters in the request body and the ve-name parameter in the URI path. The response returns a text message describing the state of the operation.

- The reconfigure-ipv4 and reconfigure-ipv6 parameters contain the description, change-cpu, ram-size, bandwidth, add-ipand drop-ip elements;
- The primary-disk-size parameter modifies the storage capacity in GiB.

4.5.1.7 Reset Server Administrator Password

The Reset Server Administrator Password operation is used to request the server administrator password regeneration. This operation does not require a request body and sends the ve-name parameter in the URI path. The response returns the pwd-response parameter.

4.5.1.8 Obtain Server Information

The Obtain Server Information operation is used to request information about a specified server. This operation does not require a request body and sends the ve-name parameter in the URI path. The response returns the ve parameter.

- The ve parameter contains the id, uuid, hnid, customer-id, name, description, subscription-id, cpu, ram-size, bandwidth, ve-disk, platform, network, backup-schedule, state, primary-disk-id, template-id, admin, last-operation-rc, app-info, load-balancer and steady-state elements.

4.5.1.9 Obtain Server History

The Obtain Server History operation is used to request the modification history for the specified server. This operation does not require a request body and sends the ve-name, records, from-inclusive and to-exclusive parameters in the URI path. The response returns the ve-history parameter.

- The records parameter specifies the number of records (from the end) to include in the result set;
- The from-inclusive and to-exclusive parameters are used to retrieve the records that were created during the specified date-time period;
- The ve-hostory parameter is composed by a list of multiple ve-snapshot elements containing the cpu, ram, local-disk, nbd, bandwidth, backup-scheme, last-operation-rc, last-touched-from, state, steady-state, last-changed-by, event-timestamp, no-of-public-ip, no-of-public-ipv6, is-lb, private-incoming-traffic, private-outgoing-traffic, public-incoming-traffic and public-outgoing-traffic attributes.

4.5.1.10 Delete Server

The Delete Server operation is used to permanently delete a server. This operation does not require a request body and sends the ve-name parameter in the URI path. The response returns a message describing the status of the operation.

4.5.1.11 Set Backup Schedule

The Set Backup Schedule operation is used to assign a backup schedule to the specified server. This operation does not require a request body and sends the ve-name and schedule-name parameters in the URI path. The response returns a text message describing the status of the operation.

- The schedule-name parameter specifies the name of an existing backup schedule.

4.5.1.12 Cancel Backup Schedule

The Cancel Server Backup Schedule operation is used to cancel a backup schedule assigned to a server. This operation does not require a request body and sends the ve-name parameter in the URI path. The response returns a text message describing the status of the operation.

4.5.1.13 List Backup Schedule

The List Backups operation is used to list the available backups for the specified server. This operation does not require a request body and sends the ve-name, from-inclusive and to-exclusive parameters in the URI path. The response returns a ve-backups parameter.

- The ve-backups parameter includes a list of backup elements constituted by the in-backup-id, cloud-backup-id, schedule-name, started, ended, successful, backup-size, backup-node-name and delta-of attributes.

4.5.1.14 Restore a Server

The Restore a Server operation is used to restore a specified server from a specified backup. This operation does not require a request body and sends the ve-name and cloud-backup-id parameters in the URI path. The response returns a text message describing the status of the operation.

- The cloud-backup-id parameter specifies the backup ID.

The operations and parameters used by the Server Management component from PACI interface are resumed in Table 4.16.

Table 4.16: Server Management Table (Parameters *vs* Operations).

	Operations													
Parameters	**List Servers**	**Start/Stop a Server**	**Create Server**	**Create Server from Image**	**Modify Server Configuration**	**Clone Server**	**Reset Server Admin. Password**	**Obtain Server Info.**	**Obtain Server History**	**Delete Server**	**Set Backup Schedule**	**Cancel Backup Schedule**	**List Backups**	**Restore Server**
Sent Parameters														
name			x											
description			x											
subscription-id	x		x	x										
cpu			x											
ram-size			x											
bandwidth			x											
no-of-public-ip			x											
no-of-public-ipv6			x											
ve-disk			x											
template-info			x											
os-info			x											
backup-schedule			x											
admin			x											
primary-disk-size						x								
from-inclusive									x					
to-exclusive									x				x	
records									x				x	
schedule-name											x			
ve-name		x		x	x	x	x	x	x	x	x	x	x	x
image-name				x										
new-server-name					x									
reconfigure-ipv4						x								
reconfigure-ipv6						x								
cloud-backup-id														x
Received Parameters														
ve-info	x													
text message		x		x		x				x	x	x		x
pwd-response			x		x		x							
ve								x						
ve-history									x					
ve-backups													x	

4.5.2 Firewall Management

The Firewall Management component is used to manage the firewall rules of a virtual machine through the following list of operations: (*i*) List Firewall Rules; (*ii*) Create Firewall Rule; (*iii*) Modify Firewall Rule; and (*iv*) Delete Firewall Rule.

4.5.2.1 List Firewall Rules

The List Firewall Rules operation requests a list of existing firewall rules for the specified server. This operation does not require a request body and sends the ve-name parameter in the URI path. The response returns the rule and remote-net parameters.

- The rule parameter is constituted by the id, name, protocol, local-port and remote-port attributes containing the firewall information for each rule;
- The remote-net parameter specifies the remote address and an optional mask.

4.5.2.2 Create Firewall Rule

The Create Firewall Rules operation requests the creation of firewall rules for the specified server. This operation has a rule and remote-net parameters in the request body as well as the ve-name parameter in the URI path. The response returns a text message describing the operation status.

4.5.2.3 Modify Firewall Rule

The Modify Firewall Rules operation requests the modification of the existing firewall rules. This operation sends the rule and remote-net parameters in the request body as well as the ve-name parameter in the URI path. The response returns a text message describing the operation status.

4.5.2.4 Delete Firewall Rule

The Delete Firewall Rules operation is used to delete all existing firewall rules of a specified server. This operation does not require a request body and sends the ve-name parameter in the URI path. The response returns a text message describing the status of the operation.

The operations and parameters used by the Firewall Management component from PACI interface are resumed in Table 4.17.

Table 4.17: Firewall Management Table (Parameters *vs* Operations).

	Operations			
Parameters	**List Firewall Rules**	**Modify Firewall Rule**	**Create Firewall Rule**	**Delete Firewall Rule**
Sent Parameters				
ve-name	x	x	x	x
rule		x	x	
remote-net		x	x	
Received Parameters				
text message		x	x	x
rule	x			
remote-net	x			

4.5.3 Application Template Management

The Application Template Management component is used to obtain information about the available application templates and to install them in servers. An application template represents a software application that can be installed in a server. This component is created for a specific operating system that must be compatible with the OS template used to create a server. The available operations are: (*i*) List Application Templates; (*ii*) Get Application Templates Information; and (*iii*) Install Application Templates.

4.5.3.1 List Application Templates

The List Application Templates operation is used to list the available application templates. This operation has no request parameters. The response returns the application-list parameter.

- The application-list parameter contains a list of application-template elements with the id, name, active, c2u-version and for-os attributes.

4.5.3.2 Get Application Templates Information

The Get Application Templates Information operation is used to obtain a detailed information about a specified application template. This operation does not require a request body and sends the name and for-os parameters in the URI. The response returns the application-template parameter.

- The application-template parameter contains the id, name, active, c2u-version and for-os attributes.

4.5.3.3 Install Application Templates

The Install Application Templates operation is used to install an application template into a server. The application template must be compatible with the OS template installed in the target server. This operation does not require a request body and sends the ve-name and app-name parameters in the URI. The response returns a text message describing the status of the operation.

- The app-name parameter specifies the application template name to be installed.

The operations and parameters used by the Application Template Management component from PACI interface are resumed in Table 4.18.

Table 4.18: Application Template Management Table (Parameters *vs* Operations).

	Operations		
Parameters	**List Application Template**	**Get Application Template**	**Install Application Template**
Sent Parameters			
app-name			x
for-os		x	
id		x	
ve-name			x
Received Parameters			
application-list	x		
application-template		x	
text message			x

4.5.4 Image Management

The Image Management component provides operations to manage server images. A server image is created from an existing server and can be used later to create new servers. The available operations are: (*i*) List Images; (*ii*) Get Image Information; (*iii*) Create Image; and (*iv*) Delete Image.

4.5.4.1 List Images

The List Images operation is used to obtain a list of the existing server images. This operation has no request parameters and returns the image-list parameter.

- The image-info parameter contains a list of image-info elements with the name, size, created, subscription-id, load-balancer, active and image-of attributes.

4.5.4.2 Get Image Information

The Get Image Information operation is used to obtain a detailed information for the specified server image. This operation does not require a request body and sends the image-name parameter in the URI. The response returns the ve-image parameter.

- The image-name parameter specifies the name of the image;
- The ve-image parameter contains the id, bnode-uuid, customer-id, subscription-id, load-balancer, active, image-of, name, cpu-number, cpu-power, ram-size, bandwidth, login, template-id, primary-disk-id, image-size, created, no-of-public-ip, no-of-publicipv6, description and disks attributes.

4.5.4.3 Create Image

The Create Image operation creates an image from an existing (stopped) server. This operation does not require a request body and sends the ve-name, image-name and subscription-id parameters in the URI. The response returns a text message describing the status of the operation.

4.5.4.4 Delete Image

The Delete Image operation deletes an existing server image. This operation does not require a request body and sends the image-name parameter in the URI. The response returns a text message describing the status of the operation.

The operations and parameters used by the Image Management component from PACI interface are resumed in Table 4.19.

Table 4.19: Image Management Table (Parameters *vs* Operations).

	Operations			
Parameters	**List Images**	**Get Image Information**	**Create Image**	**Delete Image**
Sent Parameters				
image-name		x	x	x
ve-name			x	
subscription-id			x	
Received Parameters				
text message			x	x
ve-image		x		
image-list	x			

4.6 Conclusions

There are significant differences regarding the type and number of interfaces, the level of customization, the organization of the groups of operations and the structure of the request/response messages provided by the four IaaS platform interface libraries.

OpenStack and PACI rely on RESTful interfaces, while OpenNebula and CloudStack use XML-RPC and Query (RESTlike) interfaces, respectively. Whereas the last three solutions present a single API, OpenStack, due to the modularity of the system, presents an API for each software component. The heterogeneity of the different interface types, in particular in the case on open source solutions, hinders the communication between different software solutions, forcing the creation of additional software layers (increasing the system's complexity) to prevent the user lock-in phenomenon.

The way the management components are organised as well as the set of specific system operations offered vary between IaaS platform. The provision of

additional operations by some IaaS platforms enable end-users with an increased level of customization. This is the case of the CloudStack API since it provides the end-user with a larger set of operations, including unique groups of operations, *e.g.*, the Virtual Private Cloud (VPC). On the other hand, the PACI API provides less management functionalities to the end-user, limiting the common management components of the four IaaS platform solutions to the Server and Image Management components. However, if only open source solutions are considered, there are additional common management components, namely, Virtual Networks, Volumes and VM Templates.

The organization of the operations differs between the four IaaS API. Some rely on one operation to provide multiple functionalities, while others define a specific operation for each functionality. For example, OpenNebula uses the same operation to perform different actions to a VM, while the other solutions use one operation for each action. The structure of the operations differs significantly between IaaS platform. OpenNebula adopts a simpler request/response message structure for the operations, exchanging a reduced set of parameters, while a CloudStack operation typically sends and retrieves a massive number of parameters.

Table 4.20 shows the comparison between the exposed management components.

Table 4.20: Management Components Comparison.

	OpenNebula	**OpenStack**	**CloudStack**	**PACI**
	Servers			
VM Management	x	x	x	x
	Images			
Image Management	x	x	x	x
	Storage			
Snapshots Management	x	x	x	-
Volumes Management	x	x	x	-
Objects Management	-	x	-	-
	Templates			
VM Templates Management	x	x	x	-
Application Templates Management	-	-	-	x
	Virtual Networks			
Networks Management	x	x	x	-
Firewall Management	-	-*	x	x
NAT Management	-	-*	x	-
VPN Management	-	-*	x	-
	Other Functionalities			
Group Management	x	x	x	-
Project Management	-	x	x	-
Zone Management	-	x	x	-
Auto Scale Management	-	x	x	-
Load Balancer	-	-*	x	x
VPC	-	-	x	-

In the following Chapter, the identified common management components between the four platforms will be analysed according to the number of similar operations and send/receive parameters in order to propose and develop an Interoperable Service.

Chapter 5

Interoperable Interface Proposal

This chapter proposes an interoperable interface solution based on the study and comparison of the different IaaS platforms performed in the previous chapter. The development of a new Interoperable Service with OpenNebula, OpenStack, CloudStack and PACI can be achieved by reusing existing cloud abstraction libraries or by designing and implementing a dedicated solution. These two approaches are analysed and a final selection is performed.

5.1 Dedicated Interoperable Service

A Dedicated Interoperable Service (DIS) with the four studied IaaS platforms can be created based on the identified common operations. This service can be developed using the Java programming language and should be organized in three fundamental layers: (*i*) Interface; (*ii*) Abstraction; and (*iii*) Interaction. Figure 5.1 illustrates a top level architecture that can be used in the development of such a service.

The knowledge representation of the overall process can be performed via a dedicated ontology, allowing the representation of the knowledge structure, creation of instances and inferencing. The Interface Layer knowledge models the exposed components features (operations, input and output parameters); the Abstraction Layer knowledge maps the Interface Layer knowledge with the Interaction Layer knowledge; and the Interaction layer knowledge represents the identified common components features (operations, input and output parameters) of the IaaS interface libraries.

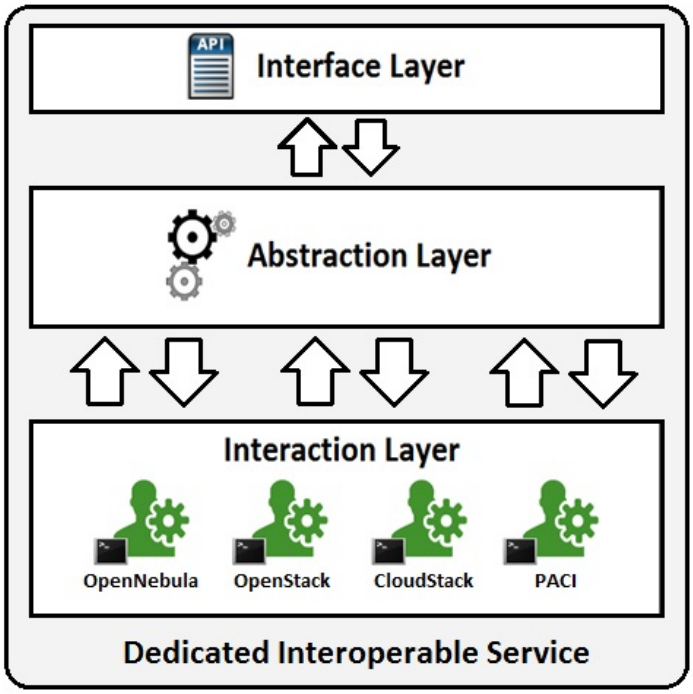


Figure 5.1: Dedicated Interoperable Service architecture.

5.1.1 Interaction Layer

The Interaction Layer is responsible for the interaction with the four IaaS platforms API, behaving like a software client that complies with each interface library specifications. This way, the communication between the DIS and the different IaaS platforms can include the different API technologies (Query, REST or XML-RPC), messaging formats (XML or JSON) and defined parameters. The requests are directly forwarded to the receiver IaaS platform API to be processed and the responses from the different IaaS platforms are delivered to the middle layer - Abstraction Layer - for further processing.

5.1.2 Abstraction Layer

The Abstraction Layer is responsible for processing the incoming messages and outgoing actions between the different cloud IaaS platforms and the exposed Dedicated Interoperable Service, creating the necessary abstraction for the end user. Since it is positioned between the Interaction Layer and the Interface Layer it is aware of the different IaaS platforms. This way, request messages are created mapping input parameters to the addressed IaaS API, retrieving any additional required information from a cache and translating the invoked actions to the specific IaaS API operations. The incoming messages are processed in a similar

way. The parameters from the response messages are analysed, cached for further utilization and selected to be delivered to the Interface Layer. The need for extra request operations is also evaluated, since the response messages provided by the IaaS platforms are significantly different and, in some cases, a single response to a request operation may not be sufficient.

5.1.3 Interface Layer

The Interface Layer interacts with the CLI and the Web dashboard through a RESTful API, exposing a common list of component operations regarding the different underlying IaaS platforms. The implementation of such an interoperable RESTful interface library can be supported by Apache CFX Framework [143], Apache Wink [144] or Apache Axis2 [145]. According to the previous chapter, the common components supported by the studied IaaS platform interface libraries are restricted to: (*i*) Virtual Machine Management; and (*ii*) Image Management. The following sections describe the specific list of operations exposed by such an DIS RESTful API.

5.1.3.1 Virtual Machine Management

The Interoperable Server Management component provides the following group of operations to manage the virtual machines: (*i*) List Virtual Machines; (*ii*) Get Virtual Machine Information; (*iii*) Create Virtual Machine; (*iv*) Delete Virtual Machine; (*v*) Start Virtual Machine; and (*vi*) Stop Virtual Machine.

List Virtual Machines:

The List Virtual Machines operation lists the available machines owned by an end user. This operation does not require a request body sending the **state** and **platform** parameters in the URI. The response returns the **servers_list** parameter.

- The **state** parameter can be used to filter the available virtual machines by its state. The available values are ACTIVE, STOPPED or DELETED;
- The **platform** parameter can filter the virtual machines available on a specific IaaS platform. The available values are opennebula, openstack, cloudstack and paci;
- The **servers_list** parameter is formed by a list of server elements containing the **name**, **id**, **state** and **platform** attributes of the virtual machines.

Get Virtual Machine Information:

The Get Virtual Machine Information operation retrieves detailed information for a specific virtual machine. This operation does not require a request body and sends the **id** parameter in the URI. The response returns the **server** parameter.

- The **server** parameter is composed by the **id**, **name**, **image_id**, **template_id**, **network**, **state**, **description** and **platform** attributes.

Create Virtual Machine:

The Create Virtual Machine operation generates a new virtual machine. This operation sends the **name**, **template_id**, **image_id** and **description** parameters. The response returns the **server** parameter.

Delete Virtual Machine:

The Delete Virtual Machine operation deletes an existing virtual machine. This operation does not require a request body and sends the **id** parameter in the URI. The response returns the **status** parameter.

Start Virtual Machine:

The Start Virtual Machine operation starts a paused virtual machine. This operation does not require a request body and sends the **id** parameter in the URI. The response returns the **status** parameter.

Stop Virtual Machine:

The Stop Virtual Machine operation pauses a running virtual machine. This operation does not require a request body and sends the **id** parameter in the URI. The response returns the **status** parameter.

5.1.3.2 Image Management

The Image Management component provides the following group of operations to manage the images from the different IaaS platforms: (*i*) List Images; (*ii*) Get Image Information; (*iii*) Create Image; and (*iv*) Delete Image.

List Images:

The List Images operation lists the images available to the end user. This operation does not require a request body and sends the **type** and **platform** parameters in the URI. The response returns the **image_list** parameter.

- The **type** parameter filter the returned list to public or user created images. The available values are public or user. If the parameter is omitted all the images available to the user are returned;

- The **platform** parameter selects the images by the IaaS platform;
- The **image_list** parameter is composed by a list of image elements with the **name**, **id**, **OS_name**, **OS_ARCH**, **description** and **platform** attributes.

Get Image Information:

The Get Image Information operation retrieves detailed information for a specific image. This operation does not require a request body and sends the **id** parameter in the URI. The response returns the **image** parameter.

- The **image** parameter is composed by the **name**, **id**, **OS_name**, **OS_ARCH**, **description** and **platform** attributes.

Create Image:

The Create Image operation generates a new image. This operation sends the **name**, **OS_option** and **description** parameters. The response returns the **image** parameter.

- The **OS_option** parameter specifies the available OS image option to be used.

Delete Image:

The Delete Image operation deletes an existing image. This operation does not require a request body and sends the **id** parameter in the URI. The response returns the **status** parameter.

5.2 Cloud Abstraction Interface Solutions

Interface abstraction libraries provide a collection of implementations of behaviour for the development of middle-ware systems, that in a multiple IaaS platform environment, abstract the peculiarities of a specific IaaS platform API offering a standard and unique API for the management of multiple IaaS clouds. Deltacloud [146], jClouds [147] and Libcloud [148] are examples of existing cloud abstraction solutions.

5.2.1 Deltacloud

Deltacloud is an open source top-level project from the Apache Software Foundation developed by Red Hat that aims to abstract differences between IaaS cloud platform interface libraries. Deltacloud is written in Ruby and, as illustrated

on Figure 5.2, it is divided into an API server - it can be either the Deltacloud RESTful API, the Distributed Management Task Force (DMTF) open standard Cloud Infrastructure Management Interface (CIMI) REST API [149] or the AWS EC2 API - and the drivers necessary for connecting with the cloud providers.

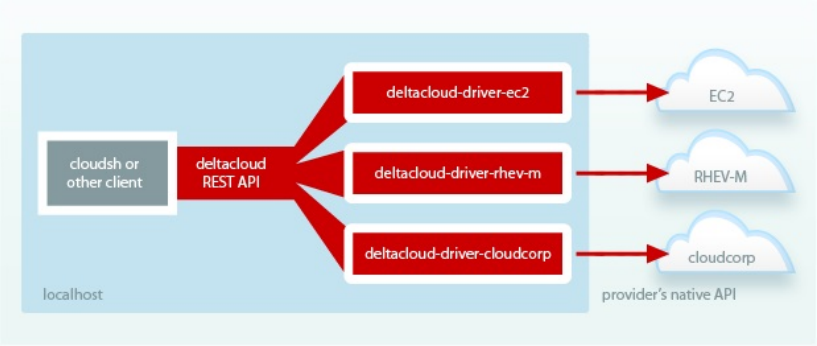


Figure 5.2: Deltacloud architecture [150].

Deltacloud also provides a Ruby client, a HTTP dashboard and a group of IaaS provider drivers (the entire list can be consulted at [151] that includes OpenNebula, OpenStack and Eucalyptus. It provides documentation for the development of new provider drivers (written in Ruby) that, in the case of this project, can be used to develop drivers for the Lunacloud IaaS platform PACI. Each driver exposes the list of supported Ruby collections that may differ from provider to provider. These collections describe the abstractions used by Deltacloud API and each collection represents an entity in the back-end provider cloud, *e.g.*, a running virtual server or a server image. The available collections are:

- Realms - Distinct organizational units within back-end clouds, *e.g.*, a data centre. A realm may, but does not necessarily represent, the geographical location of the accessed computing resources;
- Instances - Realized virtual servers running in given back-end clouds that are instantiated from server images;
- Images - Virtual machine images (or raw OS images) from which instances are created. Each image defines the root partition and initial storage for the instance OS;
- Instance states - Representations of instance life-cycle (start, pending, running, stopped, shutting_down, finished);
- Keys - Credentials used to access a running instance. Keys can take the form of key (*e.g.*, an RSA key) or of password (*i.e.*, with username and password attributes);

- Storage volume - A virtual storage device that can be attached to an instance and mounted by the OS;
- Storage snapshot - A copy of a storage volume at a specified time;
- Bucket - A container for data . The organizational unit of a generic key/-value based on a data-store (*e.g.*, Rackspace CloudFiles [152] or Amazon S3 [86]). Individual data items are exposed as a sub-collection under a bucket;
- Blob - A generic Binary Large Object (BLOB) data item that exists within a specified bucket (*e.g.*, an object in Amazon S3 [86] and Rackspace CloudFiles [152]);
- Address - Represents an IP address. Depending on the back-end cloud provider, the address can be public or private;
- Load Balancer - A load balancer allows a distribution of ingress network traffic received by a specified IP address to a number of instances;
- Firewalls - Sets of rules that govern the accessibility of a running instance over the public Internet;
- Metrics - Useful information about cloud resources, *e.g.*, CPU utilization or network throughput.

Using the described collections, the Deltacloud framework exposes compute, network and storage groups of operations. The server can respond to client requests in various formats. The appropriate response format is determined by HTTP content negotiation and the primary format is XML. The output is also available in JSON and HyperText Markup Language (HTML). Clients can also explicitly request a specific response format by including the format= request parameter, as shown below:

```
http://deltacloudserver.foo/api?format=xml
http://deltacloudserver.foo/api?format=json
```

Deltacloud uses basic HTTP authentication [126] to receive credentials from the client and passes them to the particular back-end cloud. The credentials always consist of a username and password and are never stored in the server. The exact login credentials, and the place to find them, depends on the back-end cloud platform that the Deltacloud server API is interacting with. A list of existing drivers together with the authentication details is available at [151]. There is also a Ruby cloud service library named Fog [153] that provides an accessible entry point and facilitates cross service compatibility to IaaS services like compute, DNS and storage. This library is included in Deltacloud.

Deltacloud interacts with multiple IaaS platforms by creating dedicated server instances for each IaaS platform driver, *i.e.*, provides a dedicated endpoint to interact with each IaaS platform, or reuses a single server instance to interact with all supported IaaS platform drivers.

5.2.2 jClouds

Apache jclouds is an open source library, developed by Apache Software Foundation, for cloud enabling and inter IaaS cloud platforms communication that allows the provisioning and control of cloud resources. jClouds API offers both portable abstractions and cloud-specific features that enable the management of buckets (BlobStore) and compute operations (ComputeService) with a compatible list of cloud providers and IaaS platforms, including OpenStack, CloudStack and Eucalyptus (since AWS is also supported) [154].

jClouds enables developers to add the support for new IaaS platform providers through the utilization of Maven archetypes. These archetypes supply the project set-up, the necessary dependencies and some code samples. The exposed Compute API provides a basic abstraction across Compute interface libraries and also integrates popular tools such as Ant [155] and Maven [156]. It can manage nodes as a set and address resources in any supported IaaS cloud platform without needing separate connections. The abstraction provided by the jClouds Compute API is based on the following vocabulary:

- ComputeService - a service level API endpoint, *e.g.*, EC2, that contains an inventory of hardware profiles, images and nodes, the ability to create and destroy nodes and resolve templates;
- Provider - runs a compute provisioning API, *e.g.*, vCloud, EC2. It may run multiple endpoints (*e.g.*, west, east, north) and is generally bound to a context and a unique identity;
- Image - a pre-configured operating system representing the base software for a new node. It is a part of the template used to create nodes;
- Hardware - a set of resource configurations that includes the input memory, CPU and disk required for the creation of a server;
- Location - an assignable physical or logical location inside a provider where nodes can be launched. Location can be scoped as Provider, Region or Zone. A location, which often refers to a virtual or physical data-centre, is a mandatory input for the creation of a server;
- TemplateOptions - options for creations of resources, including ports to open, scripts to run at bootstrap;

- Template - composite of Image, Hardware, Location and TemplateOptions for node creation. It allows for repeated creation of nodes;
- Node - a named instance of a Template sometimes called server or virtual machine. A node has, typically, login, IP and security metadata;
- Group - A name that aggregates nodes and related incidental resources such as keys, so that groups of nodes can be controlled as a unit. As many clouds do not support multiple groups, a group implies a primary grouping.

The complete information about jClouds API packages, classes and methods is available at [157]. There is a similar open source abstraction Java library that enables a Java developer to access functionality across a number of cloud providers through a single interface named Dasein Cloud API [158]. This API has a larger scope than jClouds, covering features such as networking and cloud PaaS functionalities.

5.2.3 Libcloud

Apache Libcloud is a standard Python library that abstracts the differences among multiple cloud provider interface libraries. It was originally created by Cloudkick (a start-up acquired by Rackspace) [159] and has grown into an independent free software project licensed under the Apache License (2.0).

The current version allows users to manage four different cloud resources:

- Cloud Servers - services such as Amazon EC2 [60];
- Cloud Storage - services such as Amazon S3 [86] and Rackspace CloudFiles [152];
- Load Balancer as a Service (LBaaS);
- DNS as a Service (DNSaaS).

The complete list of IaaS cloud providers is available at [160]. The group of compute operations are: (*i*) List; (*ii*) Reboot; (*iii*) Create; (*iv*) Destroy; (*v*) Images; (*vi*) Sizes; and (*vii*) Deploy. These compute operations are supported by Eucalyptus, OpenNebula, CloudStack and OpenStack. The compute terminology utilized to abstract the underlying differences between IaaS platforms is composed by the following terms:

- Node - represents a cloud or virtual server;

- NodeSize - represents a node hardware configuration, including the available RAM, bandwidth, CPU speed and disk size and, in most cases, exposing an hourly price (in dollars);
- NodeImage - represents an operating system image;
- NodeLocation - represents a server physical location;
- NodeState - represents a node state, which can be running, stopped, rebooting, terminated, pending and unknown.

The support for other IaaS platforms can be provides through the creation of new drivers (named third-party drivers by the Libcloud documentation). These drivers can be developed according to the LibCloud documentation [161]. To use a new driver it is necessary to register it using the provider.set_driver() function from the corresponding component. This function takes the **provider_name**, **path.to.the.module** and **DriverClass** arguments. An example of the registration of a third-party can be obtained from [162].

5.3 Conclusions

The IaaS cloud market is dynamically growing, but is supported by multiple non-interoperable IaaS solutions. Cloud libraries that are capable of abstracting IaaS API differences are fundamental in the current context of cloud computing. Consumers interact with different IaaS providers according to the existing available offerings and, thus, need to deal with the diversity and specificity of the supporting IaaS platforms. The challenge is to overcome existing lack of interoperability, vendor lock-in, terminology issues and distinct authentication methods, while exposing a common API to manage resources across different IaaS clouds. There are two possible approaches to address this problem: the creation of dedicated interoperable services or the adoption of existing abstraction solutions.

Cloud abstraction libraries or frameworks and dedicated interoperable solutions are commonly used to interact with different cloud infrastructures (with different deployment models) in several R&D cloud interoperability related projects. Some examples of such projects are:

- Aeolus [163] - a project composed by a suite of open-source tools that help the construction of custom clouds from both public and private resources. It uses Deltacloud as a core component for the enabling of cross and hybrid cloud functionality;

- mOSAIC [164] - a 7th Framework Programme (FP7) funded project with the goal to provide an open source API to support a multiple clouds platform. In terms of interfacing with exiting IaaS solutions, mOSAIC adopts a specific type of component called Driver. These Drivers reuse Libcloud, jclouds, DeltaCloud, PHP Simple Cloud [165] and Simple API for Grid Applications (SAGA) [166]. Saga also uses the Libcloud abstraction library to interface with the AWS EC2 API.

- Contrail [167] - an European project, partially funded by FP7, with the objective to design, implement, evaluate and promote an open source system for Cloud Federations. The Federated Cloud resources are highly heterogeneous in their hardware configuration and system-level organization, taking the form of physical machines running the XtreemOS operating system, an 6th Framework Programme (FP6) intellectual property project [168], virtual instances from external Clouds (using open standards such as OCCI and CIMI), virtual machines running XtreemOS or XtreemOS machines running virtualization software.

The goal of this project is to develop an open source solution for the integrated management of OpenNebula, OpenStack, CloudStack and PACI IaaS platforms resources. There are two possible approaches: the design and development of a dedicated interoperable Web service API or the reuse of existing abstraction solutions (libraries and frameworks). The first approach, which is based on the list of common IaaS platforms resource management operations, is specific and is not extensible. The latter, which is supported by existing an cloud abstraction libraries and frameworks, provides the required functionalities, is reusable and extensible.

Deltacloud, jClouds and Libcloud are among the most representative cloud IaaS abstraction solutions. Deltacloud, which provides by default three different service API (native RESTful Deltacloud API, CIMI and AWS EC2), is a framework that includes a Ruby client, a Web dashboard and a driver development environment to support the integration of further IaaS platforms. jClouds and Libcloud are standard programming libraries and, unlike Deltacloud, do not integrate additional development tools. In terms of IaaS platform support, Libcloud provides official integration with the studied open source IaaS platforms (OpenNebula, Eucalyptus, OpenStack and CloudStack), jClouds supports Eucalyptus, CloudStack and OpenStack while Deltacloud supports OpenNebula, Eucalyptus and OpenStack. None of these abstraction solutions provides support for PACI. The Table 5.1 presents a comparison between these open-source abstraction solutions.

Table 5.1: Abstraction solutions comparison.

	Open-source Abstraction Solutions		
Characteristics	***Deltacloud***	***jClouds***	***Libcloud***
Solution type	Framework	Library	Lybrary
Programming language	Ruby	Java	Python
Providers supported	17 cloud providers	30 cloud providers	38 cloud providers
Operations supported	Compute, Storage, Network	Compute, Storage	Compute, Storage, Network
IaaS platform interaction form	Drivers	Maven dependencies	Drivers
API	Native REST API, CIMI API, AWS API	-	-
Other features	Web dashboard, Ruby client	-	-

Although Libcloud provides official support for the analysed open source IaaS platforms, there are also third-party drivers that integrate CloudStack with Deltacloud. Thus, the Deltacloud abstraction framework will be adopted because it provides additional development tools and Web services (*e.g.*, the Ruby CLI, Web Dashboard), exposes broadly used interface libraries (CIMI API and AWS EC2 API) that will make the overall development project more complete and provide documentation for the development of Deltacloud drivers to integrate new IaaS platforms. The PACI IaaS platform support can be added through the development of a third-party driver.

Chapter 6

Project Development

This chapter presents the development environment, architecture, implementation, deployment modes and functionalities of an Interoperable Interface Service for the integrated management of the following IaaS platforms: PACI (LunaCloud IaaS proprietary platform) and OpenNebula, OpenStack and CloudStack open-source platforms.

6.1 Development Environment

The development environment is composed of the programming languages, back-end and front-end technologies as well as the development tools.

6.1.1 Languages

The programming languages adopted are presented in the following sections according to its application domain.

6.1.1.1 General Purpose Language

The Generic Purpose Language (GPL) adopted is the Ruby programming language [169]. This language, created by Yukihiro Matsumoto, is an object-oriented programming language with an easy-to-use interpreter, familiar syntax, complete object-oriented functionality and powerful class libraries [170]. Ruby's current stable version is 2.1.1, but, due to compatibility reasons, the Ruby version used was 1.9.1. The RubyGems [171] software (version 1.8.23) was installed together with Ruby. RubyGems allows the download, installation and usage of Ruby software packages. These software packages are called "gems" and contain Ruby applications or libraries.

6.1.1.2 Domain Specific Languages

The Domain Specific Language (DSL) used include XML and JSON for data transfer as well as HTML Abstraction Markup Language (HAML) and JavaScript (jQuery mobile library) for the user interface.

6.1.2 Back-end Technology

The Deltacloud framework constitutes the back-end technology of this project. It acts as an abstraction middleware for OpenNebula, OpenStack, CloudStack and PACI IaaS platforms. The framework, as Figure 6.1 illustrates, is composed of three main layers: (*i*) Drivers Layer; (*ii*) Core Layer; and (*iii*) User Interface Layer.

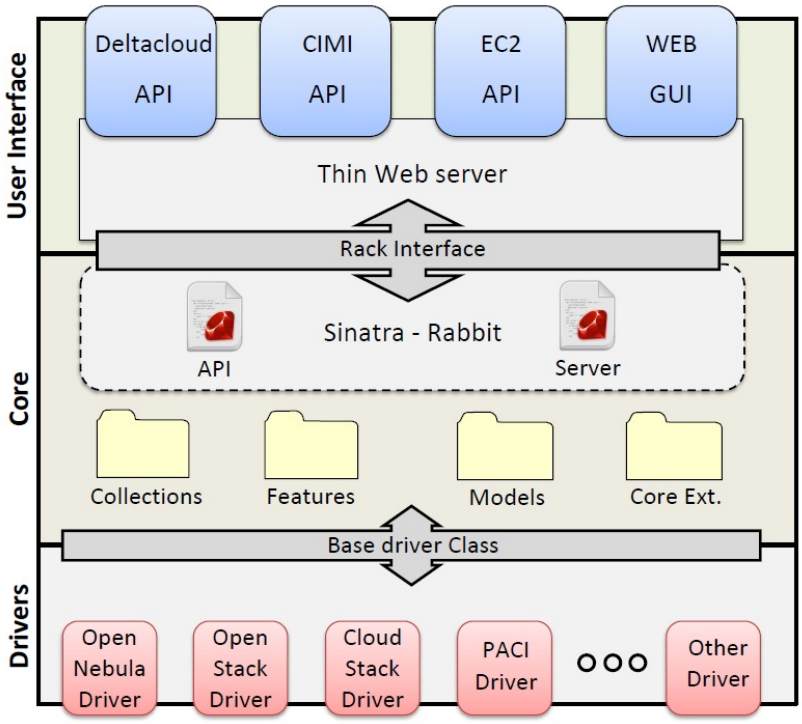


Figure 6.1: Deltacloud framework.

The Drivers Layer contains the individual drivers, written in Ruby, which interact with the specific back-end IaaS platforms and process the HTTP requests/responses. Each driver uses external Ruby gems or cloud clients to interact with the corresponding IaaS platform API. These libraries provide methods to perform API calls that comply with the IaaS API (protocol and operations). The drivers rely on methods and attributes inherited by the BaseDriver superclass to

create and access collections of objects used to expose and retrieve information from the IaaS platform via the User Interface Layer.

The middle layer - Core Layer - is composed of modules and classes that support the remaining two layers. It contains general Ruby libraries (*i.e.,* Core Ext.) as well as modules and classes that define the collection object type models, the collections and the features that each API currently exposes and that each driver can implement (represented as the Collections, Features and Models yellow folders in Figure 6.1). It also implements the Web service interfaces (*i.e.,* API file) and the Web dashboard (*i.e.,* Server file) as well as the HTTP body response format (XML, JSON or HTML) exposed by the top layer. Whereas the Web dashboard pages are defined in HAML, the remaining components are written in Ruby using Sinatra [172] and Sinatra's Rabbit extension [173].

The User Interface Layer is responsible for the exposure of the Deltacloud Web services. These Web services, which reuse the Sinatra library and rely on the Rack interface [174] for HTTP processing, are deployed in Thin [175], the Deltacloud Web application server. End-users access via Thin the available Deltacloud interfaces (API and Web dashboard).

6.1.2.1 Web Application Library

Sinatra is a Ruby-based DSL for building Web sites, Web services and Web applications [172]. It is a lightweight wrapper around Rack middleware that establishes a close relationship between service endpoints and HTTP operations, making it suitable for Web services. It emphasizes a minimalist approach to development, providing only what is essential to handle HTTP requests and deliver responses to clients. Sinatra, by itself, is not a Web framework, *i.e.*, it has no built-in ORM tools or pre-fabricated configuration files (like other Web frameworks such as Rails [176]) and it does not require the implementation of specific software models [177]. Nevertheless, the Deltacloud framework uses Sinatra version 1.4.4 and adopts a Model View Controller (MVC) approach.

6.1.2.2 Web Application Server

Thin is a HTTP Ruby Web server [175] included with Deltacloud and is composed by Mongrel's HTTP parser, the network I/O Event Machine library and the Rack Web server interface. The version used is 1.6.1.

6.1.3 Front-end Technologies

The Front-end support technologies adopted were the Web browser and the cURL command line tool [178]. The Web browser is used to interact with Deltacloud's Web dashboard and the cURL tool to access and invoke directly the Deltacloud's API. Alternatively, it is possible to use the Deltacloud Ruby Client library.

6.1.4 Development Tools

The tools used on the development environment consist of the Unix text editor (*e.g.*, VIM [179]), the Interactive Ruby Shell (IRB), the command line tool cURL [178] and the open-source packet analyser Wireshark [180]. While the Unix text editor and IRB are employed for programming and testing Ruby code, cURL and Wireshark are used for interacting with and testing the Web API as well as to measure the data load and time response of the HTTP/TCP packets.

6.2 Architecture

The Interoperable Service uses the Deltacloud abstraction framework as a middleware between cloud users and IaaS platforms, permitting the management of multiple IaaS platforms via a single service.

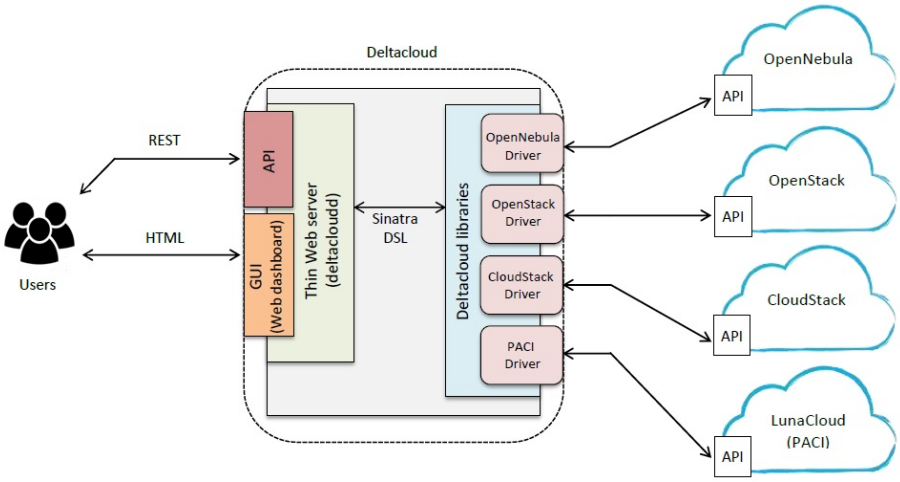


Figure 6.2: Interoperable Service architecture.

The architecture of this Interoperable Service is composed, essentially, by the back-end driver modules (OpenNebula, OpenStack, CloudStack and PACI driver), the software daemon deltacloudd and the GUI and API service interfaces. Figure 6.2 illustrates the architecture of the Interoperable Service.

The back-end driver modules, composed of the OpenNebula, OpenStack, CloudStack and PACI drivers, are integrated and developed to enable the abstraction and interaction with the respective back-end IaaS platforms. These drivers define, through method instantiation and implementation, the Deltacloud operations that the IaaS platform provides.

The software daemon deltacloudd is included in the deltacloud-core component and defines the service start-up and deployment using a set of established

options. Once instantiated, the deltacloudd starts the Thin Web server and loads the front-end Web services and defined driver modules.

The front-end Web services offer GUI and API service interfaces. The GUI service presents a simple Web dashboard containing the driver implemented Collections and operations. The API service has a RESTful implementation that uses driver defined collections and operations to expose the cloud resources from the IaaS platforms.

6.3 Interoperable Service API

The Interoperable Service uses the Deltacloud API [181] to manage the compute and storage resources of the integrated IaaS platforms. The management of these resources is made via the Deltacloud defined collections. These collections are formed by groups of operations and represent the abstracted IaaS platform entity. The existing collections are: (*i*) Realms; (*ii*) Hardware Profiles; (*iii*) Images; (*iv*) Instances; (*v*) Keys; (*vi*) Firewalls; (*vii*) Addresses; (*viii*) Load Balancers; (*ix*) Volumes; (*x*) Snapshots; and (*xi*) Blobs.

This section describes the collections and operations offered by Deltacloud.

6.3.1 Realms

The Realms collection implements a boundary containing the resources of a specific domain, *e.g.*, a data centre. This collection is formed by the following operations: (*i*) List Realms; and (*ii*) Show Realm Information.

6.3.1.1 List Realms

The List Realms operation lists all the available and implemented realms of the back-end IaaS platform. This operation accepts an optional **architecture** parameter in the request to filter the realms that support a specific architecture. The response returns a list of **realm** parameters containing the **href** and **id** attributes as well as the **name** and **state** sub-parameters for each realm.

6.3.1.2 Show Realm Information

The Show Realm Information operation details the information of a specific realm. This operation sends the **id** parameter of the realm in the request. The response returns a **realm** parameter containing the **href** and **id** attributes of the realm as well as the **name**, **state** and **limit** sub-parameters.

6.3.2 Hardware Profiles

The Hardware Profiles collection describes the attributes of a virtual machine, *e.g.*, vCPU, RAM and Disk. This collection is formed by the following operations: (*i*) List Hardware Profiles; and (*ii*) Show Hardware Profile Information.

6.3.2.1 List Hardware Profiles

The List Hardware Profiles operation lists all the available hardware profiles. This operation does not require request parameters. The response returns a list of **hardware_profile** parameters containing the **href** and **id** attributes as well as the **name** and **property** sub-parameters. The **property** sub-parameters specify the kind ('fixed', 'ranged', 'enumeration'), name ('cpu', 'memory', 'storage', 'architecture'), unit ('count', 'MB', 'GB', 'label') and value attributes for each hardware profile property.

6.3.2.2 Show Hardware Profile Information

The Show Hardware Profile Information details the information of a specific hardware profile. This operation sends the **id** parameter of the hardware profile in the request. The response returns a **hardware_profile** parameter containing the **href** and **id** attributes as well as the **name** and **property** sub-parameters.

6.3.3 Images

The Images collection describes image operations. The supported operations are: (*i*) List Images; (*ii*) Show Image Information; (*iii*) Create Image from Instance; and (*iv*) Delete Image.

6.3.3.1 List Images

The List Images operation lists all the available images. This operation sends optionally the **owner_id** and **architecture** parameters. The response returns a list of **image** parameters containing the **href** and **id** attributes as well as the **name**, **owner_id**, **description**, **architecture**, **state** and **actions** sub-parameters.

6.3.3.2 Show Image Information

The Show Image Information operation details the information of a specific image. This operation sends the **id** parameter of the specified image. The response returns an **image** parameter containing the **href** and **id** attributes as well as the **name**, **owner_id**, **description**, **architecture**, **state** and **actions** sub-parameters.

6.3.3.3 Create Image from Instance

The Create Image from Instance operation creates an image from a specific instance. This operation sends the **instance_id** parameter as well as the optional **name** and **description** parameters in the request. The response returns the same list of parameters as the Show Image Information operation.

6.3.3.4 Delete Image

The Delete Image operation deletes a specific image. This operation sends the **id** parameter of the image in the request. The response returns a HTTP 204 No Content header.

6.3.4 Instances

The Instances collection describes instance operations. The operations supported by this collection are: (*i*) List Instances; (*ii*) Show Instance Information; (*iii*) Instance Action; (*iv*) Create Instance; and (*v*) Delete Instance.

6.3.4.1 List Instances

The List Instances operation enumerates all the available instances. This operation does not require request parameters. The response returns a list of **instance** parameters containing the **href** and **id** attributes as well as the **name**, **owner_id**, **image**, **realm**, **state**, **hardware_profile**, **actions**, **launch_time**, **public_addresses**, **private_address**, **firewalls** and **authentication** sub-parameters.

6.3.4.2 Show Instance Information

The Show Instance Information operation details the information of a specific instance. This operation sends the **id** parameter of the instance in the request. The response returns an **instance** parameter containing the **href** and **id** attributes as well as the **name**, **owner_id**, **image**, **realm**, **state**, **hardware_profile**, **actions**, **launch_time**, **public_addresses**, **private_address**, **firewalls** and **authentication** sub-parameters.

6.3.4.3 Instance Action

The Instance Action operation performs a start, stop and reboot action on a specific instance. This operation sends the **id** and **action** parameters. The response returns the same list of parameters as the Show Instance Information operation.

6.3.4.4 Create Instance

The Create Instance operation creates a new instance. This operation sends the required **image_id** parameter as well as the optional **hwp_id**, **realm_id**, and **name** parameters. The response returns the same list of parameters as the Show Instance Information operation.

6.3.4.5 Delete Instance

The Delete Instance operation deletes the specified instance. This operation sends the instance **id** parameter in the request. The response returns the HTTP 204 No Content header.

6.3.5 Keys

The Keys collection capture the credentials required to access an Instance. This collection is formed by the following operations: (*i*) List Keys; (*ii*) Show Key Information; (*iii*) Create Key; and (*iv*) Delete Key.

6.3.5.1 List Keys

The List Keys operation lists all the available keys. This operation does not require request parameters. The response returns a list of **key** parameters with the **href** and **id** attributes as well as the **action**, **fingerprint** and **state** sub-parameters.

6.3.5.2 Show Key Information

The Show Key Information operation details the information of a specific key. This operation sends the **id** parameter in the request. The response returns the **key** parameter with the **href** and **id** attributes as well as the **action**, **fingerprint** and **state** sub-parameters.

6.3.5.3 Create Key

The Create Key operation generates a new key. This operation sends the **name** parameter in the request. The response returns the same parameter list as the Show Key Information operation.

6.3.5.4 Delete Key

The Delete Key operation deletes an existing key. This operation sends the **id** parameter in the request. The response returns the HTTP 204 No Content header.

6.3.6 Firewalls

The Firewalls collection manages sets of rules that govern the accessibility of a running instance over the public Internet. Currently, only the Amazon EC2 and Fujitsu GCP support this Deltacloud collection. This collection is formed by the following operations: (*i*) List Firewalls; (*ii*) Show Firewall Information; (*iii*) Create Firewall; (*iv*) Delete Firewall; (*v*) Create Firewall Rule; and (*vi*) Delete Firewall Rule.

6.3.6.1 List Firewalls

The List Firewalls operation lists all the available firewalls. This operation does not require request parameters. The response returns a list of **firewall** parameters with the **href** and **id** attributes as well as the **name**, **description**, **owner** and **rules** sub-parameters.

6.3.6.2 Show Firewall Information

The Show Firewall Information operation details the information of a specific firewall. This operation sends the **firewall_id** parameter in the request. The response returns the **firewall** parameter with the **href** and **id** attributes as well as the **name**, **description**, **owner** and **rules** sub-parameters.

6.3.6.3 Create Firewall

The Create Firewall operation creates a new firewall instance. This operation sends the **name** and **description** parameters in the request. The response returns the same list of parameters as the Show Firewall Information operation.

6.3.6.4 Delete Firewall

The Delete Firewall operation deletes a specific firewall. This operation sends the **id** parameter in the request. The response returns the HTTP 204 No Content header.

6.3.6.5 Create Firewall Rule

The Create Firewall Rule adds a new rule to an existing firewall. This operation sends the **firewall_id**, protocol, **port_from**, **port_to** and **sources** parameters. The response returns the same list of parameters as the Show Firewall Information operation.

6.3.6.6 Delete Firewall Rule

The Delete Firewall Rule deletes a rule from an existing firewall. This operation sends the **firewall_id** and **rule_id** parameters in the request. The response returns a HTTP 204 No Content header.

6.3.7 Addresses

The Addresses collection allows IP address management. This collection is formed by the following operations: (*i*) List Addresses; (*ii*) Shows Address Information; (*iii*) Create Address; (*iv*) Delete Address; (*v*) Associate Address; and (*vi*) Dissociate Address.

6.3.7.1 List Addresses

The List Addresses operation lists all the available IP addresses. This operation does not require request parameters. The response returns the list of **address** parameters with the **href** and **id** attributes as well as the **actions** and **ip** sub-parameters.

6.3.7.2 Shows Address Information

The Shows Address Information operation details the information of a specific address. This operation sends the **id** parameter in the request. The response returns the **address** parameter with the **href** and **id** attributes as well as the **actions** and **ip** sub-parameters.

6.3.7.3 Create Address

The Create Address operation creates a new IP address. This operation does not require request parameters. The response returns the same list of parameters as the Show Address Information operation.

6.3.7.4 Delete Address

The Delete Address operation deletes a specific IP address. This operation sends the **id** parameter in the request. The response returns the HTTP 204 No Content header.

6.3.7.5 Associate Address

The Associate Address operation associates an IP address to an instance. This operation sends the **id** and **instance_id** parameters in the request. The response returns the HTTP 202 Accepted header.

6.3.7.6 Dissociate Address

The Dissociate Address operation, dissociates an IP address from an instance. This operation sends the **id** parameter in the request. The response returns the HTTP 202 Accepted header.

6.3.8 Load Balancers

The Load Balancers collection allows the distribution of ingress network traffic, received by a specified IP address, to a number of running instances. This collection is formed by the following operations: (*i*) List Load Balancers; (*ii*) Show Load Balancer Information; (*iii*) Create Load Balancer; (*iv*) Delete Load Balancer; (*v*) Register Instance to Load Balancer; and (*vi*) Unregister Instance from Load Balancer.

6.3.8.1 List Load Balancers

The List Load Balancers operation returns the available load balancers. This operations does not require request parameters. The response returns the list of **load_balancer** parameters with a **href** and **id** attributes as well as the **actions**, **public_addresses**, **created_at**, **realm**, **listeners** and **instances** sub-parameters.

6.3.8.2 Show Load Balancer Information

The Show Load Balancer Information operation details the information of a specific load balancer. This operation sends the **id** parameter in the request. The response returns the **load_balancer** parameter with a **href** and **id** attributes as well as the **actions**, **public_addresses**, **created_at**, **realm**, **listeners** and **instances** sub-parameters.

6.3.8.3 Create Load Balancer

The Create Load Balancer operation creates a new load balancer. This operation sends the **name**, **realm_id**, **listener_protocol**, **listener_balance_port** and **listener_instance_port** parameters in the request. The response returns the same list of parameters as the Show Load Balancer Information operation.

6.3.8.4 Delete Load Balancer

The Delete Load Balancer operation deletes a specified load balancer with no associated instances. This operation sends the **id** parameter in the request. The response returns a HTTP 204 No Content header.

6.3.8.5 Register Instance to Load Balancer

The Register Instance to Load Balancer operation attaches an instance to the load balancer. This operation sends the **id** and **instance_id** parameters in the request. The response returns a HTTP 204 No Content header.

6.3.8.6 Unregister Instance from Load Balancer

The Unregister Instance from Load Balancer operation detaches an instance from a load balancer. This operation sends the **id** and **instance_id** parameters in the request. The response returns a HTTP 204 No Content header.

6.3.9 Volumes

The Volumes collection manages storage volumes. This collection is formed by the following operations: (*i*) List Volumes; (*ii*) Show Volume Information; (*iii*) Create Volume; (*iv*) Delete Volume; (*v*) Attach Volume; and (*vi*) Detach Volume.

6.3.9.1 List Volumes

The List Volumes operation lists all the available storage volumes. This operation does not require request parameters. The response returns a list of **storage_volume** parameters with the **href** and **id** attributes as well as the **created**, **capacity**, **state** and **actions** sub-parameters.

6.3.9.2 Show Volume Information

The Show Volume Information operation details the information of a specific storage volume. This operation sends the **id** parameter in the request. The response returns the **storage_volume** parameter with the **href** and **id** attributes as well as the **created**, **capacity**, **realm**, **state**, **mount** and **actions** sub-parameters.

6.3.9.3 Create Volume

The Create Volume operation creates a new storage volume. This operation sends the required **capacity** parameter as well as the optional **snapshot_id** and **realm_id** parameters. The response returns the same list of parameters as the Show Volume Information operation.

6.3.9.4 Delete Volume

The Delete Volume operation deletes a specific storage volume. This operation sends the **id** parameter in the request. The response returns the HTTP 204 No Content header.

6.3.9.5 Attach Volume

The Attach Volume operation attaches an existing storage volume to an instance. This operation sends the **id**, **instance_id** and **device** parameters. The response returns the same list of parameters as the Show Volume Information operation.

6.3.9.6 Detach Volume

The Detach Volume detaches a specific storage volume from an instance. This operation sends the **id** parameter in the request. The response returns the same list of parameters as the Show Volume Information operation.

6.3.10 Snapshots

The Snapshots collection creates and manages storage volumes backups. This collection is formed by the following operations: (*i*) List Snapshots; (*ii*) Show Snapshot Information; (*iii*) Create Snapshot; and (*iv*) Delete Snapshot.

6.3.10.1 List Snapshots

The List Snapshots operation returns all the available storage volume snapshots. This operation does not require request parameters. The response returns the list of **storage_snapshot** parameters with the **href** and **id** attributes as well as the **created** and **storage_volume** sub-parameters.

6.3.10.2 Show Snapshot Information

The Show Snapshot Information operation details the information of a specific storage volume snapshot. This operation sends the **id** parameter in the request. The response returns the **storage_snapshot** parameter with the **href** and **id** attributes as well as the **created** and **storage_volume** sub-parameters.

6.3.10.3 Create Snapshot

The Create Snapshot operation creates a new backup for a defined storage volume. This operation sends the **volume_id** parameter in the request. The response returns the same list of parameters as the Show Snapshot Information operation.

6.3.10.4 Delete Snapshot

The Delete Snapshot operation deletes a specific snapshot. This operation sends the **id** parameter in the request. The response returns the HTTP 204 No Content header.

6.3.11 Blobs

The Blob collection manages storage objects. This collection is formed by the following operations: (*i*) List Buckets; (*ii*) Show Bucket Information; (*iii*) Create Bucket; (*iv*) Delete Bucket; (*v*) Show Blob Information; (*vi*) Create Blob; (*vii*) Delete Blob; (*viii*) Show Blob Metadata; and (*ix*) Update Blob Metadata.

6.3.11.1 List Buckets

The List Buckets operation enumerates all the available buckets. This operation does not require request parameters. The response returns a list of **bucket** parameters with the **href** and **id** attributes as well as the **name** and **size** sub-parameters.

6.3.11.2 Show Bucket Information

The Show Bucket Information operation details the information of a specific bucket. This operation sends the **bucket_id** parameter in the request. The response returns the **bucket** parameter with the **href** and **id** attributes as well as the **name**, **size** and **blob** sub-parameters.

6.3.11.3 Create Bucket

The Create Bucket operation creates a new bucket. This operation sends the required **name** parameter as well as the optional **location** parameter. The response returns the same list of parameters as the Show Bucket Information operation.

6.3.11.4 Delete Bucket

The Delete Bucket operation deletes a specific bucket. This operation sends the **bucket_id** parameter in the request. The response returns a HTTP 204 No Content header.

6.3.11.5 Show Blob Information

The Show Blob Information details the information of a specific blob. This operation sends the **id** parameter in the request. The response returns the **blob** parameter with a **href** and **id** attributes as well as the **bucket**, **content_length**, **content_type**, **last_modified**, **user_metadata** and **content** sub-parameters.

6.3.11.6 Create Blob

The Create Blob operation creates or updates a blob object in a specific bucket. This operation sends the required **bucket_id**, **name**, **content_length**, **data** and **metadata** parameters as well as the optional **content_type** parameter in the request. The response returns the same list of parameters as the Show Blob Information operation.

6.3.11.7 Delete Blob

The Delete Blob operation deletes a blob object from a specific bucket. This operation sends the **bucket_id** and **blob_id** parameters in the request. The response returns the HTTP 204 No Content header

6.3.11.8 Show Blob Metadata

The Show Blob Metadata operation details the metadata information of a specific blob. This operation sends the **bucket_id** and **blob_id** parameters in the request. The response returns the HTTP 204 No Content header as well as the X-Deltacloud-Blobmeta headers.

6.3.11.9 Update Blob Metadata

The Update Blob Metadata operation overwrites the metadata information of a specific blob. This operation sends the **bucket_id** and **blob_id** parameters in the request. The response returns the HTTP 204 No Content header as well as the X-Deltacloud-Blobmeta headers.

6.4 Interoperable Service GUI

The Interoperable Service GUI consists of the Deltacloud Web dashboard. This dashboard presents the collections implemented by the loaded driver and exposes a simple graphic interface to manage the back-end IaaS platform resources. The GUI service implements also a mechanism to switch between back-end drivers to access other platforms collections. This latter mechanism only works if the authentication and back-end IaaS platforms endpoints are specified in the driver or in a YAML Ain't Markup Language (YAML) configuration file (as explained in the next section). The Figure 6.3 presents snapshots of the GUI service loaded with the PACI driver.

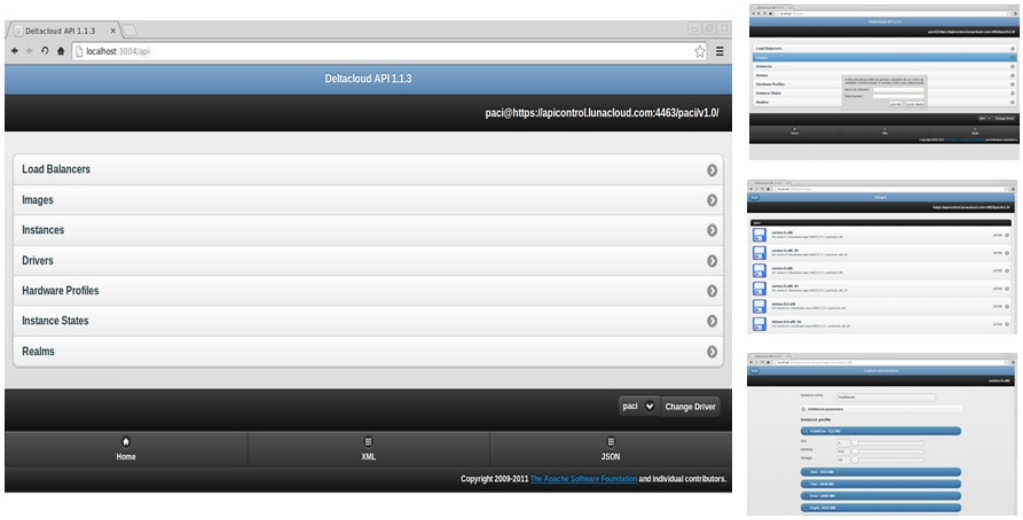


Figure 6.3: Deltacloud GUI service.

6.5 Deployment Configurations

The Deltacloud daemon deltacloudd is responsible for the start-up and deployment of the front-end interface services (the GUI and API services). The daemon can be launched via a Linux terminal using the following syntax:

```
$ deltacloudd -i [driver ID] -p [port number] -P [cloud provider endpoint URL]
```

Deltacloud only requires the specification of thedriver ID option on the daemon initiation. If the server port and cloud provider endpoint URL are not defined, Deltacloud will use the default configured port *i.e.*, 3001 and it will assume that the cloud provider endpoint URL is set within the used driver. The complete list of options that can be used with the Deltacloud daemon are defined in the Table 6.1.

Depending on the configuration of the Deltacloud daemon, two different deployments modes can be implemented: (*i*) Single Tenant Configuration; and (*ii*) Multiple Tenant Configuration. The following sections will address these two Deltacloud deployments.

6.5.1 Single Tenant Configuration

The Single Tenant Configuration deployment mode uses a single Deltacloud daemon that loads a pre-defined YAML file containing the credentials and the cloud provider URL endpoint for each driver module. This mode disables the authentication on the exposed services, forwarding all request to the back-end cloud with the specified credentials. This way, the user can switch between different

Table 6.1: Deltacloud daemon Operations.

Options	***Description***
-i –driver [Driver ID]	Specifies the driver to load.
-r –hostname [Hostname]	Binds to a host address. The default host is localhost.
-p –port [Port]	Defines the server port. The default is 3001.
-P –provider [Endpoint URL]	Sets the cloud provider endpoint URL. The default is set in the driver.
-f –frontends [Frontends]	Enable different front-end API (cimi, ec2, deltacloud).
-c –config [File]	Uses the credentials and the cloud provider endpoint URL from the HAML files.
-e –env [Env]	Specifies an environment variable.
-d –daemon	To run the daemon in the background.
-u –user [User]	Defines the user to run daemon as. Use with -d (default: “nobody”).
-g –group [Group]	Defines the group to run daemon as. Use with -d (default: “nobody”).
-b –pid [File]	File to store the server Process ID (PID). The default file is tmp/pids/thin.pid.
-l –drivers	Lists all the available drivers.
-L –log [File]	Saves the log requests to a file. Option disabled by default.
-s –ssl	Enables Secure Sockets Layer (SSL). It is disabled by default.
-k –ssl-key [Key]	Defines the SSL key file to use.
-C –ssl-cert [File]	Defines the SSL certificate file to use.
-t –timeout [Timeout]	Defines the time-out for a single request. The default is 60 seconds.
-V –verbose	Sets the verbose logging on.
-w –webrick	Forces the use of Ruby WEBRick server.
–logdir [Log dir]	Defines the directory for the log files.
-h –help	Shows deltacloudd options.

cloud provider drivers without the need to restart the server daemon (to specify the endpoint URL for each driver) or to include new HTTP headers in the API call. This approach is insecure since anybody with access to the server running the API and GUI services can use the defined credentials to manage the back-end cloud resources.

6.5.2 Multiple Tenant Configuration

The Multiple Tenant Configuration uses multiple server instances, containing each the GUI and API services defined by a specific back-end driver module. This mode is set-up by instantiating the deltacloudd daemon multiple times with different driver, port and cloud provider endpoint URL combinations. In this mode, the services initiated in the started servers are fixed and the users must switch between servers to manage different back-end IaaS platforms. Using this deployment mode, users need to authenticate with each server to access the corresponding services. Thus, this approach is safer and enables load balancing.

6.6 Reused Driver Modules

The tested driver modules were OpenNebula, OpenStack, CloudStack and PACI drivers. While the OpenNebula and OpenStack drivers are officially provided

by Deltacloud, the CloudStack driver is not officially included in the framework. Nevertheless, a third-party CloudStack driver is available. In the following sections, the OpenNebula, OpenStack and CloudStack driver are described.

6.6.1 OpenNebula Driver

The OpenNebula driver was created to support OpenNebula 3.X versions using the OCCI API [182] and, since the OCCI API specification (draft 0.8) continues to be supported by OpenNebula, is expected that the OpenNebula driver remains compatible with the most recent versions of OpenNebula.

The driver module is formed by three Ruby files: (*i*) cloud_client.rb; (*ii*) occi_client.rb; and (*iii*) opennebula_driver.rb. The cloud_client.rb implements a generic CloudClient library with methods and classes to authenticate, perform and monitor HTTP calls. The occi_client.rb file implements an OCCI Client class, composed of OCCI operations methods, to interact with the OpenNebula OCCI API. Whereas the opennebula_driver.rb file defines the OpennebulaDriver class, which resorts to the CloudClient library and OCCI Client class to implement the supported collections and operations compliant with Deltacloud's specifications.

The OpenNebula driver supports the Realms, Hardware Profiles, Images and Instances collections. The Realms collection is implemented with fixed values, since the OCCI API does not provide support for this functionality. The Hardware Profiles collection makes use of OpenNebula OCCI Instance Type collection to implement the defined compute resource templates (small, medium and large). The Images collection offers the following Image operations: (*i*) List Images; (*ii*) Show Image Information; and (*iii*) Delete Image. The Instances collection provides the following Instance operations: (*i*) List Instances; (*ii*) Show Instance Information; (*iii*) Create Instance; (*iv*) Start Instance; (*v*) Stop Instance; (*vi*) Reboot Instance; and (*vii*) Delete Instance.

6.6.2 OpenStack Driver

The OpenStack driver consists of a single file. The driver is implemented by the OpenstackDriver class, defined in the openstack_driver.rb Ruby file, that uses the openstack rubygem (library) version 1.1.2 [183] to interact with the Keystone, Nova, Cinder and Swift projects.

This driver implements the Realms, Hardware Profiles, Images, Instances, Keys, Volumes, Snapshots and Blobs collections. Since the concept of Realm is undefined in the OpenStack projects, only the information about user limits can be obtained from the Nova project API. This way, the Realms collection is implemented with the limits information obtained from the Nova project and fixed values for the name and identification definition of the realm. The Hardware

Profiles collection uses and exposes the OpenStack Nova "flavors" (VM resource templates). The Images collection exposes the following operations: (*i*) List Images; (*ii*) Show Image Information; (*iii*) Create Image from Instance; and (*iv*) Delete Image. The Instances collection offers the following operations: (*i*) List Instances; (*ii*) Show Instance Information; (*iii*) Create Instance; (*iv*) Reboot Instance and (*v*) Delete Instance. The Keys collection provides the following operations: (*i*) List Keys; (*ii*) Show Key Information; (*iii*) Create Key; and (*iv*) Delete Key. The Volumes and Snapshots collections interact with the Cinder project API. The Volumes collection implements the following operations: (*i*) List Volumes; (*ii*) Show Volume Information; (*iii*) Create Volume; (*iv*) Delete Volume; (*v*) Attach Volume; and (*vi*) Detach Volume. The Snapshots collection includes the following operations: (*i*) List Snapshots; (*ii*) Show Snapshot Information; (*iii*) Create Snapshot; and (*iv*) Delete Snapshot. The Blobs Collection interacts with the Swift project and exposes the following operations: (*i*) List Buckets; (*ii*) Show Bucket Information; (*iii*) Create Bucket; (*iv*) Delete Bucket; (*v*) Show Blob Information; (*vi*) Create Blob; (*vii*) Delete Blob; (*viii*) Obtain Blob Metadata; and (*ix*) Update Blob Metadata.

6.6.3 CloudStack Driver

The Deltacloud 1.1.3 version does not include a CloudStack driver. However, Chip Childers, a member of the Apache Software Foundation (ASF) involved in the Apache CloudStack project, started the development of a CloudStack driver for Deltacloud (available here [184]). This third-party driver was tested (Chapter 8 - Test, Debugging and Validation) without success.

6.7 Developed PACI Driver

There were no PACI driver implementations available for use with the Deltacloud framework. Therefore, to integrate Lunacloud's proprietary IaaS platform with Deltacloud, a dedicated PACI driver, compliant with the DeltaCloud framework, had to be designed and developed. The PACI IaaS platform operations were not defined in the existing rubygem libraries, meaning that a PACI client had also to be created to enable the communication with the back-end LunaCloud PACI API. This way, by relying on the available driver specification information [185] provided by Deltacloud as well as by analysing the existing drivers code, a new PACI driver was developed.

Following the OpenNebula driver module approach, the PACI driver implementation consists of three Ruby files: (*i*) cloud_client.rb; (*ii*) paci_client.rb; and (*iii*) paci_driver.rb. The cloud_client.rb file, which was inspired on the OpenNebula driver module to exploit the already defined methods and classes, performs and monitors the HTTP calls implemented by theCloudClient library.

The paci_client.rb Ruby file was created to implement the PACI client, which consists of a Client class with defined methods to interact with the Lunacloud PACI API. The paci_driver.rb Ruby file contains the PaciDriver class, which is responsible for the implementation of the PACI driver. This class inherits the features and properties (methods and attributes) from the BaseDriver superclass, defined in the base_driver.rb file for the definition of Deltacloud collections and operations. Figure 6.4 represents a Unified Model Language (UML) class diagram of the developed PACI driver.

Figure 6.4: UML classes diagram of the PACI driver.

The development of the PACI driver module was conducted in two main phases. The fist phase consisted on the development of the PACI client, while the second phase focussed on the development and implementation of the PACI driver. The PACI client was defined with methods to describe and implement the PACI API HTTP requests, enabling the support for the following PACI API operations: (*i*) List Servers; (*ii*) Start/Stop Server; (*iii*) Create Server; (*iv*) Create Server From Image; (*v*) Obtain Server Information; (*vi*) Delete Server; (*vii*) List Images; (*viii*) Get Image Information; (*ix*) Create Image From Server; (*x*) Delete Image; (*xi*) List Installed OS Templates; (*xii*) List Load Balancers; (*xiii*) Get Load Balancer Information; (*xiv*) Create Load Balancer; (*xv*) Delete Load Balancer; (*xvi*) Attach Server to Load Balancer; and (*xvii*) Detach Server From Load Balancer.

The methods that implement the described operations comply with the PACI RESTful API documentation [142]. The Code Snippet 6.1 presents the PACI client Create Server operation, which is defined by the create_instance method:

Code Snippet 6.1: PACI client create_instance method

```
1   def create_instance(xml)
2       post('/ve', xml)
3   end
```

The required parameters are passed through the method arguments. The method invokes the post private method responsible for processing the HTTP POST verb. The PACI client, in the Client class, has private methods that implement the processing of the GET, POST, DELETE and PUT HTTP verbs. The Code Snippet 6.2 presents the description of the post method:

Code Snippet 6.2: PACI client post method

```
1   def post(path, xml=nil)
2       url = URI.parse(@endpoint+path)
3       req = Net::HTTP::Post.new(url.path)
4       if !xml.nil?
5           req.content_type= 'application/xml'
6           req.body=xml.to_s
7       end
8       do_request(url, req)
9   end
```

This method has two input arguments: the path argument containing the path for the operation and the xml argument with the description of the HTTP request body (since the PACI API uses the XML notation for the HTTP message body). The url object from the URI::HTTP class is created, using the URI.parse() method, by concatenating the endpoint and path strings. Then, the NET::HTTP::POST superclass is instantiated to define the HTTP request (req) object. If the operation uses a HTTP body, the Multi-purpose Internet Mail Extension (MIME) type is specified using the Content-Type: application/xml HTTP header field and the xml argument string with the XML description is included to the HTTP request body. Once the request is created, the do_request method performs the request. The do_request private method code is presented in the Code Snippet 6.3:

The do_request invokes the http_start method from the CloudClient library in a Ruby block, passing the HTTP request information. The http_start method establishes the HTTP connection, sending the HTTP request and collecting the HTTP server response. The HTTP connection is monitored using the CloudClient error class. The complete description of the CloudClient library and the

Code Snippet 6.3: PACI client do_request method

```
1     def do_request(url, req)
2       req.basic_auth @auth[0], @auth[1]
3
4       res = CloudClient::http_start(url, @timeout) do |http|
5     http.request(req)
6       end
7
8       if CloudClient::is_error?(res)
9           return res
10      else
11          return res.body
12      end
13    end
```

Client class from the PACI client are available in the Appendix A and Appendix B respectively.

The PACI driver implements the Realms, Hardware Profiles, Images, Instances and Load Balancers collections and exposes the following operations: (*i*) List Realms; (*ii*) Show Realm Information; (*iii*) List Hardware Profiles; (*iv*) Show Hardware Profile Information; (*v*) List Instances; (*vi*) Show Instance Information; (*vii*) Create Instance; (*viii*) Start Instance; (*ix*) Stop Instance; (*x*) Delete Instance; (*xi*) List Images; (*xii*) Show Image Information; (*xiii*) List Load Balancers; (*xiv*) Show Load Balancer Information; (*xv*) Create Load Balancer; (*xvi*) Delete Load Balancer; (*xvii*) Attach Instance to Load Balancer; and (*xviii*) Detach Instance From Load Balancer. The operations described are implemented using methods inherited from the BaseDriver superclass.

The Realms collection specifies Lunacloud existing domains (EU Central and EU West). Since the PACI API does not provide such operations, the values that define the Realms collection are static. Thus, PACI does not use resource templates. The virtual machine resources are defined in the XML description request body of the Create Server operation. However, Deltacloud resorts to resource templates for the creation of instances. This way, the PACI driver uses the define_hardware_profile methods from the Hardware Profiles collection to implement resource templates. The code snippet 6.4 presents the define_hardware_profile method used to create a custom hardware profile:

Code Snippet 6.4: PACI driver define_hardware_profile method

```
1     define_hardware_profile('Custom') do
2         cpu                    (1 .. 8)
3         memory                 (512 .. 192*512)
4         storage                (10 .. 2000)
5     end
```

This hardware profile corresponds to an unspecified resource template that

allows the usage of custom VM CPU, RAM and Storage resources with predefined size ranges. The range was implemented according to the Lunacloud limits specifications for the CPU, RAM and Storage resources [186]. Other hardware profiles were created, using the define_hardware_profile method, based on LunaCloud's resource configurations (*e.g.*, PointFive, One, Two, Four, Eight and OneSix).

The Images collection is defined to use PACI OS templates. These OS templates are responsible for the configuration and installation of the virtual machine OS at the time of the VM creation. The OS templates are implemented and managed by the cloud provider (*e.g.*, LunaCloud) and are available to multiple users on the data centre. Thus, since the OS templates cannot be created or destroyed by normal users, only read operations can be implemented from the Deltacloud Images collection (*i.e.*, the List Images and Show Image Information operations).

The Instances collection implements all instance operations with the exception of the reboot operation since is not included in the PACI platform. The only requirement for the creation of an instance using the Deltacloud is the instantiation of an OS Image. The hardware profile as well as the instance name, realm and defined password are optional. However, these instantiations are insufficient to create a new VM in the PACI IaaS platform. The PACI API requires that an additional set of parameters are specified in XML format. Therefore, to enable the creation of virtual machines in the LunaCloud PACI platform using Deltacloud, the following XML template (Code Snippet 6.5) was defined as a global VE_TEMPLATE string in the PACI driver class:

Code Snippet 6.5: PACI driver XML server template

```
1  VE_TEMPLATE = \%q{
2      <ve>
3      <name><%=ve_name%></name>
4      <cpu number="<%=opts[:hwp_cpu]%>" power="1600"/>
5      <ram-size><%=opts[:hwp_memory].to_i%></ram-size>
6      <bandwidth>10240</bandwidth>
7      <no-of-public-ip>1</no-of-public-ip>
8      <no-of-public-ipv6>0</no-of-public-ipv6>
9      <ve-disk local="true" size="<%=opts[:hwp_storage]%>"/>
10     <platform>
11      <template-info name="<%=image_id%>"/>
12      <os-info technology="<%=tech%>" type="<%=type%>"/>
13     </platform>
14     <backup-schedule name="weekly"/>
15     <% if opts[:password] %>
16     <admin login="root" password="<%=opts[:password]%>"/>
17     <% else %>
18     <admin login="root" />
19     <% end %>
20 </ve>
21 }
```

This template uses a XML structure, compliant with the PACI API documentation, that contains embedded Ruby code. This approach allows the imple-

mentation of Deltacloud defined parameters (*e.g.*, image_id and opts[] variables) as well as the inclusion of other parameters and attributes that must be specified (*e.g.*, the cpu power attribute or the bandwidth parameter). The no-of-public-ip, no-of-public-ipv6, bandwidth and backup-schedule parameters as well as the admin login attribute were implemented with default values since Deltacloud does not support these features.

The Load Balancer collection implements all Load Balancer operations. However, since the Deltacloud Load Balancer operations require more parameters than those required by the PACI platform, the extra parameters are initialised with mock values.

The PACI driver operations were defined by overriding the BaseDriver super class corresponding methods. For example, the Create Instance operation, overrides the create_instance(...) BaseDriver method. The implemented create_instance(credentials, image_id, opts={}) method is presented in the Code Snippet 6.6:

Code Snippet 6.6: PACI driver create_instance method

```
1   def create_instance(credentials, image_id, opts={})
2
3       paci_client = new_client(credentials)
4
5       if opts[:name] && opts[:name].length>0
6           ve_name= opts[:name]
7       else
8           time=Time.now.to_s
9           time=time.split('+').first
10          time=time.gsub(/\D/,'')
11          ve_name= "Server-"+time
12      end
13
14      img_xml = treat_response( paci_client.get_os_template(image_id) )
15      buffer = REXML::Document.new(img_xml.to_s).root.attributes
16      tech = buffer['technology']
17      type = buffer['osType']
18      req_xml = ERB.new(VE_TEMPLATE).result(binding)
19      treat_response( paci_client.create_instance(req_xml) )
20
21      instance(credentials, id: ve_name)
22  end
```

This method obtains the parameters required to fill the PACI XML template, requests the creation of an instance to the back-end PACI platform and shows the created instance in front-end Deltacloud interfaces. Like all the other defined methods that implement operations, this method instantiates the new_client private method to create a PaciClient object in order to use the PACI client methods. Then, the create_instance method requests the PACI platform for the information regarding a specific OS template, using treat_response private method for monitoring, and invokes the PACI client get_os_template(image_id) method to perform the HTTP call. The returned response is parsed to obtain

the required attributes, using the Ruby Electric XML (REXML) superclass [187] and the Embedded Ruby (ERB) superclass [188], and the XML request body is created. Once again, the treat_response private method is used, with the create_instance(req_xml) PACI client method, to create a new VM. The response returned by the PACI API is discarded since it contains only information about the created password (Lunacloud sends this information to the user e-mail). Finally, in order to show the new instance, it is necessary to invoke the instance(credentials, id: ve_name) method, passing the identification of the created instance. This latter method was defined (overriden) to implement the Deltacloud Show Instance Information operation.

The information exposed by Deltacloud in the front-end API and GUI interfaces is filtered from the PACI API returned responses. This process is implemented by private methods that receive the returned XML body and map its parameters and attributes. The Code Snippet 6.7 presents the convert_image method:

Code Snippet 6.7: PACI driver convert_image method

```
1   def convert_image(xml, credentials)
2       buffer = REXML::Document.new(xml.to_s).root
3       if buffer.attributes['active']=="false"
4           default_state = "DISABLED"
5       else
6           default_state = "ACTIVE"
7       end
8
9   Image.new({
10      :id=>buffer.attributes['name'],
11      :name=>buffer.attributes['name'],
12      :description=>"OS: "+buffer.elements[2].attributes['value']+",
            Virtualization type (VM/CT): "+buffer.attributes['technology'],
13      :owner_id=>"LunaCloud",
14      :state=>default_state,
15      :architecture=>buffer.elements[1].attributes['value'],
16      :hardware_profiles=>hardware_profiles(nil)
17      })
18  end
19
```

The convert_image method, which retrieves image-related information, uses the REXML superclass to parse the XML from the xml argument and to obtain its parameters and attributes. This information is, then, included in the Deltacloud Image objects. The Image objects are represented in hash tables containing the id, name, description, owner_id, state, architecture and hardware_profiles keys. The value defined in these keys is the information exposed by Deltacloud interfaces.

Another important method defined in the PACI driver is the define_instance_states method that implements a state machine for the virtual machine life-cycle of a PACI VM. This method is presented in the Code Snippet 6.8:

Code Snippet 6.8: PACI driver define_instance_states method

```
define_instance_states do
    start.to(:pending)          .on( :create )
    pending.to(:stopped)        .automatically
    stopped.to(:pending)        .on( :destroy )
    pending.to(:finish)         .automatically
    stopped.to(:running)        .on( :start )
    running.to(:stopping)       .on( :stop )
    stopping.to(:stopped)       .automatically
    running.to(:stopping)       .on( :destroy )
    stopping.to(:finish)        .automatically
end
```

When a VM is created, it is by default in the stopped state. Its state changes to pending when a destroy VM instruction is received, running when a start VM instruction is received and stopping when a stop VM instruction is received. The state changes from pending to stopped, pending to finish, stopping to stopped and stopping to finish automatically. The names for the VM states used by PACI are different from the ones defined by Deltacloud and needed to be changed to comply with the Deltacloud defined states. Thus, the VE_STATES hash was implemented - see Code Snippet 6.9.

Code Snippet 6.9: PACI driver VE_STATES hash

```
VE_STATES = {
    "CREATE"                => "START",
    "CREATION_IN_PROGRESS"  => "PENDING"
    "CREATED"               => "STOPPED",
    "START_IN_PROGRESS"     => "RUNNING",
    "STARTED"               => "RUNNING",
    "STOP_IN_PROGRESS"      => "STOPPING",
    "STOPPED"               => "STOPPED",
    "DELETE_IN_PROGRESS"    => "STOPPING",
    "DELETED"               => "FINISHED"
}
```

Using the state information of an instance and the defined state machine, Deltacloud can determine which instance actions can be performed. If the instance is STOPPED, the START action is “recommended” and, since the define_instance_states method is defined to change from start to stop and stop to destroy, an instance cannot be destroyed when its state is STARTED. The PaciDriver class containing the description of the PACI driver code is provided in the Appendix C.

Finally, so that the PACI driver can be loaded by the Deltacloud daemon and recognized by the front-end services, a paci.yaml YAML configuration file was defined containing the following configuration:

```
---
:paci:
    :name PACI
```

6.8 Conclusions

The development environment provides support for the system back-end and front-end. The back-end is provided by Deltacloud and the front-end consists of the standard Web browser or the cURL CLI. The development support language is Ruby and the application layer protocol is HTTP. The dynamic Web pages templates are defined in HAML and the data output formats supported by API include XML, JSON and HTML.

The development of the Interoperable Service consisted on the integration of the Deltacloud abstraction framework with the proprietary IaaS platform used by LunaCloud and the most popular open-source IaaS platforms: OpenNebula, OpenStack and CloudStack. The utilization of Deltacloud permitted the adoption of a single management syntax, using its RESTful API (or the CIMI and EC2 API) and provided a simple Web dashboard to interact with different back-end IaaS platforms as well as a secure and managed access to the exposed services with the possibility of a multiple tenant deployment.

The creation of the PACI driver followed the documentation provided by Deltacloud, which proved to be insufficient and outdated, and the code from other driver implementations, mainly the OpenNebula driver and the EC2 driver code. Along with the PACI driver, it was also necessary to develop a PACI client containing the description of PACI API operations. From the analysis conducted between the offered Deltacloud collections and the group of operations exposed by the PACI API, it was decided to develop a driver with the Realms, Hardware Profiles, Images, Instances and Load Balancers collections. Since the Deltacloud API required parameters did not always match with the required PACI API parameters, *e.g.*, when creating a new VM, it was necessary to perform multiple calls to the back-end API as well as to define default and mock values for these parameters. This approach may lead to lack of front-end customization (using the Deltacloud interfaces) and performance deterioration. Nevertheless, the developed PACI driver is completely functional and can execute all the implemented operations.

Chapter 7

Test, Debugging and Validation

This chapter describes the test bed set-up, the tests performed to evaluate the developed Interoperable Service solution and the presentation, interpretation and discussion of the obtained results.

7.1 Test Bed

To conduct the functional tests of the developed Interoperable Service, the OpenNebula, OpenStack and CloudStack IaaS platforms as well as the Deltacloud framework were installed and configured. For this purpose, a test bed containing the following components was assembled:

- OpenNebula cloud node;
- OpenStack cloud node;
- CloudStack cloud node;
- Interoperable Service node;
- Test network with access to the Internet.

To deploy the OpenNebula, OpenStack and CloudStack nodes two desktop computers with a Intel Pentium G3220 CPU (virtualization enabled), 4096 MB RAM, 500 GB of disk storage capacity and 1 NIC were used. Since there were no more platforms with enabled virtualization capabilities available, the hard disk drive of one PC was partitioned to accommodate both the OpenStack and CloudStack IaaS platforms. To ensure identical deployment conditions for all

IaaS platforms, the Linux CentOS 6.5 OS was installed in the three systems. The PACI IaaS platform node is provided and supported by Lunacloud.

The node containing the Interoperable Service was deployed in a laptop with the Linux Mint 16 OS with the Deltacloud framework (containing the OpenNebula and OpenStack driver modules) and the PACI driver module. Since the laptop was also used as the client platform to access the other set-up nodes, the Deltacloud server was configured to run in the loopback network. All the nodes of the test bed were connected through a test network (192.168.1.0/24) using a router that isolates the components from the laboratory network and provides Internet access to the PACI IaaS platform. All nodes were connected to the router via Ethernet crossover cables and all computer clocks where configured to synchronise with Lunacloud's Network Time Protocol (NTP) server (ntp.lunacloud.com). The Figure 7.1 illustrates the assembled test bed.

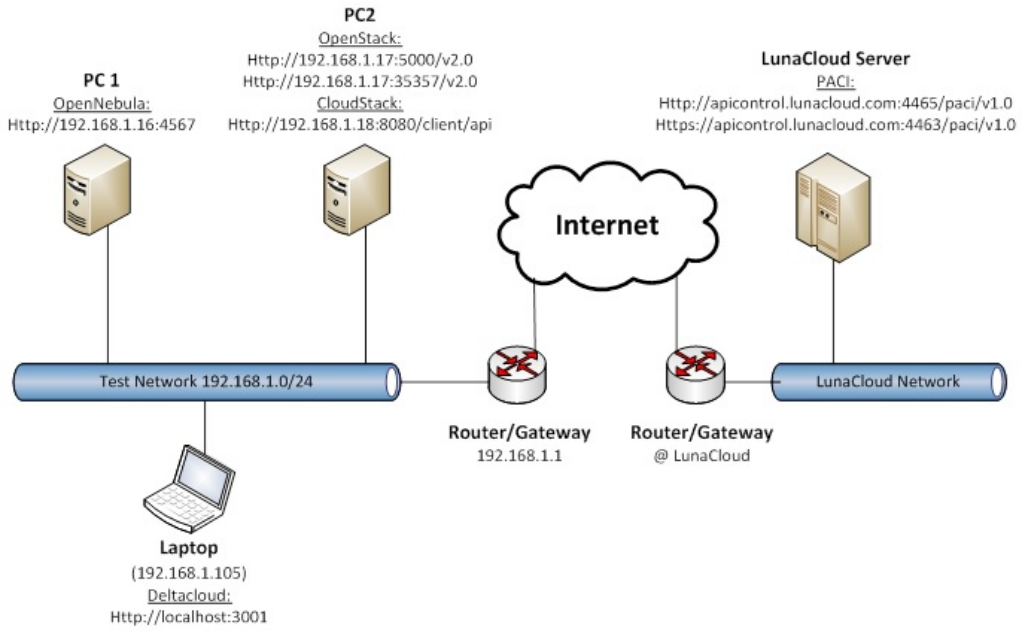


Figure 7.1: Test bed platform.

The following sections describe the test bed installation procedures, including the Interoperable service and the OpenNebula, OpenStack and CloudStack IaaS platforms in the referred nodes.

7.1.1 Interoperable Service Installation

The installation of the Interoperable service is divided in three steps: (*i*) Installation of Deltacloud framework; (*ii*) Inclusion of the CloudStack driver module; and (*iii*) Inclusion of the PACI driver module.

7.1.1.1 Deltacloud Installation

The installation of Deltacloud consists of the installation of the required dependencies to run the Deltacloud server and the installation of the Deltacloud service.

Deltacloud requires the following dependencies:

- Ruby, Ruby-devel and RubyGems;
- GCC-C++, Libxml2, Libxml2-devel, Libxslt, Libxslt-devel, sqlite and sqlite-devel libraries;
- Thin, sinatra, rack-accept, rest-client, sinatra-content-for, and nokogiri gems.

The Ruby version 1.9.1, Ruby-devel and RubyGems can be installed with the following command:

```
# apt-get install ruby1.9.1-full
```

Next, the required libraries to build RubyGems with C extensions and for the CIMI persistence layer are installed:

```
# apt-get install g++
# apt-get install libxml++2.6-dev libxml2-dev
# apt-get install libxslt1.1 libxslt-dev
# apt-get install sqlite libsqlite3-dev
```

Then, the required gems are set using RubyGems:

```
# gem install thin sinatra rack-accept rest-client \
sinatra-content-for nokogiri
```

and a symbolic link is created for the service launcher:

```
# ln -s /deltacloud-core-1.1.3/bin/deltacloudd \
/usr/bin/deltacloudd
```

Finally, with all dependencies resolved, the Deltacloud is installed using RubyGems:

```
# gem install deltacloud-core
```

7.1.1.2 CloudStack Driver Module Inclusion

To include the third-party CloudStack driver module in the Deltacloud framework it was necessary to move the obtained cloudstack_driver.rb and cloudstack_driver_cimi_methods.rb files to a created cloudstack folder inside the Deltacloud drivers folder:

```
# mkdir /deltacloud-core-1.1.3/lib/deltacloud/drivers/cloudstack
# mv cloudstack_driver.rb cloudstack_driver_cimi_methods.rb \
/deltacloud-core-1.1.3/lib/deltacloud/drivers/cloudstack/
```

Then, it was necessary the creation of a cloudstack.yaml file containing the following information:

```
---
:cloudstack:
   :name Cloudstack
```

Finally, the cloudstack.yaml file needed to be moved to the drivers configuration folder:

```
# mv cloudstack.yaml /deltacloud-core-1.1.3/config/drivers/
```

7.1.1.3 PACI Driver Module Inclusion

To include the developed PACI driver module in the Deltacloud framework it was necessary to move the cloud_client.rb, paci_client.rb and paci_driver.rb files to a created paci folder inside the Deltacloud drivers folder:

```
# mkdir deltacloud-core-1.1.3/lib/deltacloud/drivers/paci
# mv cloud_client.rb  paci_client.rb  paci_driver.rb \
/deltacloud-core-1.1.3/lib/deltacloud/drivers/paci/
```

Then, the paci.yaml configuration file needed to be moved to the Deltacloud drivers configuration folder:

```
# mv paci.yaml /deltacloud-core-1.1.3/config/drivers/
```

7.1.2 OpenNebula Installation

The OpenNebula installation is divided in four main procedures: (*i*) OS Configuration; (*ii*) OpenNebula Services Installation; (*iii*) KVM Hypervisor Installation and (*iv*) Virtual Resources Allocation.

7.1.2.1 OS Configuration

The OS Configuration procedure prepares the host OS for the OpenNebula installation. This procedure configures the OS network service for the OpenNebula usage and sets the OpenNebula repository to download and install automatically the required software packages.

The OS NIC eth0 has to be configured as a network bridge (*e.g*, br0) that will act as the public interface for the KVM virtual machines.

The eth0 interface can be configured by replacing `/etc/sysconfig/network-scripts/ifcfg-eth0` file with:

```
DEVICE=eth0
BOOTPROTO=none
NM_CONTROLLED=no
ONBOOT=yes
TYPE=Ethernet
BRIDGE=br0
```

Then, a new ifcfg-br0 file needs be added to the `/etc/sysconfig/network-scripts/network-scripts/` folder with the following configuration:

```
# cat << EOT > /etc/sysconfig/network-scripts/network-scripts/ifcfg-br0
DEVICE=br0
TYPE=Bridge
IPADDR=192.168.1.16
NETMASK=255.255.255.0
ONBOOT=yes
BOOTPROTO=static
NM_CONTROLLED=no
EOT
```

After the network bridge is configured, the OS network service is restarted for the changes to take effect:

```
# service network restart
```

Once the OS network service configured, the OpenNebula repository can be created to get and install the necessary software packages for the installation of OpenNebula:

```
# cat << EOT > /etc/yum.repos.d/opennebula.repo
[opennebula]
name=opennebula
baseurl=http://downloads.opennebula.org/repo/CentOS/6/stable/\$basearch
enabled=1
gpgcheck=0
EOT
```

7.1.2.2 OpenNebula Services Installation

The OpenNebula Services packages consist of the opennebula-server, opennebula-sunstone and opennebula-ruby. The opennebula-server package contains the main OpenNebula daemon, scheduler and other core functions. The opennebula-sunstone package contains the Sunstone Web dashboard as well as the EC2 and OCCI interface servers. The opennebula-ruby package contains the necessary Ruby bindings for the OpenNebula usage. These services can be installed with the following command:

```
# yum install opennebula-server opennebula-sunstone opennebula-ruby
```

The installation creates the oneadmin user with a random password located at `~/.one/one_auth` as well as a SSH key.

The SSH key allows the oneadmin user to access the virtualization nodes without inserting the password. This is achieved by granting the user with read and write privileges over the `~/.ssh/config` file:

```
# chmod 600 ~/.ssh/config
```

and by specifying the following `~/.ssh/config` file configuration:

```
Host *
    StrictHostKeyChecking no
    UserKnownHostsFile /dev/null
```

OpenNebula also provides a script that installs any required Ruby gems as well as some development libraries packages. This script can be executed from the /usr/share/one/ directory :

```
# cd /usr/share/one
# ./install_gems
```

The Sunstone and OCCI servers listen, by default, to the loopback interface. To access these services from other machines in the network it is necessary to edit `/etc/one/sunstone-server.conf` and `/etc/one/occi-server.conf` server configuration files and change the :host: 127.0.0.1 loopback address to the defined fixed IP address, *e.g.*, :host: 192.168.1.16. Then, to enable the access to these services it is necessary to add the following iptables rules:

```
# iptables -I INPUT -p tcp --dport 9869 -m state \
--state=NEW,ESTABLISHED,RELATED -j ACCEPT
# iptables -I INPUT -p tcp --dport 4567 -m state \
--state=NEW,ESTABLISHED,RELATED -j ACCEPT

# service iptables save
# service iptables restart
```

After installing and configuring OpenNebula's services, it is necessary to launch them and to configure boot startup:

```
# service opennebula start
# service messagebus start
# service opennebula-sunstone start
# occi-server start
# chkconfig opennebula on
# chkconfig messagebus on
# chkconfig messagebus opennebula-sunstone on
```

7.1.2.3 KVM Hypervisor Installation

The KVM Hypervisor Installation procedure is done by installing the opennebula-node-kvm package:

```
# yum localinstall opennebula-node-kvm
```

This package installs the KVM hypervisor and the necessary configuration files to work with the oneadmin user. After the installation and configuration of the hypervisor, it is necessary to start and configure the libvirt service boot startup:

```
# service libvirtd start
# chkconfig libvirtd on
```

7.1.2.4 Virtual Resources Allocation

Prior to running virtual machines with OpenNebula, it is necessary to create a virtual network, an image and a virtual machine template.

The OpenNebula virtual networks are based on Linux bridging and they are divided into Fixed and Ranged. A Fixed virtual network has fixed IP addresses while a Ranged virtual network has a pre-defined network address and number of IP addresses. A Fixed virtual network with ten IP addresses is created, using a network template, *i.e.*, a template file (MyNetwork.one), with the following configuration:

```
# cat << EOT > MyNetwork.one
NAME = "private"
TYPE = FIXED

BRIDGE = br0

LEASES = [ IP=192.168.1.110 ]
LEASES = [ IP=192.168.1.111 ]
LEASES = [ IP=192.168.1.112 ]
LEASES = [ IP=192.168.1.113 ]
LEASES = [ IP=192.168.1.114 ]
LEASES = [ IP=192.168.1.115 ]
LEASES = [ IP=192.168.1.116 ]
LEASES = [ IP=192.168.1.117 ]
LEASES = [ IP=192.168.1.118 ]
LEASES = [ IP=192.168.1.119 ]

EOT
```

Once the virtual network template has been defined, the virtual network can be created by the oneadmin user using Opennebula CLI command onevnet:

```
$ onevnet create mynetwork.one
```

The OS image used is a CentOS-6.4_x86_64 OS image in the QEMU QEMU Copy on Write Version 2 (QCOW2) image format, which is supported by the Quick Emulator (QEMU) virtual machine monitor. This OS image was registered with OpenNebula in the standard Linux FS using the CLI command oneimage as oneadmin user:

```
$ oneimage create \
    --name "CentOS-6.4_x86_64" \
    --path "http://us.cloud.centos.org/i/one/c6-x86_64-20130910-1.qcow2.bz2" \
    --driver qcow2 \
    --datastore default
```

The image, which is stored in the provided path, remains in locked state until it is ready for use.

The virtual machine template defines the virtual machine with the specified name, CPU, RAM and also attaches the disk (in this case the registered image), the network as well as other useful parameters that the virtual machine requires (*e.g.*, SSH keys and network contextualization). The virtual machine template can be created by the oneadmin user using the CLI command onetemplate :

```
$ onetemplate create \
    --name "CentOS-6.4" \
    --cpu 1 --vcpu 1 --memory 512 \
    --arch x86_64 --disk "CentOS-6.4_x86_64" --nic "private" --vnc \
    --ssh
```

With the virtual machine template defined, the virtual machine can be generated by the oneadmin user using the CLI command onetemplate:

```
$ onetemplate instantiate "CentOS-6.4" --name "Test VM"
```

The allocation of virtual resources was conducted without any problems, confirming that the OpenNebula IaaS platform was successfully installed and configured. For troubleshooting, OpenNebula provides logs at /var/log/one.

7.1.3 OpenStack Installation

OpenStack installation is divided in five procedures : (*i*) OS Configuration; (*ii*) Keystone Installation; (*iii*) Glance Installation; (*iv*) Nova Installation and (*v*) Virtual Resources Allocation.

7.1.3.1 OS Configuration

The OS Configuration procedure sets the required software packages and OS configurations required for the installation of OpenStack.

Like OpenNebula, the NIC eth0 has to be configured to use a network bridge br0, replacing `/etc/sysconfig/network-scripts/ifcfg-eth0` file parameters with:

```
DEVICE=eth0
BOOTPROTO=none
NM_CONTROLLED=no
ONBOOT=yes
TYPE=Ethernet
BRIDGE=br0
```

and the ifcfg-br0 file that contains the network bridge information needs also to be created with a public static IP address:

```
# cat << EOT > /etc/sysconfig/network-scripts/network-scripts/ifcfg-br0
DEVICE=br0
TYPE=Bridge
IPADDR=192.168.1.17
NETMASK=255.255.255.0
ONBOOT=yes
BOOTPROTO=static
NM_CONTROLLED=no
EOT
```

The host name osnode is set to simplify the services configuration process. It is defined using the hostname CLI command:

```
# hostname osnode
```

Then, the host name must be specified accordingly in the configuration file `/etc/sysconfig/network` to ensure that the change persists after reboot. This is done by changing the line starting with HOSTNAME=:

```
HOSTNAME=osnode
```

Finally, the host name and public static IP address is added to the `/etc/hosts` file:

```
192.168.1.17     osnode
```

At this stage, the network service needs to be restarted for changes to take effect:

```
# service network restart
```

The OpenStack services require a database to store information. This way, before proceeding with OpenStak packages installation, it is necessary to install a database management system. The database system adopted is MySQL, which is an open-source relational database management system. To install MySQL it is necessary to install the MySQL client and server packages as well as the Python library:

```
# yum install mysql mysql-server MySQL-python
```

Then, in order to to set the bind-address with the IP address of the machine, the following statement is added to the `/etc/my.cnf` file:

```
[mysqld]
...
bind-address            = 192.168.1.17
```

Next, the MySQL database server has to be started and configured to launch automatically when the system boots:

```
# service mysqld start
# chkconfig mysqld on
```

Finally, it is also necessary to define a root password for the MySQL database and erase the anonymous users that are created when the database is first started. This is done by following the instructions of mysql_secure_installation command:

```
# mysql_install_db
# mysql_secure_installation
```

The first time the root password is set for the MySQL Relational Database Management System (RDBMS), the system asks for a password, which is not defined yet. This step can be skipped by pressing the enter key. The mysql_secure_installation CLI command presents a number of other options to secure the database installation. All these prompts can be answered with yes.

With MySQL configured, the system is ready for OpenStack installation. The required OpenStack packages are available on the RDO repository. This repository provides packages that work on Red Hat Enterprise Linux (RHEL) 6, compatible versions of CentOS and Fedora 19. The RDO repository access is enabled by downloading and installing the rdo-release-havana package:

```
# yum install http://repos.fedorapeople.org/repos/openstack/
openstack-havana/rdo-release-havana-6.noarch.rpm
```

Together with the rdo-release-havana package it is necessary to install the openstack-utils and openstack-selinux packages:

```
# yum install openstack-utils openstack-selinux
```

While the rdo-release-havana package contains utility programs that make installation and configuration easier, the openstack-selinux package includes the policy files that are required to configure the Security-Enhanced Linux (SELinux) during the OpenStack installation.

Next, it is necessary to upgrade and reboot the system:

```
# yum upgrade
# reboot
```

After the system reboot, the next step is to install the messaging queue server Qpid (RabbitMQ and ZeroMQ can also be used):

```
# yum install qpid-cpp-server memcached
```

and disable the Qpid authentication by editing `/etc/qpidd.conf` file and changing the auth option to no:

```
auth=no
```

After disabling the default Qpid authentication mechanism, the Qpid service can be started and configured to launch at system boot using the following commands:

```
# service qpidd start
# chkconfig qpidd on
```

7.1.3.2 Keystone Installation

Keystone is the OpenStack identity service and enables user management as well as a catalogue of available services together with their API endpoints. It can be installed using the following command:

```
# yum install openstack-keystone python-keystoneclient
```

Since the identity service stores the identity data in the MySQL database, it is necessary to specify the location of the database with the Keystone user name and KEYSTONE_DBPASS password in the `/etc/keystone/keystone.conf` configuration file:

```
# openstack-config --set /etc/keystone/keystone.conf \
   sql connection mysql://keystone:KEYSTONE_DBPASS@osnode/keystone
```

Then, the openstack-db command is used to create the Keystone database and user (keystone user and KEYSTONE_DBPASS password):

```
# openstack-db --init --service keystone --password KEYSTONE_DBPASS
```

Next, it is necessary to define an authorization token to be used as a shared secret between the identity service and other OpenStack services. Openssl is used to generate and stores the token in the configuration file `/etc/keystone/keystone.conf`:

```
# ADMIN_TOKEN=$(openssl rand -hex 10)
# echo $ADMIN_TOKEN
# openstack-config --set /etc/keystone/keystone.conf DEFAULT \
   admin_token $ADMIN_TOKEN
```

By default, Keystone uses Public Key Infrastructure (PKI) tokens for signing keys and certificates:

```
# keystone-manage pki_setup --keystone-user keystone --keystone-group keystone
# chown -R keystone:keystone /etc/keystone/* /var/log/keystone/keystone.log
```

To apply this configuration, Keystone must be started and configured to automatically start at system boot:

```
# service openstack-keystone start
# chkconfig openstack-keystone on
```

With Keystone installed and running, the users, tenants and roles must be created to allow access to services and endpoints. Since, by default, no users are defined, the authorization token created earlier must be used to interact with Keystone. To simplify the process, the OS_SERVICE_TOKEN and the OS_SERVICE_ENDPOINT environment variables can be set to contain the ADMIN_TOKEN and the identity service endpoint:

```
# export OS_SERVICE_TOKEN=$ADMIN_TOKEN
# export OS_SERVICE_ENDPOINT=http://osnode:35357/v2.0
```

Then, is necessary to create a tenant for the administrative user and a tenant for the remaining OpenStack services:

```
# keystone tenant-create --name=admin --description="Admin Tenant"
# keystone tenant-create --name=service --description="Service Tenant"
```

In the case of the administrative user, the user credentials (admin with password ADMIN_PASS) and email address (*e.g.*,1050758@isep.ipp.pt) must be set:

```
# keystone user-create --name=admin --pass=ADMIN_PASS \
  --email=1050758@isep.ipp.pt
```

This is followed by the definition of a role for the administrative tasks (also called admin):

```
# keystone role-create --name=admin
```

and by the assignment of the admin role and the admin tenant to the admin user:

```
# keystone user-role-add --user=admin --tenant=admin --role=admin
```

After creating the admin user, tenant and role it is necessary to register the corresponding Keystone service and API endpoint:

```
# keystone service-create --name=keystone --type=identity \
  --description="Keystone Identity Service"
```

Using the returned service ID, the API endpoint for Keystone can be specified (the_service_id must be substituted by the returned service ID):

```
# keystone endpoint-create \
  --service-id=the_service_id \
  --publicurl=http://osnode:5000/v2.0 \
  --internalurl=http://osnode:5000/v2.0 \
  --adminurl=http://osnode:35357/v2.0
```

With the Keystone service API endpoints configured, the –os- variables (–os-username, –os-password, –os-tenant-name and –os-auth-url) can be set in the environment to simplify command-line usage. This procedure can be made by creating a openrc.sh file containing the values of the –os-* variables in environment variables:

```
# cat << EOT > openrc.sh
export OS_USERNAME=admin
export OS_PASSWORD=ADMIN_PASS
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://osnode:35357/v2.0
EOT

# source openrc.sh
```

After sourcing this file, the Keystone CLI commands can be instantiated without using the –os- variables. Finally, the access from other machines in the network to the Keystone API endpoints can be granted by adding the following iptables rules:

```
# iptables -I INPUT -p tcp -m multiport --dports 5000,35357 -j ACCEPT

# service iptables save
# service iptables restart
```

7.1.3.3 Glance Installation

Glance is the OpenStack image service and enables users to discover, register and retrieve virtual machine images. It can be installed using the following command:

```
# yum install openstack-glance
```

Like OpenNebula, this service uses the Linux FS (/var/lib/glance/images/) to store images. The image service provides the glance-api and glance-registry services, each with its own configuration file that must include the location of the MySQL RDBMS (GLANCE_DBPASS is the Image Service database password):

```
# openstack-config --set /etc/glance/glance-api.conf \
  DEFAULT sql_connection mysql://glance:GLANCE_DBPASS@osnode/glance
# openstack-config --set /etc/glance/glance-registry.conf \
  DEFAULT sql_connection mysql://glance:GLANCE_DBPASS@osnode/glance
```

Then, the Image Service database and the database user are created using the openstack-db CLI command:

```
# openstack-db --init --service glance --password GLANCE_DBPASS
```

The glance user is created in a similar way to the keystone admin user and associated with the service tenant and admin role:

```
# keystone user-create --name=glance --pass=GLANCE_PASS \
  --email=glance@example.com
# keystone user-role-add --user=glance --tenant=service --role=admin
```

At this stage, is necessary to configure Glance (both glance-api and glance-registry configuration files) to use the identity service for authentication:

```
# openstack-config --set /etc/glance/glance-api.conf keystone_authtoken \
  auth_uri http://osnode:5000
# openstack-config --set /etc/glance/glance-api.conf keystone_authtoken \
  auth_host osnode
# openstack-config --set /etc/glance/glance-api.conf keystone_authtoken \
  admin_tenant_name service
# openstack-config --set /etc/glance/glance-api.conf keystone_authtoken \
  admin_user glance
# openstack-config --set /etc/glance/glance-api.conf keystone_authtoken \
  admin_password GLANCE_PASS
# openstack-config --set /etc/glance/glance-api.conf paste_deploy \
  flavor keystone
# openstack-config --set /etc/glance/glance-registry.conf keystone_authtoken \
  auth_uri http://osnode:5000
# openstack-config --set /etc/glance/glance-registry.conf keystone_authtoken \
  auth_host osnode
# openstack-config --set /etc/glance/glance-registry.conf keystone_authtoken \
  admin_tenant_name service
# openstack-config --set /etc/glance/glance-registry.conf keystone_authtoken \
  admin_user glance
# openstack-config --set /etc/glance/glance-registry.conf keystone_authtoken \
  admin_password GLANCE_PASS
# openstack-config --set /etc/glance/glance-registry.conf paste_deploy \
  flavor keystone
```

To grant access to the Glance database, the Glance credentials must be added to the `/etc/glance/glance-api-paste.ini` and `/etc/glance/glance-registry-paste.ini` files. First, these files need to be copied to the correct location:

```
# cp /usr/share/glance/glance-api-dist-paste.ini \
/etc/glance/glance-api-paste.ini
# cp /usr/share/glance/glance-registry-dist-paste.ini \
/etc/glance/glance-registry-paste.ini
```

Then, each file is edited to set the following options in the [filter:authtoken] section (any other existing option should remain as it is):

```
[filter:authtoken]
paste.filter_factory=keystoneclient.middleware.auth_token:filter_factory
auth_host=osnode
admin_user=glance
admin_tenant_name=service
admin_password=GLANCE_PASS
```

The next step is to register Glance with Keystone so that other OpenStack services can locate it. This is made by registering the service and creating its endpoint:

```
# keystone service-create --name=glance --type=image \
  --description="Glance Image Service"
```

The ID returned by the service-create CLI command is used to create the endpoint (the the_service_id must be substituted by the provided ID):

```
# keystone endpoint-create \
  --service-id=the_service_id \
  --publicurl=http://osnode:9292 \
  --internalurl=http://osnode:9292 \
  --adminurl=http://osnode:9292
```

Finally, the glance-api and glance-registry services have to be started and configured for automatic launch at system boot:

```
# service openstack-glance-api start
# service openstack-glance-registry start
# chkconfig openstack-glance-api on
# chkconfig openstack-glance-registry on
```

To enable the access to the Glance endpoint from other machines in the network, the firewall must be configured using the following iptables rules:

```
# iptables -I INPUT -p tcp -m multiport --dports 9292 -j ACCEPT

# service iptables save
# service iptables restart
```

7.1.3.4 Nova Installation

Nova is the OpenStack compute service used to host and manage cloud computing resources. It can be installed using the following command:

```
# yum install openstack-nova python-novaclient
```

First, similarly to the previous installed OpenStack services, the location of MySQL RDBMS must be set in /etc/nova/nova.conf file using the nova user and NOVA_DBPASS password:

```
# openstack-config --set /etc/nova/nova.conf \
  database connection mysql://nova:NOVA_DBPASS@osnode/nova
```

This file (/etc/nova/nova.conf) must, then, be configured to use the Qpid message broker:

```
# openstack-config --set /etc/nova/nova.conf \
  DEFAULT rpc_backend nova.openstack.common.rpc.impl_qpid
# openstack-config --set /etc/nova/nova.conf DEFAULT qpid_hostname osnode
```

Next, using the openstack-db command, the Nova service database and user are created:

```
# openstack-db --init --service nova --password NOVA_DBPASS
```

Then, the my_ip, vncserver_listen and vncserver_proxyclient_address options need to be configured to use the IP address of the machine:

```
# openstack-config --set /etc/nova/nova.conf DEFAULT \
my_ip 192.168.1.17
# openstack-config --set /etc/nova/nova.conf DEFAULT \
vncserver_listen 192.168.1.17
# openstack-config --set /etc/nova/nova.conf DEFAULT \
vncserver_proxyclient_address 192.168.1.17
```

The next step is to create a nova user in the Keystone service associated with the service tenant and the admin role:

```
# keystone user-create --name=nova --pass=NOVA_PASS --email=nova@example.com
# keystone user-role-add --user=nova --tenant=service --role=admin
```

Then, the Nova service has to be configured to use the Keystone credentials and the host that runs Glance:

```
# openstack-config --set /etc/nova/nova.conf DEFAULT \
auth_strategy keystone
# openstack-config --set /etc/nova/nova.conf keystone_authtoken \
auth_host osnode
# openstack-config --set /etc/nova/nova.conf keystone_authtoken \
auth_protocol http
# openstack-config --set /etc/nova/nova.conf keystone_authtoken \
auth_port 35357
# openstack-config --set /etc/nova/nova.conf keystone_authtoken \
admin_user nova
# openstack-config --set /etc/nova/nova.conf keystone_authtoken \
admin_tenant_name service
# openstack-config --set /etc/nova/nova.conf keystone_authtoken \
admin_password NOVA_PASS
# openstack-config --set /etc/nova/nova.conf DEFAULT glance_host \
osnode
```

At this stage, the api_paste_config=/etc/nova/api-paste.ini option must be defined in the /etc/nova/nova.conf file. Then, the Nova credentials must be included to the /etc/nova/api-paste.ini file by adding the following options to the [filter:authtoken] section:

```
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
auth_host = osnode
auth_port = 35357
auth_protocol = http
auth_uri = http://osnode:5000/v2.0
admin_tenant_name = service
admin_user = nova
admin_password = NOVA_PASS
```

Once the Nova credentials have been set, it is necessary to register the Nova compute service with the Keystone identity service:

```
# keystone service-create --name=nova --type=compute \
  --description="Nova Compute service"
```

Using the ID property returned, the Nova service endpoint can be created using the following command:

```
# keystone endpoint-create \
  --service-id=the_service_id \
  --publicurl=http://osnode:8774/v2/%\(tenant_id\)s \
  --internalurl=http://osnode:8774/v2/%\(tenant_id\)s \
  --adminurl=http://osnode:8774/v2/%\(tenant_id\)s
```

To implement a simple network configuration based on Linux bridging, the legacy Nova network service that implements DHCP (FlatDHCPManager) is used:

```
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  network_manager nova.network.manager.FlatDHCPManager
# openstack-config --set /etc/nova/nova.conf DEFAULT \
  firewall_driver nova.virt.libvirt.firewall.IptablesFirewallDriver
# openstack-config --set /etc/nova/nova.conf DEFAULT \
network_size 254
# openstack-config --set /etc/nova/nova.conf DEFAULT \
allow_same_net_traffic False
# openstack-config --set /etc/nova/nova.conf DEFAULT \
send_arp_for_ha True
# openstack-config --set /etc/nova/nova.conf DEFAULT \
share_dhcp_address True
# openstack-config --set /etc/nova/nova.conf DEFAULT \
force_dhcp_release True
# openstack-config --set /etc/nova/nova.conf DEFAULT \
flat_interface eth0
# openstack-config --set /etc/nova/nova.conf DEFAULT \
flat_network_bridge br0
# openstack-config --set /etc/nova/nova.conf DEFAULT \
public_interface eth0
```

Then, Nova can be started and configured to automatically launch at system boot:

```
# service openstack-nova-api start
# service openstack-nova-cert start
# service openstack-nova-consoleauth start
# service openstack-nova-scheduler start
# service openstack-nova-conductor start
# service openstack-nova-novncproxy start
# service openstack-nova-compute start
# service openstack-nova-network start
# service libvirtd start
# service messagebus start
# chkconfig openstack-nova-api on
# chkconfig openstack-nova-cert on
# chkconfig openstack-nova-consoleauth on
# chkconfig openstack-nova-scheduler on
# chkconfig openstack-nova-conductor on
# chkconfig openstack-nova-novncproxy on
# chkconfig openstack-nova-compute on
# chkconfig openstack-nova-network on
# chkconfig libvirtd on
# chkconfig messagebus on
```

To enable access to the VNC proxy and the Nova API, the firewall must be configured using the following iptables rules:

```
# iptables -I INPUT -p tcp -m multiport --dports 6080 -j ACCEPT
# iptables -I INPUT -p tcp -m multiport --dports 8774 -j ACCEPT

# service iptables save
# service iptables restart
```

7.1.3.5 Virtual Resources Allocation

With Keystone, Glance and Nova services installed, configured and running, the Openstack IaaS platform is up with the minimum set of components that allows the creation of virtual resources and provision of virtual machines.

The virtual network for the virtual machines can be created using Nova network-create CLI command:

```
# nova network-create vmnet --fixed-range-v4=192.168.2.0/28 \
  --bridge=br0
```

A small test image that is often used for testingOpenStack deployments, the 64-bit CirrOS QCOW2 image, is downloaded and placed in the created `~/images/` folder:

```
$ mkdir ~/images
$ cd ~/images/
$ wget http://cdn.download.cirros-cloud.net/0.3.1/cirros-0.3.1-x86_64-disk.img
```

Then, the image is uploaded to the image service Glance using the image-create CLI command:

```
# glance image-create \
--name="CirrOS 0.3.1" \
--disk-format=qcow2 \
--container-format=bare \
--is-public=true < cirros-0.3.1-x86_64-disk.img
```

The image ID can be consulted using the Glance image-list CLI command:

```
$ glance image-list
```

To provide virtual machine password-less SSH access, a key-pair must be created:

```
$ ssh-keygen
$ cd .ssh
$ nova keypair-add --pub_key id_rsa.pub mykey
```

To launch an instance, a flavour ID must be specified. A flavour is a resource allocation profile that specifies the number of virtual CPU as well as the disk and RAM size that the instances will have. OpenStack provides the following profiles:

- m1.tiny - 512 MB of RAM, 1 vCPU and 1 GB of disk storage capacity;
- m1.small - 2048 MB of RAM, 1 vCPU and 20 GB of disk storage capacity;
- m1.medium - 4096 MB of RAM, 2 vCPU and 40 GB of disk storage capacity;
- m1.large - 8192 MB of RAM, 4 vCPU and 80 GB of disk storage capacity;
- m1.xlarge - 16384 MB of RAM, 8 vCPU and 160 GB of disk storage capacity.

The Nova compute service uses security groups, which are sets of IP filter rules, to be applied to the networking instances. The security groups rules are configured to enable SSH and ping:

```
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
```

Once the virtual resources are created, a virtual machine can be launched using the m1.tiny flavor, a key-pair mykey for password-less access to the VM when using SSH, the cirrOS image (ID = caede980-890f-477d-8856-4cc7ba2c2f39), the default security group profile that enables SSH and ping to the VM and finally the VM name cirrOS:

```
$ nova boot --flavor 1 --key_name mykey \
--image caede980-890f-477d-8856-4cc7ba2c2f39 \
--security_group default cirrOS
```

The allocation of virtual resources was conducted without any problems, confirming that the Keystone, Glance and Nova projects deployed in the OpenStack IaaS platform were successfully installed and configured. For troubleshooting, OpenStack provides logs at /var/log/keystone for Keystone, /var/log/glance for Glance and /var/log/nova for Nova.

7.1.4 CloudStack Installation

The CloudStack installation is composed of five main procedures: (*i*) OS Configuration; (*ii*) Management Server Installation; (*iii*) KVM Hypervisor Installation; (*iv*) Cloud Infrastructure Configuration and (*v*) Virtual Resources Allocation.

7.1.4.1 OS Configuration

The OS Configuration procedure configures OS network for CloudStack usage and sets the CloudStack repository and required services for CloudStack installation.

The NIC eth0 has to be configured to use a network bridge cloudbr0, replacing `/etc/sysconfig/network-scripts/ifcfg-eth0` file parameters with:

```
DEVICE=eth0
BOOTPROTO=none
NM_CONTROLLED=no
ONBOOT=yes
TYPE=Ethernet
BRIDGE=cloudbr0
```

The ifcfg-cloudbr0 file, that specifies the network bridge information, needs also to be created with a public static IP address:

```
# cat << EOT > /etc/sysconfig/network-scripts/network-scripts/ifcfg-cloudbr0
DEVICE=cloudbr0
TYPE=Bridge
IPADDR=192.168.1.18
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
ONBOOT=yes
BOOTPROTO=static
NM_CONTROLLED=no
EOT
```

The host name csnode is set to simplify the services configuration process. It is defined using the hostname CLI command:

```
# hostname csnode
```

To ensure that this host name change persists, the configuration file `/etc/sysconfig/network` is edited and the line starting with HOSTNAME= is changed:

```
HOSTNAME=csnode
```

Then, the host name with the corresponding public fixed IP address is added to the `/etc/hosts` file:

```
192.168.1.18    csnode
```

At this stage, the network service needs to be restarted for changes to take effect:

```
# service network restart
```

The SELinux security module must be set to permissive. The SELinux system security level can be set to permissive by running the following command:

```
# setenforce 0
```

To ensure that it remains in that state after a system boot, the file `/etc/selinux/config` has to be configured to reflect the permissive state:

```
SELINUX=permissive
```

To add the CloudStack repository, it is necessary to create the file `/etc/yum.repos.d/cloudstack.repo` with the following information:

```
# cat << EOT > /etc/yum.repos.d/cloudstack.repo
[cloudstack]
name=cloudstack
baseurl=http://cloudstack.apt-get.eu/rhel/4.3/
enabled=1
gpgcheck=0
EOT
```

CloudStack uses NFS for both primary and secondary storage. This way, two NFS shares need to be set-up. First, nfs-utils has to be installed:

```
# yum install nfs-utils
```

Then, the two directories must be created:

```
# mkdir /primary
# mkdir /secondary
```

To configure the new directories as NFS exports, the `/etc/exports` file has to be edited:

```
/primary *(rw,async,no_root_squash)
/secondary *(rw,async,no_root_squash)
```

Then, it is necessary to uncomment the following configuration values in the `/etc/sysconfig/nfs` file:

```
LOCKD_TCPPORT=32803
LOCKD_UDPPORT=32769
MOUNTD_PORT=892
RQUOTAD_PORT=875
STATD_PORT=662
STATD_OUTGOING_PORT=2020
```

and edit the iptables rules to allow incoming NFS connections:

```
# iptables -A INPUT -s 192.168.1.0/24 -m state \
--state NEW -p udp --dport 111 -j ACCEPT
# iptables -A INPUT -s 192.168.1.0/24 -m state \
--state NEW -p tcp --dport 111 -j ACCEPT
# iptables -A INPUT -s 192.168.1.0/24 -m state \
--state NEW -p tcp --dport 2049 -j ACCEPT
# iptables -A INPUT -s 192.168.1.0/24 -m state \
--state NEW -p tcp --dport 32803 -j ACCEPT
# iptables -A INPUT -s 192.168.1.0/24 -m state \
--state NEW -p udp --dport 32769 -j ACCEPT
# iptables -A INPUT -s 192.168.1.0/24 -m state \
--state NEW -p tcp --dport 892 -j ACCEPT
# iptables -A INPUT -s 192.168.1.0/24 -m state \
--state NEW -p udp --dport 892 -j ACCEPT
# iptables -A INPUT -s 192.168.1.0/24 -m state \
--state NEW -p tcp --dport 875 -j ACCEPT
# iptables -A INPUT -s 192.168.1.0/24 -m state \
--state NEW -p udp --dport 875 -j ACCEPT
# iptables -A INPUT -s 192.168.1.0/24 -m state \
--state NEW -p tcp --dport 662 -j ACCEPT
# iptables -A INPUT -s 192.168.1.0/24 -m state \
--state NEW -p udp --dport 662 -j ACCEPT

# service iptables save
# service iptables restart
```

The NFS service needs to be initiated and configured to start on boot:

```
# service rpcbind start
# service nfs start
# chkconfig rpcbind on
# chkconfig nfs on
```

CloudStack also requires the usage of a database. Like in the case of Openstack, the RDBMS installed was MySQL:

```
# yum -y install mysql-server
```

Then, the following options need to be added to the [mysqld] section of the `/etc/my.cnf` file:

```
innodb_rollback_on_timeout=1
innodb_lock_wait_timeout=600
max_connections=350
log-bin=mysql-bin
binlog-format = 'ROW'
```

With MySQL properly configured, it can be started and configured to start on boot:

```
# service mysqld start
# chkconfig mysqld on
```

Finally, it is necessary to define a root password (DB_PASS) for the MySQL RDBMS and erase the anonymous users created by default by running the mysql_secure_installation daemon:

```
# mysql_install_db
# mysql_secure_installation
```

7.1.4.2 Management Server Installation

The Management Server can be installed by executing the following command:

```
# yum -y install cloud-client
```

Then, it is necessary to create the cloud user in the MySQL RDBMS:

```
# cloudstack-setup-databases cloud:DB_PASS@csnode \
--deploy-as=root
```

At this stage, the Management Server can be started:

```
# cloudstack-setup-management
```

CloudStack uses a number of system virtual machines to provide access the the virtual machines, providing various networking services and managing various storage features. This way, a system VM template must be downloaded and deployed to the mounted NFS /export/secondary share:

```
# /usr/share/cloudstack-common/scripts/storage/secondary/cloud-install-sys-tmplt
-m /secondary -u \
http://download.cloud.com/templates/acton/acton-systemvm-02062012.qcow2.bz2 \
-h kvm -F
```

7.1.4.3 KVM Hypervisor Installation

The KVM agent installation is done by installing the cloud-agent package:

```
# yum -y install cloud-agent
```

Then, libvirt and QEMU need to be configured. The QEMU configuration enables the VNC server and it is done by editing `/etc/libvirt/qemu.conf` and ensuring that the following line is present and uncommented:

```
vnc_listen=0.0.0.0
```

Libvirt is used by Cloudstack for managing virtual machines. Its configuration is performed in `/etc/libvirt/libvirtd.conf` and `/etc/sysconfig/libvirtd` files so that it listens for unsecured TCP connections for live migration and to turn off libvirts attempt to use multicast DNS advertising. The `/etc/libvirt/libvirtd.conf` file is configured by setting the following parameters:

```
listen_tls = 0
listen_tcp = 1
tcp_port = "16059"
auth_tcp = "none"
mdns_adv = 0
```

Additionally, the following line of the `/etc/sysconfig/libvirtd` file must be uncommented:

```
#LIBVIRTD_ARGS="--listen"
```

Finally, the libvirt is restarted:

```
# service libvirtd restart
```

The KVM agent is properly configured and running.

7.1.4.4 Cloud Infrastructure Configuration

The IaaS platform is implemented using CloudStack's Web interface basic set-up option. This Web interface is accessible through the Web browser using the `http://192.168.1.18:8080/client` URL. Then, using the default admin authentication credentials (username admin and password password), the configuration process can be started by choosing the Continue with Basic Setup option.

The cloud configuration process is divided in six sections: (*i*) Admin password modification; (*ii*) Zone creation; (*iii*) Pod creation; (*iv*) Cluster configuration; (*v*) Primary storage configuration; and (*vi*) Secondary storage configuration.

The first section requires the admin password modification. This way the password for the admin user was modified to CS_PASS.

The Zone creation section allows the specification of CloudStack's Zone Name and public and private DNS. Only Public and Internal DNS 1 are mandatory. The Internal DNS is assumed to be capable of resolving internal-only host names, while the Public DNS is provided to the guest virtual machines to resolve public IP addresses. In this specific set-up, no names will be attributed for internal resources. As a result, the Public DNS were configured to use Google's public DNS (8.8.8.8 and 8.8.4.4) and the private DNS were configured to use the local network DNS (193.136.63.3 and 193.136.60.2):

CloudStack Zone:

1. Name: zone1;
2. Public DNS 1: 8.8.8.8;
3. Public DNS 2: 8.8.4.4;
4. Internal DNS 1: 193.136.63.3;
5. Internal DNS 2: 193.136.60.2.

Next, the Pod configuration section allows setting the network for the system virtual machines and the guest virtual machines. In the first page the Pod's name pod1 and the service network were configured as described:

1. Name: pod1;

2. Gateway: 192.168.1.1;
3. Netmask: 255.255.255.0;
4. IP Range: 192.168.1.30 –192.168.1.39.

Then, on the second page, the guest network was specified:

1. Gateway: 192.168.1.1;
2. Netmask: 255.255.255.0;
3. IP Range: 192.168.1.40 –192.168.1.49.

The Cluster configuration section allows the definition of the name and the hypervisor used by a cluster (in this case, cluster1):

1. Name: cluster1;
2. Hypervisor: KVM.

The second page configures the Linux root account access to the physical machine where CloudStack is installed, using the IP address of the physical machine, the root username and password:

1. Host Name: 192.168.1.18;
2. Username: root;
3. Password: $CSnode2014.

With the Cluster configured, the user is directed to a page to set the primary storage, including the NFS protocol, the primary1 name, the address (192.168.1.18) to physical machine and the location path of the primary storage:

1. Name: primary1;
2. protocol: NFS;
3. Server: 192.168.1.18;
4. Path: /primary.

The Secondary storage section allows the configuration of the secondary storage, including the NFS server address and the location path for the secondary storage:

1. NFS Server: 192.168.1.18;
2. Path: /secondary.

7.1.4.5 Virtual Resources Allocation

After CloudStack installation and configuration, the physical machine is ready for the provision of virtual machines. Each VM creation involves the CloudStack service offerings, *i.e.*, the virtual resources available, consisting of compute, disk, network and templates offerings. By default, CloudStack has two compute offerings options:

- Small - with 1 vCPU with 500 MHz and 512 MB of RAM;
- Medium - with 1 vCPU with 1000 MHz and 1024 MB of RAM.

and four disk offerings options:

- Small - with 5 GB of storage capacity;
- Medium - with 20 GB of storage capacity;
- Large - with 100 GB of of storage capacity;
- Custom - with no defined value of capacity.

The default network offering uses the Pod network configuration and a CentOS template image for the guest virtual machines. ISO files, security groups, users, groups, zones and projects are also available features that can be associated with the virtual machines.

To test the CloudStack installation, an Ubuntu 12.04 server 64 bit ISO was registered and used to create a VM via the CloudStack set-up GUI.

The ISO registration procedure is done by clicking on the Templates option in the left navigation bar, selecting ISO images from the Select View and clicking on the Add ISO button. Then, in the Add ISO screen, the following configuration was set:

1. Name: ubuntu-12.04.2-server-amd64;
2. Description: ubuntu-12.04.2-server-amd64;
3. URL: `http://releases.ubuntu.com/precise/ubuntu-12.04.4-server-amd64.iso`;
4. Zones: All zones;

5. Bootable: checked;
6. OS: Ubuntu 12.04 (64-bit);
7. Extractable: unchecked;
8. Public: checked;
9. Featured: checked.

Next, after clicking on the OK button, the Management Server downloads the ISO image from the provided URL and the ISO status column will display "Ready" once it has been successfully downloaded into the secondary storage.

The "Ready" ISO image can, then, be associated with a VM. The VM can be created by clicking on the Instances option in the left navigation bar and choosing the Add Instance option. This procedure launches the VM creation wizard that was configured with the following options:

1. Select a zone: zone1;
2. Select ISO or template: ISO;
3. ubuntu-12.04.2-server-amd64;
4. Compute Offering: Small Instance;
5. Data Disk Offering: Small;
6. security group(s): default;
7. Name: CS-vm.

The allocation of virtual resources was conducted without any problems, confirming that the CloudStack IaaS platform was successfully installed and configured. For troubleshooting, the CloudStack logs are available at /var/log/cloudstack.

7.2 Testing and Evaluation

Once the test bed was assembled, the OpenNebula, OpenStack, CloudStack and PACI driver modules were tested and evaluated in terms of functionality and interoperability performance. These tests consisted of a comparison between the intermediary utilization of the Deltacloud API and the direct use of the dedicated IaaS platforms API operations in terms of time response per operation (*i.e.*, the total amount of time required to perform a HTTP request to the API and obtain

the returned response) as well as the HTTP request packet length and HTTP response content length. The execution of the API operations (either to the Deltacloud API and the dedicated IaaS platform API) and the measurement of its time response was performed with the cURL command line tool, using the following syntax and options:

```
$ curl --verbose -X <custom request method> -H <header> \
-d '<request data>' -u '<user:pass>' 'URL' -w '<variables>'
```

- The –v, –verbose option shows additional information (*e.g.*, HTTP header data sent and received by curl) useful for debugging;
- The –X, –request option specifies the custom request method to use, which in HTTP are the HTTP verbs (*e.g.*, –X POST);
- The –H, –header option specifies extra header to include in the HTTP request (*e.g.*, –H "Content-Type: application/xml");
- The –d, –data option specifies the data to send by the HTTP POST request;
- The –u, –user option specifies the user credentials for server authentication. If only the user is provided, curl will prompt for a password. This option can be combined with the –digest option to use an authentication scheme that prevents the password from being sent in clear text;
- The URL formed by the API endpoint and the path of the resource (*e.g.*, `http://192.168.1.16:4567/compute/9'`);
- The –w, –write_out option enables the combination of plain text with defined curl variables, using a string syntax, to display messages after a completed and successful operation. This operation was used to obtain the time response for the tested operations (*e.g.*, –w '\n total time of the operation: %{time_total} seconds').

A complete list of cURL options is available at [189]. The HTTP request packet length and response payload were measured using the Wireshark software, analysing the local loopback interface lo (127.0.0.1 IP address) and the test network interface eth0 (192.168.1.105 IP address) of the laptop computer for the Deltacloud API request operations and the dedicated IaaS platforms API request operations, respectively. A filter formed by the HTTP protocol and the endpoint IP address (*e.g.*, http&&ip.addr==192.168.1.16) was adopted to catch and inspect the desired HTTP request and response packets. For the sake of these tests, the HTTP Secure Sockets Layer (SSL) encryption security procedure was purposely disregarded.

The following sections present the tests performed with each driver module, including the analysis of the driver operation as well as the obtained results constituted by the mean and standard deviation of ten runs per operation.

7.2.1 OpenNebula Interaction Tests and Results

To test the interaction with the OpenNebula IaaS platform, the Deltacloud server was started with the OpenNebula driver and the endpoint URL of the OCCI server API using the localhost 3001 port:

```
$ deltacloudd -i opennebula -P "http://192.168.1.16:4567" -p 3001
Starting Deltacloud API :: opennebula ::
http://192.168.1.16:4567 :: http://localhost:3001/api
Thin web server (v1.6.1 codename Death Proof)
Debugging ON
Maximum connections set to 1024
Listening on localhost:3001, CTRL+C to stop
```

The OCCI server implemented in the OpenNebula IaaS platform uses by default a HTTP basic authentication [126] formed by the user identification (oneadmin) and a Secure Hash Algorithm 1 (SHA-1) hashed password (defined in the `~/.one/one_auth` file) separated by a colon (":"). However, since the OpenNebula driver uses an authentication method (defined in the Client class of the occi_client.rb file) that implements the SHA-1 hash function on the password, the authentication with the Deltacloud server used the unprotected plain text password of the OpenNebula account.

Along with this security problem (that can be resolved by the adoption of cryptographic protocols such as SSL), the driver supplied with the Deltacloud framework had two minor bugs that had to be corrected. The first problem was related with an id argument mismatch in the destroy_image method that was included in the opennebula_driver.rb file. The second problem was associated with the instantiation of an unused xmlfile argument in the delete method, which was specified in the occi_client.rb file. Both problems were corrected and reported. Additionally, with this test confirmed that the driver works with the current version of OpenNebula (version 4.4).

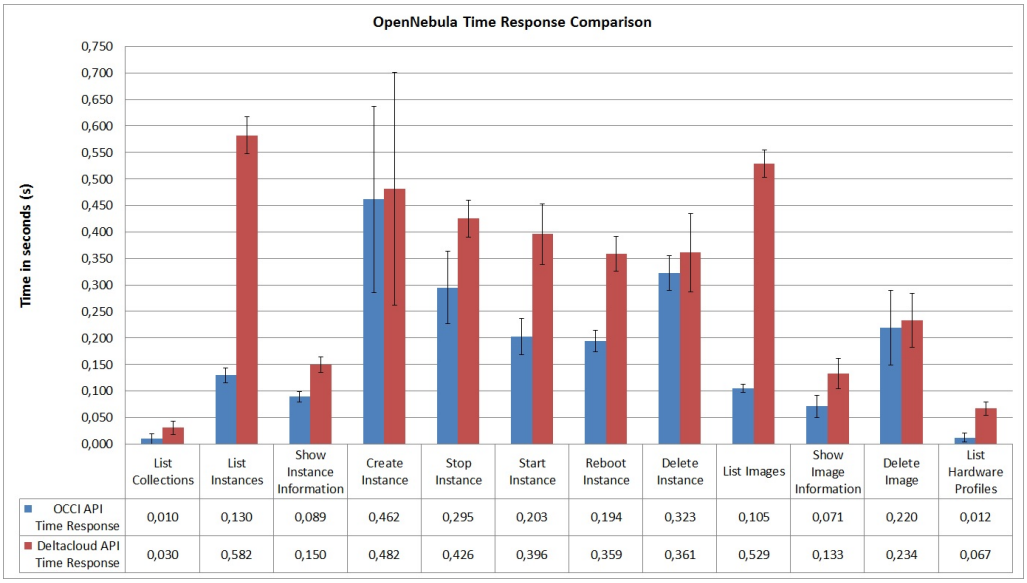
To maximize the execution of the interoperable tests and to simulate a normal usage of the resources, the operations tested were executed in the following order:

1. List collections;
2. List Images;
3. Show Image Information;
4. List Hardware Profiles;
5. Create Instance;

6. List instances;
7. Show Instance Information;
8. Stop Instance;
9. Start Instance;
10. Reboot Instance;
11. Delete Instance;
12. Delete Image.

Since each operation was executed ten times, it was necessary to allocate ten images for each interaction (in fact, for the Deltacloud API and the dedicated OCCI API tests, twenty images were allocated). This way, and to save time, a ttylinux - kvm image, downloaded from the OpenNebula marketplace (accessed via Sunstone Web dashboard)[190] with a tiny footprint (40 MB), was used. In the Create Instance operation the smallest hardware profile configuration (1 virtual CPU, 512 MB of RAM and 40 MB of disk size) was used in order to guarantee that the deployed OpenNebula node could handle the creation of ten consecutive virtual machines (instances).

The time response test results, displaying the mean and standard deviation per operations and for both the Deltacloud and the direct API interactions, are presented in the chart of Figure 7.2.

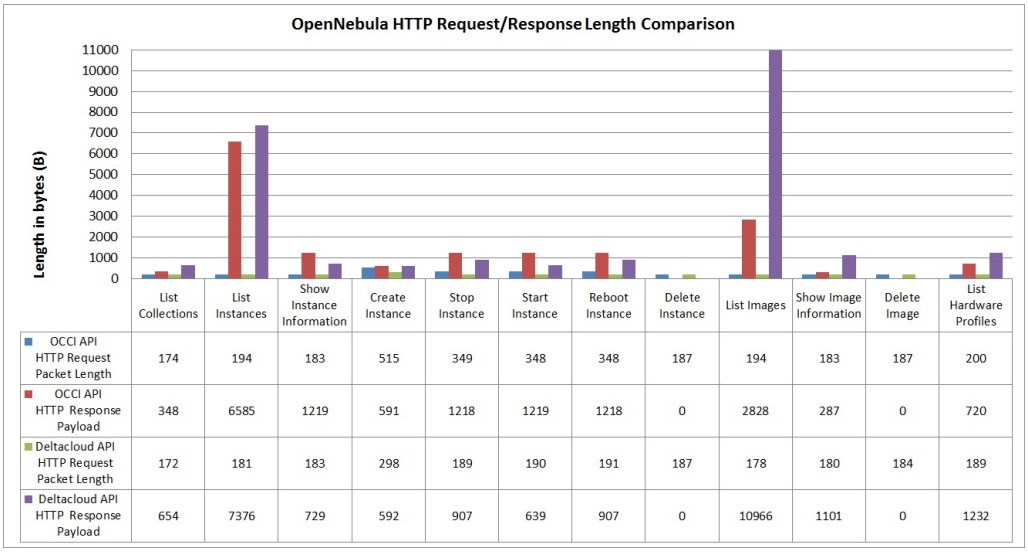


	List Collections	List Instances	Show Instance Information	Create Instance	Stop Instance	Start Instance	Reboot Instance	Delete Instance	List Images	Show Image Information	Delete Image	List Hardware Profiles
OCCI API Time Response	0,010	0,130	0,089	0,462	0,295	0,203	0,194	0,323	0,105	0,071	0,220	0,012
Deltacloud API Time Response	0,030	0,582	0,150	0,482	0,426	0,396	0,359	0,361	0,529	0,133	0,234	0,067

Figure 7.2: OpenNebula time response comparison.

This chart shows that the intermediary interaction with the Deltacloud API, using the OpenNebula driver module, increases the operations time response, particularly the operations that return lists of information *i.e.*, the List Instances, List Images and List Hardware Profiles operations. It is also possible to observe that the Delete Image and Create Instance operations have almost the same average response time, and the collected samples from the last referred operation (Create Instance operation) varied significantly.

The results of the HTTP request packet length and returned payload per operation in both cases are presented in the chart of Figure 7.3. They reinforce the interpretation of the time response results from Figure 7.2.



	List Collections	List Instances	Show Instance Information	Create Instance	Stop Instance	Start Instance	Reboot Instance	Delete Instance	List Images	Show Image Information	Delete Image	List Hardware Profiles
OCCI API HTTP Request Packet Length	174	194	183	515	349	348	348	187	194	183	187	200
OCCI API HTTP Response Payload	348	6585	1219	591	1218	1219	1218	0	2828	287	0	720
Deltacloud API HTTP Request Packet Length	172	181	183	298	189	190	191	187	178	180	184	189
Deltacloud API HTTP Response Payload	654	7376	729	592	907	639	907	0	10966	1101	0	1232

Figure 7.3: OpenNebula HTTP request/response length comparison.

Figure 7.3 shows that the HTTP requests packets length of the OpenNebula OCCI API operations are slightly bigger than the ones of the Deltacloud API operations. This can be observer mainly in the Create Instance, Stop Instance, Start Instance and Reboot Instance operations. On the other hand, the length of the HTTP response payloads varies and is bigger for the responses of Deltacloud API List Collections, List Instances, List Images, Show Image Information and List Hardware Profiles operations (being the List Instances and List Images the responses containing the greater measured values), identical in the Create Instance operation and larger for the responses of the OpenNebula OCCI API Show Instance, Stop Instance, Start Instance and Reboot Instance operations. In the case of the Delete Instance and Delete Image operations, the returned payload length is nil since these are responses without HTTP body.

7.2.2 OpenStack Interaction Tests and Results

To test the interaction with the OpenStack IaaS platform, the Deltacloud server was started with the OpenStack driver and the endpoint URL of the OpenStack Identity Service (Keystone) API using the localhost 3001 port:

```
$ deltacloudd -i openstack -P "http://192.168.1.17:5000/v2.0/" -p 3001
Starting Deltacloud API :: openstack
:: http://192.168.1.17:5000/v2.0/ :: http://localhost:3001/api
Thin web server (v1.6.1 codename Death Proof)
Debugging ON
Maximum connections set to 1024
Listening on localhost:3001, CTRL+C to stop
```

To authenticate with the OpenStack services, it was necessary to first perform an authentication request to the OpenStack identity service (Keystone) API. This authentication request consists of a HTTP request body, usually formatted in JSON, containing the credentials of the user (the user name, password, and optionally, the name or ID of the tenant). The following curl command exemplifies a HTTP authentication request to the Keystone API:

```
curl -H "Content-Type: application/json" -X POST \
-d '{"auth": {"tenantName": "admin", "passwordCredentials":
{"username": "admin", "password": "ADMIN_PASS"}}}' \
'http://192.168.1.17:5000/v2.0/tokens'
```

The returned HTTP response contained the authentication token and the tenant ID (which is required to access OpenStack resources). The authentication token is valid for one day and has to be included in the HTTP request headers (using the X-Auth-Token header) of all HTTP requests operations.

In the case of the authentication with the Deltacloud Server API, the HTTP basic authentication was used, exposing the credentials and providing the tenant name concatenated with the user name separated by a plus character, *i.e.*, user+tenant_name:password.

As in the case of the OpenNebula driver, the OpenStack driver exposes the credentials if the connection to the Deltacloud Server is not secure (using cryptographic protocols). The connection to the OpenStack Keystone server is also insecure, although the credentials are only exchanged once in 24 hours.

Although the OpenStack driver is fully functional (with the Keystone, Nova and Glance API), since it lacks a start and stop VM operations in the OpenStack rubygem, the VM life-cycle used by the OpenStack driver differs from the one provided by the Nova project. Moreover, for an unexplained reason, the method that implements the Delete Instance operation was defined as an alias of the Stop Instance operation method. Thus, this implementation causes the destruction of a started VM when the Stop Instance operation is invoked. More recently, a Polish company named Cybercom proposed an OpenStack driver implementation with separate Start Instance and Stop Instance operations [191]. This newer version

of the driver was only announced after the execution of these tests and, for this reason, it was not used.

The OpenStack node deployed in the test bed was only configured with the Keystone, Glance and Nova OpenStack services, meaning that the Swift and Cinder implemented collections were not tested. This way, the OpenStack driver operations tested were the following:

1. List Images;
2. Show Image Information;
3. List Hardware Profiles;
4. Create Instance;
5. List Instances;
6. Show Instance Information;
7. Reboot Instance;
8. Delete Instance;
9. Delete Image.

Once again, ten images were created for each interaction. This time, the same image used to test the Glance service installation - the 64-bit CirrOS QCOW2 image - using the same CLI command (the glance image-create CLI command) was allocated. The smallest hardware profile configuration (1 virtual CPU, 512 MB of RAM and 10 GB of Disk) was used for the Create Instance operation so that the OpenStack node could handle the creation of ten consecutive virtual machines (instances).

The time response results are presented in the chart of Figure 7.4. Since the authentication request is performed in each Deltacloud API operation when using the OpenStack driver, the mean value of HTTP authentication request time response was added to the mean value of the OpenStack services (Nova and Glance) API operations time response.

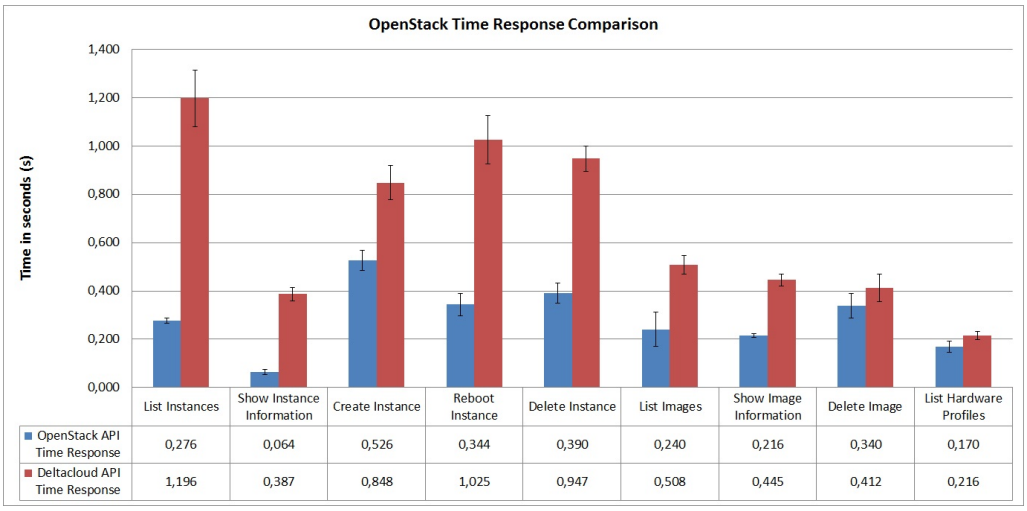
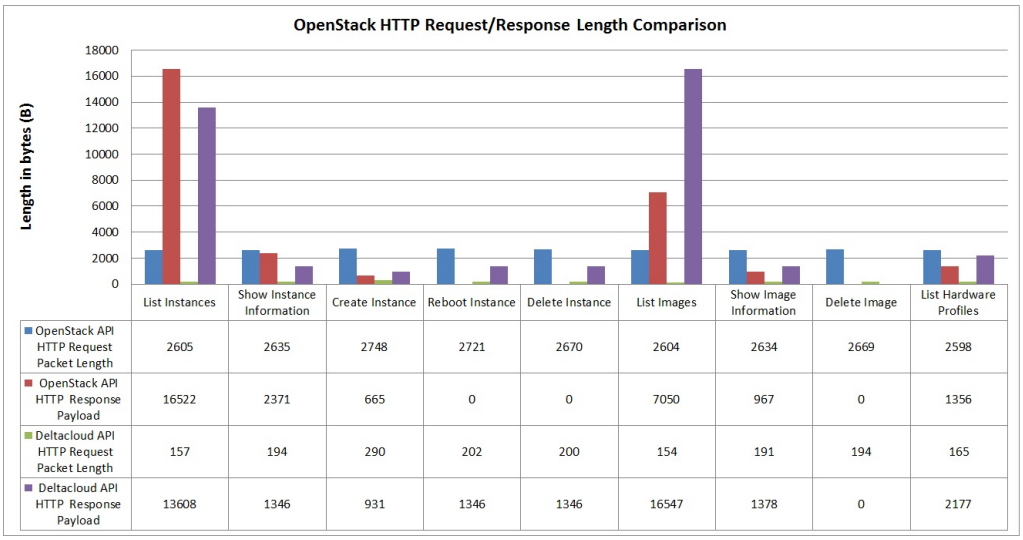


Figure 7.4: OpenStack time response comparison.

As expected, the time response from the Deltacloud API operations is significantly higher than the time response of the OpenStack services API (Keystone, Nova and Glance) operations. This is clearly observed in every listed operation with the exception of the Delete Image and List Hardware Profiles operations. In fact, the time response mean of some operations like the List Instances, Create Instance and Reboot Instance Deltacloud API operations reached values higher than the operations performed to the Deltacloud API using the OpenNebula driver. There are three main reasons for this behaviour: (*i*) the instantiation of multiple OpenStack operations in the OpenStack driver to execute a single Deltacloud operation; (*ii*) the incremental time response of the OpenStack API operations in comparison with the OpenNebula OCCI API operations time response, *e.g.*, the List Instances OpenNebula OCCI API operation has a time response mean of approximately 0,120 s while the same OpenStack API operation has a time response value mean of approximately 0,230 s; and (*iii*) the HTTP response payload length, that is illustrated in the chart of Figure 7.5, which presents the results of the HTTP request packet length and returned payload for each executed operations in both cases (using the Deltacloud API and the OpenStack services API).



	List Instances	Show Instance Information	Create Instance	Reboot Instance	Delete Instance	List Images	Show Image Information	Delete Image	List Hardware Profiles
OpenStack API HTTP Request Packet Length	2605	2635	2748	2721	2670	2604	2634	2669	2598
OpenStack API HTTP Response Payload	16522	2371	665	0	0	7050	967	0	1356
Deltacloud API HTTP Request Packet Length	157	194	290	202	200	154	191	194	165
Deltacloud API HTTP Response Payload	13608	1346	931	1346	1346	16547	1378	0	2177

Figure 7.5: OpenStack HTTP request/response length comparison.

The authentication procedure (authentication token) used by OpenStack is reflected in the values of the HTTP request packet length of the OpenStack API operations, which is substantially bigger than the values of the corresponding Deltacloud API operations. The HTTP response payload, on the other hand, is not so linear. The OpenStack API List Instance and Show Instance Information operations return bigger payloads than the corresponding Deltacloud API operations, the Delete Image operation returns the same payload in both cases (a void HTTP response body) and the remaining operations return a smaller payload than the Deltacloud API counterparts. In the case of the Reboot Instance and Delete Instance operations, the HTTP responses of the OpenStack API do not provide a HTTP body. Usually, the HTTP response of the Delete Instance operation defined by the Deltacloud API does not provide a HTTP body also. However, since the OpenStack driver defined the destroy_instance method as an alias of the stop_instance method, the deletion of an OpenStack instance with the Deltacloud API stops the instance. In fact it sends the delete instance instruction, but returns the Stop Instance operation information.

7.2.3 CloudStack Interaction Tests and Results

The third party CloudStack driver was added to Deltacloud in order to integrate The CloudStack IaaS platform, but it did not work. From the analysis of the driver implementation, it was possible to conclude that, although the driver is based on the Deltacloud mock driver, it is incomplete and, thus, non functional. The driver implementation includes a Ruby CloudStack client that can be used to interact successfully with the CloudStack IaaS platform, but it does not imple-

ment the Deltacloud collections. Since the development of the CloudStack driver falls out of the scope of this project, the integration of the CloudStack platform was dropped.

7.2.4 PACI Interaction Tests and Results

The interaction with the PACI IaaS platform was tested by starting the Deltacloud server with the PACI driver and the URL endpoint of the Lunacloud PACI server API, using the localhost 3001 port:

```
$ deltacloudd -i paci -P "http://apicontrol.lunacloud.com:4465/paci/v1.0/" -p 3001
Starting Deltacloud API :: paci :: http://apicontrol.lunacloud.com:4465/paci/v1.0/
:: http://localhost:3001/api
Thin web server (v1.6.1 codename Death Proof)
Debugging ON
Maximum connections set to 1024
Listening on localhost:3001, CTRL+C to stop
```

The PACI API uses a HTTP basic authentication method without any hash function applied to the user's password. Nevertheless, Lunacloud uses cryptographic protocols to protect the HTTP communications between the users and the PACI IaaS platform via the https://apicontrol.lunacloud.com:4463/paci/v1.0/ endpoint. The PACI API endpoint used to test the PACI driver interaction did not use these cryptographic protocols in order to allow the visualization of the HTTP request/response packets with Wireshark.

The developed PACI driver is fully functional and the interaction with the Deltacloud API showed that, in the case of the Create Load Balancer operations, Deltacloud requires additional parameters, *e.g.*, realm_id, listener_protocol, listener_balancer_port and listener_instance_port, that are not specified by the PACI API Create Load Balancer operation, which requires only a name parameter. Thus, to implement this operation with the Deltacloud API some mock default values were attributed to the realm_id, listener_protocol, listener_balancer_port and listener_instance_port parameters.

Contrary to the open source IaaS platforms (OpenNebula, OpenStack and CloudStack), which were in the same test network as the laptop used to perform the tests, the PACI IaaS platform was in Lunacloud's network, thirteen hoops away from the created test network:

```
$ traceroute apicontrol.lunacloud.com
traceroute to apicontrol.lunacloud.com
(80.172.242.108), 30 hops max, 60 byte packets
 1  192.168.1.1 (192.168.1.1) 0.691 ms 1.637 ms 1.982 ms
 2  10.0.16.250 (10.0.16.250) 7.498 ms  7.625 ms 7.744 ms
 3  salvador.core.ipp.pt (193.136.60.250) 12.569 ms 12.747 ms 12.867 ms
 4  193.136.61.2 (193.136.61.2) 7.720 ms 7.856 ms 7.974 ms
 5  r2isep.wan.ipp.pt (193.136.56.62) 14.483 ms 17.741 ms 23.407 ms
 6  00.wan.ipp.pt (193.136.56.1) 31.001 ms 32.701 ms 30.747 ms
 7  border.wan.ipp.pt (193.136.56.254) 31.673 ms 31.774 ms 31.859 ms
 8  193.136.4.177 (193.136.4.177) 31.073 ms 31.174 ms 31.252 ms
 9  router3.10ge.dwdm.lisboa.fccn.pt (193.136.1.1) 32.247 ms  2.363 ms *
10  router4.10ge.cr2.lisboa.fccn.pt (193.137.0.20) 36.660 ms 37.314 ms 37.800 ms
11  claranet.as8426.gigapix.pt (193.136.251.5) 38.569 ms 39.566 ms 40.703 ms
12  176.111.111.31 (176.111.111.31) 60.652 ms 61.151 ms 61.785 ms
13  80.172.242.108 (80.172.242.108) 64.270 ms 63.101 ms 29.623 ms
```

As a result, in the case of the tests performed with PACI, the network latency was measured (using the Linux command line ping) and deducted from the time response measurements. Figure 7.6 displays the registered network latency.

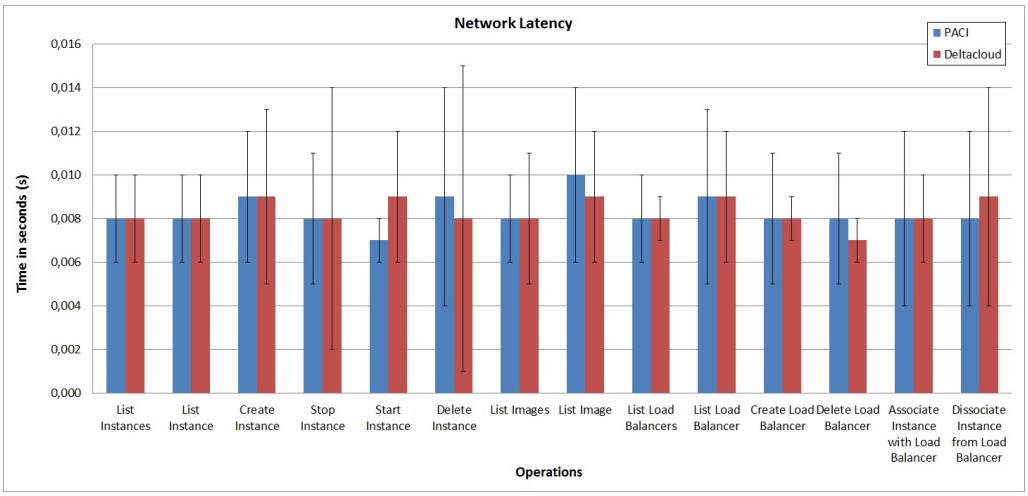


Figure 7.6: PACI IaaS platform API connection latency values.

The list of operations tested in both cases (Deltacloud API and PACI API) was ordered as follows:

1. List Images;
2. Show Image Information;
3. Create Instance;
4. List Instances;
5. Show Instance Information;
6. Start Instance;

7. Create Load Balancer;
8. Associate Instance to Load Balancer;
9. List Load Balancers;
10. Show Load Balancer Information;
11. Dissociate Instance from Load Balancer;
12. Delete Load Balancer;
13. Stop Instance;
14. Delete Instance.

The CentOS-62-x86_64-vm image was selected from the 103 images (OS templates) available for the PACI IaaS platform. The Create Instance operation used a modest hardware profile (1 virtual CPU core, 512 MB of RAM and 10 GB of disk) in order to maintain identical test conditions. The PACI IaaS platform, unlike the deployed OpenNebula, OpenStack and CloudStack nodes, is a public cloud platform configured for high availability and performance and, thus, provides physical resources with a different scale (it uses multiple PACI nodes with clusters of 24 CPU cores and 96 GB RAM).

The results of the time response tests are presented in the charter of Figure 7.7.

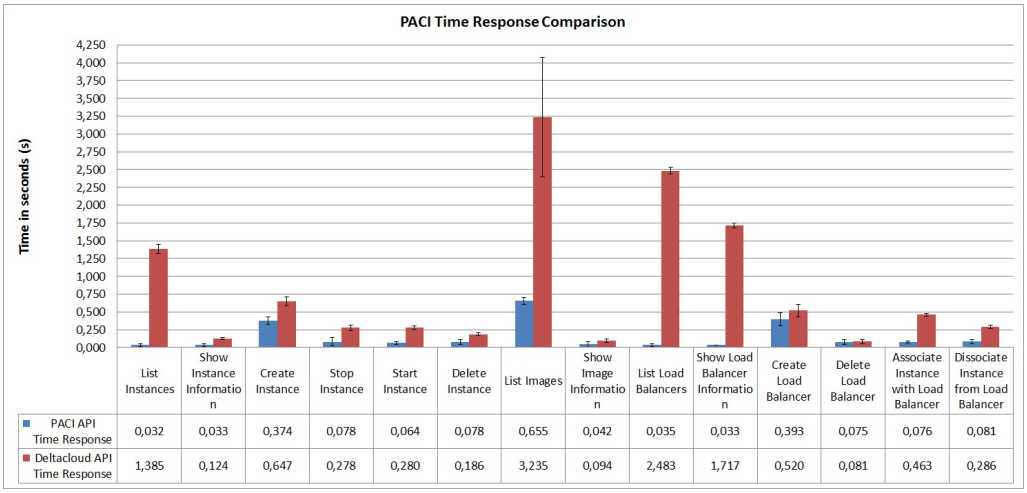
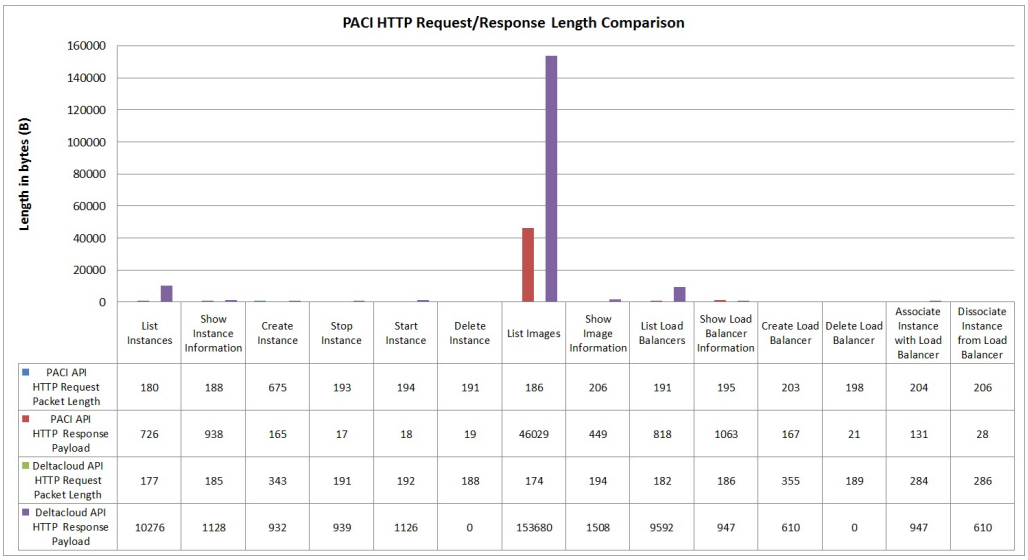


Figure 7.7: PACI time response comparison.

The analysis of the obtained results shows that, despite the registered latency, the time response values of the PACI API operations are lower than the values

registered for the OpenNebula OCCI API and OpenStack API operations, with the exception of the List Images operation. Although, the List Image operation lists 103 Images in comparison with the 10 images that were listed by the same operations of the OpenNebula OCCI API and OpenStack API. In comparison, the results of the interaction with the Deltacloud API, using the PACI driver, show a significantly time response increase, mainly with the List Instances, List Images, List Load Balancers and Show Load Balancers Information operations. The List Images operation presents the highest time response value, since the driver has to process the information of 103 returned images. On the other hand, the time response results for the remaining operations is justified by the necessity to perform additional calls to the back-end PACI API and to process the returned information. The refinement of this methodology may improve the observed time response. Other Deltacloud API operations, *e.g.*, Show Instance Information, Stop Instance, Start Instance, Delete Instance and Show Image Information present better time response results than the corresponding operations via the OpenNebula and OpenStack drivers.

The results of the HTTP request packet length and returned payload for the listed operations in both cases are presented in Figure 7.8.



	List Instances	Show Instance Information	Create Instance	Stop Instance	Start Instance	Delete Instance	List Images	Show Image Information	List Load Balancers	Show Load Balancer Information	Create Load Balancer	Delete Load Balancer	Associate Instance with Load Balancer	Dissociate Instance from Load Balancer
PACI API HTTP Request Packet Length	180	188	675	193	194	191	186	206	191	195	203	198	204	206
PACI API HTTP Response Payload	726	938	165	17	18	19	46029	449	818	1063	167	21	131	28
Deltacloud API HTTP Request Packet Length	177	185	343	191	192	188	174	194	182	186	355	189	284	286
Deltacloud API HTTP Response Payload	10276	1128	932	939	1126	0	153680	1508	9592	947	610	0	947	610

Figure 7.8: PACI HTTP request/response length comparison.

Since the List Images operation presents, in comparison to the remaining operations, a much higher received payload, the chart scale is not appropriate to analyse the overall payload results. The complete set of results is provided in the adjoining table. The length of the HTTP request packets is bigger for the PACI API operations with the exception of the Create Load Balancer, Associate Load Balancer and Dissociate Load Balancer operations. These Deltacloud API op-

erations require more parameters than the corresponding PACI API operations. The Deltacloud API operations return a larger payload than the direct API calls with the exception of the Show Load Balancer Information, Delete Instance and Delete Load Balancer operations. In the case of the last two operations, the Deltacloud API does not send a HTTP response body.

7.3 Conclusions

This chapter presented the interoperable service test bed, including the deployment of all components, the test and evaluation of the driver modules used. The tests involved direct API calls (IaaS platforms API) and calls via the implemented interoperable service (Deltacloud API and driver modules) and consisted of the measurement of the time response and data payload for the implemented operations.

The installation procedures of the OpenNebula, OpenStack and CloudStack IaaS platforms were described. The OpenNebula and CloudStack deployments consist of monolithic software installations, while the OpenStack deployment is a modular installation process (composed by the installation of the Keystone, Glance and Nova projects). Thus, the OpenNebula and CloudStack deployments were simpler in comparison to the OpenStack deployment, which was more complex. The centralized management of the OpenNebula and CloudStack platforms simplifies also the cloud user interaction with the platforms resources, *i.e.*, the users can interact with and manage the available resources via a single API, in contrast with OpenStack that uses an API for each installed project.

From integration of OpenNebula, OpenStack, CloudStack and PACI via driver modules with Deltacloud showed that the OpenNebula and OpenStack drivers, officially included with Deltacloud, were functional and managed to operate with recent released versions of its respective IaaS platforms after minor corrections and improvements. The third-party CloudStack driver however, was incomplete and, therefore, non functional. Thus, the integration of CloudStack IaaS platform with the Deltacloud framework was abandoned. The PACI driver, which was totally developed in order to integrate the PACI IaaS platform with the Deltacloud abstraction framework, was fully functional.

The tests results proved that Deltacloud (and the integrated driver modules) introduces a time response delay, observable when compared to the direct IaaS platforms API calls. The reason for this occurrence is explained by the abstraction processing and the re-transmission of the HTTP request, performed by the included driver modules, to the back-end IaaS platforms API. This is particularly visible with the Deltacloud API operations whose drivers need to perform multiple HTTP calls to the back-end IaaS platform API to obtain the necessary information, *e.g.*, List Images, List Instances, List Load Balancers operations in

general and Show Load Balancer Information in particular. This last operation requests internally a list of available instances to the back-end PACI IaaS platform API in order to show the list of dissociated instances for a specific Load Balancer.

The packet request and response payload results displayed, in the majority of the operations tested, a smaller HTTP request packet length for the Deltacloud API operations. This is explained by the syntax adopted by the Deltacloud API that, in the majority of the operations, uses less parameters than the operations of the dedicated IaaS platforms API. The results of the HTTP response payload were prominently bigger for the Deltacloud API List Instance, List Image, List Load Balancer and List Hardware Profiles operations, meaning that the Deltacloud API exposes more information when listing the available resources.

In the case of the PACI tests, and although the platform was located at the Lunacloud premises, the time response for the direct interaction with the PACI IaaS platform was faster than for the direct interaction with the OpenNebula and OpenStack IaaS platforms. This was also reflected when interacting via the Deltacloud API (and the developed PACI driver), which performed better in terms of time response than via the other drivers. The exception was the List Instances operation that registered the highest time response and payload values, due to the fact that it listed 103 images.

Chapter 8

Conclusions

This chapter presents the conclusions regarding the integrated management of IaaS cloud computing resources selected, installed and tested.

8.1 Discussion

In order to propose and develop an interoperable service for the integrated management of cloud resources provisioned by the IaaS platforms used by Lunacloud, a survey was conducted to characterise and compare the features of the most popular open source IaaS platforms - OpenNebula, OpenStack, CloudStack and Eucalyptus - and the proprietary IaaS platform used by Lunacloud - PACI. This survey concluded that the open-source IaaS platforms expose similar functionalities. Although, between them, the architecture, interface library operations and governance models are significantly different. The proprietary solution does not support directly the interaction with other IaaS platforms and originates, on purpose, the vendor lock-in problem to monetize new products and paid support services. It was also observed that the proprietary IaaS platform has a smaller group of functionalities in comparison to the open source IaaS platforms studied.

Eucalyptus, which is essentially a private IaaS platform to test AWS implementations, was discarded and the client interface API of the OpenNebula, OpenStack, CloudStack and PACI IaaS platforms were studied in detail to identify their common and distinctive features. The conclusion was that there are significant differences regarding the type and number of interfaces, the level of customization, the organization of the groups of operations and the structure of the request/response messages provided by the four IaaS platform interface libraries studied. The identified common management components between the four platforms were analysed according to the number of similar operations and

send/receive parameters in order to propose and develop an interoperable interface solution that meets the main objective of this dissertation.

A second survey on existing IaaS abstraction solutions resulted in the selection and comparison of the Deltacloud framework and the jClouds and Libcloud libraries. The comparison between these abstraction solutions resulted in the selection of the Deltacloud framework since it provides natively more services (Web dashboard, multiple API, a Ruby client application) and allows the integration of new IaaS platforms through the development of new driver modules.

The decision between the development of a new interoperable interface and the adoption of Deltacloud, which included the implementation of a new driver module, resulted in the latter. While the first option is specific and not extensible to other implementations, the Deltacloud abstraction framework already provides many of the desired functionalities and includes several IaaS platform driver modules.

The Deltacloud framework was installed and the officially supported OpenNebula and OpenStack drivers as well a third-party CloudStack driver were reused. The PACI driver did not exist and had to be developed to integrate the PACI IaaS platform with the Deltacloud framework. To test the drivers and the overall interoperable service, a test bed composed of the OpenNebula, OpenStack and CloudStack IaaS platforms was assembled. The PACI IaaS platform provisioned by Lunacloud was adopted.

The test bed was then used to verify the integration and correct operation of the OpenNebula, OpenStack, CloudStack and PACI driver modules. These tests showed that the CloudStack driver module was incomplete and non functional and, for this reason, the integration of the CloudStack platform was dropped. The OpenNebula and OpenStack drivers were functional and managed to operate with recent released versions of its respective IaaS platforms after minor corrections and improvements. The PACI driver, on the other hand, did not manifest any malfunctioning and it was fully functional. A second set of tests was conducted involving the integrated platforms (OpenNebula, OpenStack and PACI) to compare the time response and data payload between direct (via the OpenNebula, OpenStack and PACI API) and Deltacloud (via the OpenNebula, OpenStack and PACI driver modules) operations. The results showed that the use of Deltacloud (and the integrated driver modules) to access the IaaS platform resources introduces an expected time response delay when compared with the direct IaaS platforms API calls. In the majority of the operations with the three platforms, the HTTP request packet length of the Deltacloud API was lower and the results of the HTTP response payload were prominently bigger for the Deltacloud API operations that list multiple resources (List Instance, List Image, List Load Balancer and List Hardware Profiles operations). In general,

the abstraction brought by the Deltacloud framework lowers the HTTP response detail to the fundamental information and reduces, in most cases, the number of required operation parameters. The PACI platform results, despite being located at the Lunacloud premises, for the direct interaction were lower than the results of a direct interaction with the OpenNebula and OpenStack IaaS platforms. Although the PACI driver performed according to the results of the other drivers, it can be refined to enhance the time response of certain operations. Future improvements to the Deltacloud API may also enhance the performance of the included drivers.

The solution adopted for the integrated management and provision of IaaS resources from PACI, OpenNebula and OpenStack IaaS platforms fulfils the specified requirements, *i.e.*, provides Lunacloud with the ability to expand the range of adopted IaaS platforms and offers a Web dashboard and a REST API for the integrated management. The contributions of this work include the surveys and comparisons made, the selection of the abstraction framework, the testbed platform to test the interoperable service proposed and, last but not the least, the PACI driver developed. The PACI driver was shared with the Deltacloud community and the observed OpenNebula and OpenStack driver malfunctions were also reported.

8.2 Envisaged Use Cases

There are two main use cases:

- IaaS resource consumer case - this approach is intended to deliver end users with seamless access to IaaS resources from different IaaS platforms. Furthermore, it empowers the IaaS resource consumer with the ability to manage his resources via a single, common interface without needing to know any of the specificities of the underlying cloud computing IaaS platform API. In such a scenario, an user may contract, consume and free IaaS resources from any IaaS platform with an implemented Deltacloud driver.

- IaaS resource provider case - this solution exposes a single, common interface, allows the integration of multiple IaaS platforms and contributes to increase the number of offered resources and to reduce the vendor lock-in problem. Moreover, if several IaaS providers adopt this approach, federations of resource providers may be established, strengthening the visibility and increasing the market share of the federated providers.

8.3 Future Developments

The PACI driver information processing can be refined by adopting Ruby libraries that parse the information returned from the back-end PACI IaaS platform API faster, *e.g.*, changing the REXML Ruby library by the Nokogiri Ruby library [192]. The performance of the driver could also improve significantly if a newer PACI API version containing more information per operation could be adopted to decrease the number of necessary HTTP requests that have to be executed internally to obtain the desired information.

The addition of new collections and the redefinition of the required API parameters within Deltacloud would increase the range of functionalities provided by the integrated IaaS platforms and resolve some problems, *e.g.*, the need to instantiate mock parameters when using the Create Load Balancer operation to comply with the Deltacloud API requirements.

The development of an additional GUI module to provide end-users with a friendly intuitive interface to manage their resources from different back-end IaaS platforms in a highly abstracted way.

Do a survey on existing standards to develop a resource migration service between the integrated IaaS platforms.

The development of a fully functional CloudStack driver would allow the integration of another IaaS platform to the Deltacloud framework and, consequently, to the proposed interoperable service.

Appendices

Appendix A

Cloud Client Library

```
1
2  require 'rubygems'
3  require 'uri'
4  require 'net/http'
5  require "rexml/document"
6
7  begin
8      require 'rexml/formatters/pretty'
9      REXML_FORMATTERS=true
10 rescue LoadError
11     REXML_FORMATTERS=false
12 end
13
14 begin
15     require 'net/http/post/multipart'
16     MULTIPART_LOADED=true
17 rescue LoadError
18     MULTIPART_LOADED=false
19 end
20
21 ########## The CloudClient module contains general
22 # functionalities for connection and error management
23 ################################################################################
24
25 module CloudClient
26
27     ################################################################################
28     # Starts an http connection and calls the block provided. SSL flag
29     # is set if needed.
30     ################################################################################
31
32     def self.http_start(url, timeout, &block)
33         http = Net::HTTP.new(url.host, url.port)
34         if timeout
35             http.read_timeout = timeout.to_i
36         end
37         if url.scheme=='https'
38             http.use_ssl = true
39             http.verify_mode=OpenSSL::SSL::VERIFY_NONE
40         end
41         begin
42             res = http.start do |connection|
43             block.call(connection)
44         end
```

```
45          rescue Errno::ECONNREFUSED => e
46              str =  "Error connecting to server (#{e.to_s}).\n"
47              str << "Server: #{url.host}:#{url.port}"
48              return CloudClient::Error.new(str,"503")
49          rescue Errno::ETIMEDOUT => e
50              str =  "Error timeout connecting to server (#{e.to_s}).\n"
51              str << "Server: #{url.host}:#{url.port}"
52              return CloudClient::Error.new(str,"504")
53          rescue Timeout::Error => e
54              str =  "Error timeout while connected to server (#{e.to_s}).\n"
55              str << "Server: #{url.host}:#{url.port}"
56              return CloudClient::Error.new(str,"504")
57          end
58          if res.is_a?(Net::HTTPSuccess)
59              res
60          else
61              CloudClient::Error.new(res.body, res.code)
62          end
63      end
64
65      ###########################################################################
66      # The Error Class represents a generic error in the Cloud Client
67      # library. It contains a readable representation of the error.
68      ###########################################################################
69      class Error
70          attr_reader :message
71          attr_reader :code
72
73          # message+ a description of the error
74          def initialize(message=nil, code="500")
75              @message=message
76              @code=code
77          end
78          def to_s()
79              @message
80          end
81      end
82
83      ###########################################################################
84      # Returns true if the object returned by a method of the PACI
85      # library is an Error
86      ###########################################################################
87      def self.is_error?(value)
88          value.class==CloudClient::Error
89      end
90  end
```

Appendix B

PACI Client

```
require 'rubygems'
require 'rexml/document'
require 'uri'
require 'deltacloud/drivers/paci/cloud_client'

module PACIClient

#####################################################################
#  Client Library to interface with the Parallels PACI Service
#####################################################################
class Client

    attr_accessor :endpoint

    ###########################################################################
    # Initialize PACI library
    ###########################################################################
    def initialize(endpoint_str=nil, user=nil, pass=nil,
            timeout=nil, debug_flag=true)
    @debug   = debug_flag
    @timeout = timeout

    # endpoint processement
    # LunaCloud's endpoint is either
    # "http://apicontrol.lunacloud.com:4465/paci/v1.0" or
    # "https://apicontrol.lunacloud.com:4463/paci/v1.0"
    if endpoint_str[-1]=='/'
        @endpoint = endpoint_str.gsub(/\/$/, '')
    else
        @endpoint = endpoint_str
    end

    if !@endpoint || @endpoint==""
        raise "Endpoint URL not configured!
                Client needs to set \'X-Deltacloud-Provider\'
                HTTP request header, or, Deltacloud server
                administrator must set the API_PROVIDER
                environment variable"
    end

    # Autentication
    if user && pass
        @auth = [user, pass]
    # include here call for ENV. Variables auth if
    # necessary (this method needs to be implemented in
```

```
46          # the cloud_client.rb file)
47          end
48
49          if !@auth
50              raise "No authorization data present"
51          end
52          # lunacloud performs SHA-1 on server side, uncomment
53          # the following line in case SHA-1 hash encryption is needed
54          # @auth[1] = Digest::SHA1.hexdigest(@auth[1])
55          end
56
57
58          ##################################
59          # Instance Request Methods       #
60          ##################################
61
62          ########################################################################
63          # Retrieve the list of the available PACI Servers
64          ########################################################################
65          def get_instances()
66          get('/ve')
67          end
68
69          ########################################################################
70          # Retrieve the list of the available PACI Servers by subscription
71          ########################################################################
72          def get_instances(subscription_id.to_s)
73          get('/ve/?subscription='+subscription_id)
74          end
75
76          ########################################################################
77          # Obtain Server Information
78          ########################################################################
79          def get_instance(ve_name)
80          get('/ve/'+ve_name.to_s)
81          end
82
83          ########################################################################
84          # Create a Server
85          ########################################################################
86          def create_instance(xml)
87          post('/ve', xml)
88          end
89
90          ########################################################################
91          # Create a Server from Image
92          ########################################################################
93          def create_instance_from_image(ve_name, image_name)
94          post('/ve/'+ve_name.to_s+'/from/'+image_name.to_s)
95          end
96
97          ########################################################################
98          # Delete Server
99          ########################################################################
100         def delete_instance(ve_name)
101         delete('/ve/'+ ve_name.to_s)
102         end
103
104
105         ########################################################################
106         # Start the Server
107         ########################################################################
108         def start_instance(ve_name)
109         put('/ve/'+ve_name.to_s+'/start')
110         end
111
112
113         ########################################################################
114         # Stop the Server
115         ########################################################################
```

```
116         def stop_instance(ve_name)
117         put('/ve/'+ve_name.to_s+'/stop')
118         end
119
120
121         ##################################
122         # Images Request Methods         #
123         ##################################
124
125         ########################################################################
126         # Create Image from Server
127         ########################################################################
128         def create_image(ve_name, image_name)
129         post('/image/'+ve_name.to_s+'/create/'+image_name.to_s)
130         end
131
132         ########################################################################
133         # Create Image from Server (subscription)
134         ########################################################################
135         def create_image(subscription_id.to_s, ve_name.to_s, image_name.to_s)
136         post('/image/'+ve_name+'/'+subscription_id+'/create/'+image_name)
137         end
138
139         ########################################################################
140         # List Images
141         ########################################################################
142         def get_images()
143         get('/image')
144         end
145
146         ########################################################################
147         # Get Image Information
148         ########################################################################
149         def get_image(image_name)
150         get('/image/'+image_name)
151         end
152
153         ########################################################################
154         # Delete Image
155         ########################################################################
156         def delete_image(image_id)
157         delete('/image/'+image_id)
158         end
159
160         ########################################################################
161         # Get OS Template Information
162         ########################################################################
163         def get_os_template(template_name)
164         get('/template/'+template_name.to_s)
165         end
166
167         ########################################################################
168         # List Installed OS Templates
169         ########################################################################
170         def get_os_templates()
171         get('/template')
172         end
173
174         ##################################
175         # Load Balancer Request Methods #
176         ##################################
177
178         ########################################################################
179         # List Load Balancers
180         ########################################################################
181         def get_load_balancers()
182         get('/load-balancer')
183         end
184
185
```

```
186         ##########################################################################
187         # Get Load Balance Information
188         ##########################################################################
189         def get_load_balancer(lb_name)
190         get('/load-balancer/'+lb_name.to_s)
191         end
192
193         ##########################################################################
194         # Create Load Balancer
195         ##########################################################################
196         def create_load_balancer(lb_name)
197         post('/load-balancer/create/'+lb_name.to_s)
198         end
199
200         ##########################################################################
201         # Create Load Balancer (subscription)
202         ##########################################################################
203         def create_load_balancer(subscription_id, lb_name)
204         post('load-balancer/'+subscription_id.to_s+'/create/'+lb_name.to_s)
205         end
206
207         ##########################################################################
208         # Delete Load Balancer
209         ##########################################################################
210         def delete_load_balancer(lb_name)
211         delete('/load-balancer/'+lb_name.to_s)
212         end
213
214         ##########################################################################
215         # Attache Server To Load Balancer
216         ##########################################################################
217         def register_instance(lb_name, ve_name)
218         post('/load-balancer/'+lb_name.to_s+'/'+ve_name.to_s)
219         end
220
221         ##########################################################################
222         # Detache Server From Load Balancer
223         ##########################################################################
224         def unregister_instance(lb_name, ve_name)
225         delete('/load-balancer/'+lb_name.to_s+'/'+ve_name.to_s)
226         end
227
228
229         private
230
231         def get(path)
232         url = URI.parse(@endpoint+path)
233         req = Net::HTTP::Get.new(url.path)
234         do_request(url, req)
235         end
236
237         def post(path, xml=nil)
238         url = URI.parse(@endpoint+path)
239         req = Net::HTTP::Post.new(url.path)
240         if !xml.nil?
241             req.body=xml.to_s
242             req.content_type= 'application/xml'
243         end
244         do_request(url, req)
245         end
246
247         def delete(path, xml=nil)
248
249         url = URI.parse(@endpoint+path)
250         req = Net::HTTP::Delete.new(url.path)
251         if !xml.nil?
252             req.body=xml.to_s
253             req.content_type= 'application/xml'
254         end
255         do_request(url, req)
```

```
256         end
257
258         def put(path, xml=nil)
259         url = URI.parse(@endpoint+path)
260         req = Net::HTTP::Put.new(url.path)
261         if !xml.nil?
262             req.body=xml.to_s
263             req.content_type= 'application/xml'
264         end
265         do_request(url, req)
266         end
267
268         def do_request(url, req)
269         req.basic_auth @auth[0], @auth[1]
270
271         res = CloudClient::http_start(url, @timeout) do |http|
272             http.request(req)
273         end
274
275         if CloudClient::is_error?(res)
276             return res
277         else
278             return res.body
279         end
280         end
281       end
282     end
```

Appendix C

PACI Driver

```
require 'deltacloud/drivers/paci/paci_client'
require 'erb'
require 'rexml/document'

module Deltacloud
 module Drivers
  module Paci

class PaciDriver < Deltacloud::BaseDriver

feature :instances, :user_name
feature :instances, :authentication_password
feature :images,    :user_name

DEFAULT_REGION = 'EU West'

########################
# Hardware Profiles (LunaCloud)
########################################

define_hardware_profile('PointFive') do
    cpu          (1 .. 1)
    memory       (512 .. 512)
    storage          (10 .. 10)
    #architecture    ['x86_64','i386']
end

define_hardware_profile('One') do
    cpu          (1 .. 1)
    memory       (1024 .. 1024)
    storage          (50 .. 50)
    #architecture    ['x86_64','i386']
end

define_hardware_profile('Two') do
    cpu          (2 .. 2)
    memory       (2048 .. 2048)
    storage          (100 .. 100)
    #architecture    ['x86_64','i386']
end

define_hardware_profile('Four') do
    cpu          (2 .. 2)
    memory       (4096 .. 4096)
    storage          (250 .. 250)
```

```
46          #architecture    ['x86_64','i386']
47      end
48
49      define_hardware_profile('Eight') do
50          cpu             (2 .. 2)
51          memory          (8192 .. 8192)
52          storage             (500 .. 500)
53          #architecture    ['x86_64','i386']
54      end
55
56      define_hardware_profile('OneSix') do
57          cpu             (4 .. 4)
58          memory          (16384 .. 16384)
59          storage             (1000 .. 1000)
60          #architecture    ['x86_64','i386']
61      end
62
63      define_hardware_profile('Custom') do
64          cpu             (1 .. 8)
65          memory          (512 .. 192*512)
66          storage                 (10 .. 2000)
67      end
68
69      ########################
70      # Realms
71      ########################
72
73      (REALMS = [ Realm.new({
74                          :id     => 'EU_West',
75                          :name   => 'EU_West',
76                          :limit  => 'Unknown',
77                          :state  => 'AVAILABLE'
78                  }),
79                  Realm.new({
80                          :id     => 'EU_Central',
81                          :name   => 'EU_Central',
82                          :limit  => 'Unknown',
83                          :state  => 'AVAILABLE'
84                  })
85      ]) unless defined?( REALMS )
86
87      def realms(credentials, opts={})
88          return REALMS if ( opts.nil? )
89          results = REALMS
90          # results = filter_on( results, :id, opts )
91          # results
92      end
93
94      ########################
95      # Instances
96      ########################
97
98
99      # cpu power is always 1600 MHz, default bandwidth  set
100     # to 10240 (min value) but it can go to 102400.
101     # bandwidth, no-of-public-ip, and no-of-public-ipv6
102     # are static because no deltacloud instance features
103     # are available for these parameters.
104
105     # VM XML Template:
106     VE_TEMPLATE = %q{
107     <ve>
108         <name><%=ve_name%></name>
109         <cpu number="<%=opts[:hwp_cpu]%>" power="1600"/>
110         <ram-size><%=opts[:hwp_memory].to_i%></ram-size>
111         <bandwidth>10240</bandwidth>
112         <no-of-public-ip>1</no-of-public-ip>
113         <no-of-public-ipv6>0</no-of-public-ipv6>
114         <ve-disk local="true" size="<%=opts[:hwp_storage]%>"/>
115         <platform>
```

```
116         <template-info name="<%=image_id%>"/>
117         <os-info technology="<%=tech%>" type="<%=type%>"/>
118         </platform>
119         <backup-schedule name="weekly"/>
120         <% if opts[:password] %>
121         <admin login="root" password="<%=opts[:password]%>"/>
122         <% else %>
123         <admin login="root" />
124         <% end %>
125       </ve>
126       }
127
128       VE_STATES = {
129           "CREATE"                  => "START",
130           "CREATION_IN_PROGRESS"    => "PENDING",
131           "CREATED"                 => "STOPPED",
132           "START_IN_PROGRESS"       => "RUNNING",
133           "STARTED"                 => "RUNNING",
134           "STOP_IN_PROGRESS"        => "STOPPING",
135           "STOPPED"                 => "STOPPED",
136           "DELETE_IN_PROGRESS"      => "STOPPING",
137           "DELETED"                 => "FINISHED"
138       }
139
140       define_instance_states do
141           start.to(:pending)          .on( :create )
142           pending.to(:stopped)        .automatically
143           stopped.to(:pending)        .on( :destroy )
144           pending.to(:finish)         .automatically
145           stopped.to(:running)        .on( :start )
146           running.to(:stopping)       .on( :stop )
147           stopping.to(:stopped)       .automatically
148           running.to(:stopping)       .on( :destroy )
149           stopping.to(:finish)        .automatically
150       end
151
152
153       def instance(credentials, opts={})
154           paci_client = new_client(credentials)
155           ve_name=opts[:id]
156           xml = treat_response( paci_client.get_instance(ve_name) )
157           convert_instance(xml, credentials)
158       end
159
160       def instances(credentials, opts={})
161           paci_client = new_client(credentials)
162           xml = treat_response( paci_client.get_instances() )
163           instances = REXML::Document.new(xml).root.elements.map do |d|
164               convert_instances(d, credentials)
165           end
166       end
167
168       def create_instance(credentials, image_id, opts={})
169           paci_client = new_client(credentials)
170
171           # Processing name
172           if opts[:name] && opts[:name].length>0
173           ve_name= opts[:name]
174           else
175           time=Time.now.to_s
176           time=time.split('+').first
177           time=time.gsub(/\D/,'')
178           ve_name= "Server-"+time
179           end
180
181           # Mapping OS template info
182           img_xml = treat_response( paci_client.get_os_template(image_id) )
183           buffer = REXML::Document.new(img_xml.to_s).root.attributes
184           tech=buffer['technology']
185           type=buffer['osType']
```

```
186
187         # Building VE POST XML body
188         req_xml = ERB.new(VE_TEMPLATE).result(binding)
189
190         # Send/Receive (no need for a variable because
191         # the returned message is not used by Deltacloud)
192         treat_response( paci_client.create_instance(req_xml) )
193
194         # Show Instance XML (Deltacloud XML)
195         instance(credentials, id: ve_name)
196       end
197
198       def start_instance(credentials, id)
199         paci_client = new_client(credentials)
200         treat_response( paci_client.start_instance(id) )
201         ve_xml = treat_response( paci_client.get_instance(id) )
202         convert_instance(ve_xml, credentials)
203       end
204
205       def stop_instance(credentials, id)
206         paci_client = new_client(credentials)
207         treat_response( paci_client.stop_instance(id) )
208         ve_xml = treat_response( paci_client.get_instance(id) )
209         convert_instance(ve_xml, credentials)
210       end
211
212       def destroy_instance(credentials, id)
213         paci_client = new_client(credentials)
214         treat_response( paci_client.delete_instance(id) )
215         # 204 HTTP code returned
216       end
217
218       ########################
219       # Images
220       #
221       #
222       ########################
223       def image(credentials, opts={})
224         paci_client = new_client(credentials)
225         xml = treat_response( paci_client.get_os_template(opts[:id]) )
226         convert_image(xml, credentials)
227       end
228
229       def images(credentials, opts={})
230         paci_client = new_client(credentials)
231         xml = treat_response( paci_client.get_os_templates() )
232         images = REXML::Document.new(xml).root.elements.map do |d|
233             convert_image(d, credentials)
234         end
235       end
236
237       # Since the OS Templates are owned by the cloud provider
238       # the user can not delete them.
239       # def destroy_image(credentials, id)
240       #   paci_client = new_client(credentials)
241       #   treat_response( paci_client.delete_image(id) )
242       # end
243
244       ########################
245       # Load Balancers
246       ########################
247
248       def load_balancer(credentials, opts={})
249         paci_client = new_client(credentials)
250         xml=treat_response( paci_client.get_load_balancer(opts[:id]) )
251         convert_load_balancer(xml, credentials)
252       end
253
254       def load_balancers(credentials, opts={})
255         paci_client = new_client(credentials)
```

```
256         xml=treat_response( paci_client.get_load_balancers() )
257         load_balancers=REXML::Document.new(xml).root.elements.map do |d|
258             convert_load_balancers(d, credentials)
259         end
260     end
261
262     def create_load_balancer(credentials, opts={})
263         paci_client = new_client(credentials)
264         lb_name= opts['name']
265         if lb_name.nil?
266         time=Time.now.to_s
267         time=time.split('+').first
268         time=time.gsub(/\D/,'')
269         lb_name= "LB-"+time
270         end
271         treat_response( paci_client.create_load_balancer(lb_name) )
272         load_balancer(credentials, id: lb_name )
273
274     end
275
276     def destroy_load_balancer(credentials, id)
277         paci_client = new_client(credentials)
278         treat_response( paci_client.delete_load_balancer(id) )
279         # 204 HTTP code returned
280     end
281
282     def lb_register_instance(credentials, opts={})
283         paci_client = new_client(credentials)
284         lb_name= opts[:id]
285         instance_name= opts['instance_id']
286         treat_response( paci_client.register_instance(lb_name, instance_name) )
287         load_balancer(credentials, id: lb_name)
288     end
289
290     def lb_unregister_instance(credentials, opts={})
291         paci_client = new_client(credentials)
292         lb_name= opts[:id]
293         instance_name= opts['instance_id']
294         treat_response( paci_client.unregister_instance(lb_name, instance_name) )
295         load_balancer(credentials, id: lb_name)
296     end
297
298
299     private
300
301     def new_client(credentials)
302         PACIClient::Client.new(api_provider, credentials.user, credentials.password)
303     end
304
305     ########################
306     #   Mapping Parameters
307     ########################
308
309     # This method mapps only OS Templates
310     def convert_image(xml, credentials)
311         buffer = REXML::Document.new(xml.to_s).root
312         if buffer.attributes['active']=="false"
313             default_state = "DISABLED"
314         else
315             default_state = "ACTIVE"
316         end
317
318         #mapping
319         Image.new({
320
321             :id=>buffer.attributes['name'],
322         :name=>buffer.attributes['name'],
323         :description=>"OS: "+
324            buffer.elements[2].attributes['value']+
325            ", Virtualization type (VM/CT): "+
```

```
326             buffer.attributes['technology'],
327         :owner_id=>"LunaCloud",
328         :state=>default_state,
329         :architecture=>buffer.elements[1].attributes['value'],
330         :hardware_profiles=>hardware_profiles(nil)
331         })
332     end
333
334     # Convert PACI VM parameters to Deltacloud
335     def convert_instance(xml, credentials)
336         buffer = REXML::Document.new(xml.to_s).root.elements
337
338         if buffer['cpu'].attributes['number']=="1" &&
339             buffer['ram-size'].text=="512" &&
340             buffer['ve-disk'].attributes['size']=="10"
341             instance_profile='PointFive'
342         elsif buffer['cpu'].attributes['number']=="1" &&
343               buffer['ram-size'].text=="1024" &&
344               buffer['ve-disk'].attributes['size']=="50"
345             instance_profile='One'
346         elsif buffer['cpu'].attributes['number']=="2" &&
347               buffer['ram-size'].text=="2048" &&
348               buffer['ve-disk'].attributes['size']=="100"
349         instance_profile='Two'
350         elsif buffer['cpu'].attributes['number']=="2" &&
351               buffer['ram-size'].text=="4096" &&
352               buffer['ve-disk'].attributes['size']=="250"
353             instance_profile='Four'
354         elsif buffer['cpu'].attributes['number']=="2" &&
355               buffer['ram-size'].text=="8192" &&
356               buffer['ve-disk'].attributes['size']=="500"
357             instance_profile='Eight'
358         elsif buffer['cpu'].attributes['number']=="4" &&
359              buffer['ram-size'].text=="16384" &&
360              buffer['ve-disk'].attributes['size']=="1000"
361             instance_profile='OneSix'
362         else
363             instance_profile='Custom'
364         end
365
366         private_ip=[]
367         public_ip=[]
368
369         if buffer['network/public-ip']
370             buffer.each('network/public-ip') { |ip| public_ip
371             << InstanceAddress.new(ip.attributes['address'].split('/').first,
372              :type => :ipv4)}
373         end
374
375         if buffer['network/public-ipv6']
376         buffer.each('network/public-ipv6') { |ip| public_ip
377         << InstanceAddress.new(ip.attributes['address'].split('/').first,
378          :type => :ipv6)}
379         end
380
381         private_ip
382         << InstanceAddress.new(buffer['network'].attributes['private-ip'].split('/').first,
383          :type => :ipv4)
384
385         Instance.new( {
386         :id=>buffer['name'].text,
387         :owner_id=>buffer['subscription-id'].text,
388         :name=>buffer['name'].text,
389         :image_id=>buffer['platform'].elements['template-info'].attributes['name'],
390         :instance_profile=>InstanceProfile.new(instance_profile),
391         :realm_id=>DEFAULT_REGION,
392         :state=>VE_STATES[buffer['state'].text],
393         :public_addresses=>public_ip,
394         :private_addresses=>private_ip,
395         :username=>buffer['admin'].attributes['login'],
```

```
396         :password=>buffer['admin'].attributes['password'],
397         :actions=> instance_actions_for( VE_STATES[buffer['state'].text] ),
398         :storage_volumes=>[],
399         :launch_time=> "Unknown"
400         } )
401
402     end
403
404     # hack to provide vm lists with more details (uses the returned vm_list
405     # XML to query each vm at a time reusing convert_instance method
406     def convert_instances(xml, credentials)
407         paci_client = new_client(credentials)
408         ve_name=REXML::Document.new(xml.to_s).root.attributes['name']
409         vexml=treat_response( paci_client.get_instance(ve_name) )
410         convert_instance(vexml, credentials)
411     end
412
413     ########
414     # Mapping Load Balancers
415     #######
416
417     def convert_load_balancer(xml, credentials)
418         buffer = REXML::Document.new(xml.to_s).root.elements
419         addresses=[]
420         if buffer['network/public-ip']
421             buffer.each('network/public-ip') {|ip| addresses
422             << InstanceAddress.new(ip.attributes['address'].split('/').first,
423              :type=> :ipv4)}
424         end
425
426         if buffer['network/public-ipv6']
427             buffer.each('network/public-ipv6') {|ip| addresses
428             << InstanceAddress.new(ip.attributes['address'].split('/').first,
429              :type=> :ipv6)}
430         end
431
432         balancer=LoadBalancer.new({
433         :id=> buffer['name'].text,
434         :created_at=> "Unknown",
435         :public_addresses=> addresses,
436         :realms=> REALMS,
437         })
438
439         balancer.listeners = []
440
441         buffer.each('used-by') do |vm|
442             balancer.add_listener({
443             :protocol => 'HTTP',
444             :load_balancer_port => 'Unknown',
445             :instance_port => 'Unknown'
446             })
447         end
448
449         balancer.instances = []
450
451         buffer.each('used-by') do |vm|
452             balancer.instances << instance(credentials, id: vm.attributes['ve-name'])
453         end
454         balancer
455
456     end
457
458     def convert_load_balancers(xml, credentials)
459         paci_client = new_client(credentials)
460         lb_name=REXML::Document.new(xml.to_s).root.attributes['name']
461         lbxml=treat_response(paci_client.get_load_balancer( lb_name) )
462         convert_load_balancer(lbxml, credentials)
463     end
464
465     ########
```

```
466     # Errors and returned messages process
467     #######
468
469
470     # function to process returned messages
471     def treat_response(res)
472         safely do
473         if CloudClient.is_error?(res)
474             raise case res.code
475                 when "401" then "AuthenticationFailure"
476                 when "404" then "ObjectNotFound"
477                 else res.message
478                 end
479         end
480         end
481         res
482     end
483
484     # error treatment seen by Deltacloud
485     exceptions do
486         on /AuthenticationFailure/ do
487         status 401
488         end
489
490         on /ObjectNotFound/ do
491         status 404
492         end
493
494         on // do
495         status 502
496         end
497     end
498     end
499     end
500     end
501     end
```

Bibliography

[1] Amazon. (2013) Amazon Web Services. [Online]. Available: http://aws.amazon.com [cited in p. 1, 16]

[2] Google. (2013) Google App Engine. [Online]. Available: https://developers.google.com/appengine/ [cited in p. 1, 17]

[3] Spotify. About Us. [Online]. Available: https://www.spotify.com/pt/about\T1\textendashus/contact/ [cited in p. 1]

[4] Edstrom, D. (2012, nov) The Network Is The Computer. [Online]. Available: http://www.imts.com/show/newsletter/insider/article.cfm?aid=547 [cited in p. 1]

[5] Furht, B. and Escalante, A., *Handbook of Cloud Computing*. Springer US, 2010. [cited in p. 7, 8]

[6] Foster, I. and Kesselman, C., *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers, 1999. [cited in p. 7]

[7] Chetty, M. and Buyya, R., "Weaving computational grids: how analogous are they with electrical grids?" *Computing in Science Engineering*, vol. 4, no. 4, pp. 61–71, 2002. [cited in p. 7]

[8] Foster, I et al., "Cloud Computing and Grid Computing 360-Degree Compared," in *Grid Computing Environments Workshop, 2008. GCE '08*, 2008, pp. 1–10. [cited in p. 8, 9]

[9] Stanoevska-Slabeva, K. et al., *Grid and Cloud Computing*. Springer Berlin Heidelberg, 2010. [cited in p. 8, 31]

[10] Patidar, S. et al., "A Survey Paper on Cloud Computing," in *Advanced Computing Communication Technologies (ACCT), 2012 Second International Conference on*, 2012, pp. 394–398. [cited in p. 8]

[11] Vaquero, Luis M. et al., "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, Dec 2008. [cited in p. 8]

[12] Cloud Expo. (2008) Twenty-One Experts Define Cloud Computing. [Online]. Available: http://cloudcomputing.sys-con.com/node/612375 [cited in p. 8]

[13] Gartner, Inc. (NYSE: IT). (2013). [Online]. Available: http://www.gartner.com/technology/home.jsp [cited in p. 8]

[14] Bank of America Corporation, Merrill Lynch & Co., Inc. (2013). [Online]. Available: http://www.ml.com/index.asp?id=7695_15125 [cited in p. 8]

[15] Gartner. (2008, September) Gartner Says Contrasting Views on Cloud Computing Are Creating Confusion, Gartner press release. [Online]. Available: http://www.gartner.com/newsroom/id/766215 [cited in p. 8]

[16] Rangan, K. et al., "The cloud wars: $100 + billion at stake," Merrill Lynch, Technical report, 2008. [cited in p. 8]

[17] Armbrust et al., "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep., Feb 2009. [cited in p. 9]

[18] Buyya, R. et al., "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," in *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, 2008, pp. 5–13. [cited in p. 9]

[19] Mell, P. , Grace, T., "The NIST Definition of Cloud Computing," National Institute of Standards and Technology (NIST), Tech. Rep., Sep 2011. [cited in p. 9, 10, 11, 17, 18]

[20] Fingar, P. (2010, April) A BPTrends column: Enterprise as a Service (EaaS)-That's where BPM Comes In. [Online]. Available: http://bptrends.com/publicationfiles/EIGHT%2004-10-COL-EXT%20COMPETITION-Enterprise%20as%20Svc-Fingar-final1.pdf [cited in p. 9]

[21] Daconta, M. (2012, March) GCN commentary: Why NIST's cloud definition is fatally flawed. [Online]. Available: http://gcn.com/Articles/2012/04/02/Reality-Check-NIST-flawed-cloud-framework.aspx [cited in p. 9]

[22] YUNG CHOU. (2012, January) Cloud Expo Article: An Inconvenient Truth of the NIST Definition of Cloud Computing. [Online]. Available: http://cloudcomputing.sys-con.com/node/2131995 [cited in p. 9]

[23] NIST, Information Technology Laboratory. (2011, October) Final Version of NIST Cloud Computing Definition Published. [Online]. Available: http://www.nist.gov/itl/csd/cloud-102511.cfm [cited in p. 9]

[24] Liu, F. et al., "NIST Cloud Computing Reference Architecture," National Institute of Standards and Technology (NIST), Tech. Rep., Sep 2011. [cited in p. 10, 11, 12, 13, 14, 15, 16]

[25] Buyya, R. et al., *Cloud Computing: Principles and Paradigms.* Wiley, 2011. [cited in p. 16, 36, 37]

[26] LunaCloud. (2013) Lunacloud Server. [Online]. Available: http://www.lunacloud.com/en/cloud-server [cited in p. 16]

[27] Google. (2013) Google Apps for Business. [Online]. Available: http://www.google.com/enterprise/apps/business/ [cited in p. 17]

[28] Salesforce. (2013) Sales-cloud. [Online]. Available: http://www.salesforce.com/sales-cloud/overview/ [cited in p. 17]

[29] Velte, T. et al., *Cloud Computing, A Practical Approach.* Mcgraw-hill, 2009. [cited in p. 17]

[30] Gong, C. et al., "The Characteristics of Cloud Computing," in *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, 2010, pp. 275–279. [cited in p. 17]

[31] Jadeja, Y. and Modi, K., "Cloud computing - concepts, architecture and challenges," in *Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on*, 2012, pp. 877–880. [cited in p. 17]

[32] Hashizume, K. et al., "An analysis of security issues for cloud computing," *Journal of Internet Services and Applications*, vol. 4, no. 1, pp. 1–13, 2013. [Online]. Available: http://dx.doi.org/10.1186/1869-0238-4-5 [cited in p. 20, 21]

[33] Li, Wenjuan and Ping, Lingdi, "Trust Model to Enhance Security and Interoperability of Cloud Environment," in *Cloud Computing*, ser. Lecture Notes in Computer Science, Jaatun, MartinGilje and Zhao, Gansen and Rong, Chunming, Ed. Springer Berlin Heidelberg, 2009, vol. 5931, pp. 69–79. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10665-1_7 [cited in p. 20]

[34] Madrid International Conference. (2009, November) 31st International Conference of Data Protection and Privacy. [Online]. Available: http://www.privacyconference2009.org/home/index-iden-idweb.html [cited in p. 21]

[35] IBM Systems Magazine. (2012, December) 40 Years of VM Community—and Going Strong! [Online]. Available: http://www.ibmsystemsmag.com/mainframe/trends/z-vm/vm_community/ [cited in p. 22]

[36] Oracle. (2011) VM User's Guide: Brief History of Virtualization. [Online]. Available: http://docs.oracle.com/cd/E20065_01/doc.30/e18549/intro.htm [cited in p. 22]

[37] Antonopoulos, N. and Gillam, L., *Cloud Computing: Principles, Systems and Applications*. Springer London, 2010. [cited in p. 22, 24, 25, 35]

[38] Popek, Gerald J. and Goldberg, Robert P., "Formal requirements for virtualizable third generation architectures," *Commun. ACM*, vol. 17, no. 7, pp. 412–421, jul 1974. [cited in p. 22]

[39] IBM. (2005, December) IBM Systems Software Information Center: Virtual systems overview. [Online]. Available: http://publib.boulder.ibm.com/infocenter/eserver/v1r2/index.jsp?topic=/eicay/eicayvservers.htm [cited in p. 23]

[40] Creasy, R. J., "The Origin of the VM/370 Time-Sharing System," *IBM Journal of Research and Development*, vol. 25, no. 5, pp. 483–490, 1981. [cited in p. 23]

[41] Intel. Intel Virtualization Technology List: About Intel Virtualization Technology. [Online]. Available: http://ark.intel.com/products/virtualizationtechnology [cited in p. 24]

[42] AMD. AMD Virtualization. [Online]. Available: http://www.amd.com/us/solutions/servers/virtualization/Pages/virtualization.aspx#2 [cited in p. 24]

[43] VMware. virtualization software company. [Online]. Available: http://www.vmware.com/#eu [cited in p. 24, 40, 42, 47, 51, 58]

[44] KVM. virtualization solution. [Online]. Available: http://www.linux-kvm.org/page/Main_Page [cited in p. 24, 40, 47, 51, 58]

[45] Xen Server. Open Source Virtualization. [Online]. Available: http://www.xenserver.org/ [cited in p. 25, 40, 47, 58]

[46] OpenVZ. container-based virtualization for Linux. [Online]. Available: http://openvz.org/Main_Page [cited in p. 26]

[47] Parallels. Parallels Virtuozzo Containers. [Online]. Available: http://www.parallels.com/products/pvc/ [cited in p. 26]

[48] Marston, Sean et al., "Cloud computing - The business perspective," *Decis. Support Syst.*, vol. 51, no. 1, pp. 176–189, apr 2011. [cited in p. 26, 28]

[49] "Cloud computing —business models, value creation dynamics and advantages for customers," White Paper, Siemens, 2010. [cited in p. 27]

[50] Microsoft. [Online]. Available: http://www.microsoft.com/ [cited in p. 28]

[51] Google. About Google. [Online]. Available: https://www.google.pt/intl/en\T1\textendashUS/about/ [cited in p. 28]

[52] Facebook. [Online]. Available: https://www.facebook.com/facebook [cited in p. 28]

[53] RightScale. [Online]. Available: http://www.rightscale.com/ [cited in p. 29]

[54] Vordel. [Online]. Available: http://www.vordel.com/ [cited in p. 29]

[55] Wieder, Philipp et al., *Service Level Agreements for Cloud Computing*. Springer, 2011. [cited in p. 30, 31]

[56] Gangadharan, G.R. and Parrilli, DavideMaria, "Service Level Agreements in Cloud Computing: Perspectives of Private Consumers and Small-to-Medium Enterprises," in *Cloud Computing for Enterprise Architectures*. Springer London, 2011, pp. 207–225. [cited in p. 31]

[57] ETSI, "SLAs for Cloud services," European Telecommunications Standards Institute (ETSI), Tech. Rep., Nov 2012. [Online]. Available: http://www.etsi.org/deliver/etsi_tr/103100_103199/103125/01.01.01_60/tr_103125v010101p.pdf [cited in p. 31]

[58] Andrieux, A. et al., "Web Services Agreement Specification (WS-Agreement)," Open Grid Forum (OGF), Tech. Rep., Oct 2006. [Online]. Available: http://redmin.ogf.org/Public_Comment_Docs/Documents/Oct-2006/WS-AgreementSpecificationDraftFinal_sp_tn_jpver_v2.pdf [cited in p. 31, 32]

[59] Ludwig, H. et al., "Web service level agreement (WSLA) language specification," *IBM Corporation*, pp. 815–824, 2003. [cited in p. 31]

[60] Amazon. Amazon Web Services: Amazon EC2. [Online]. Available: http://aws.amazon.com/ec2/ [cited in p. 36, 147]

[61] Sotomayor, B. and Montero, Ruben S. and Llorente, I.M. and Foster, I., "Virtual Infrastructure Management in Private and Hybrid Clouds," *Internet Computing, IEEE*, vol. 13, no. 5, pp. 14–22, Sept 2009. [cited in p. 36]

[62] C12G Labs. OPEN—SOURCE ENTERPRISE CLOUD SIMPLIFIED. [Online]. Available: http://opennebula.org [cited in p. 36]

[63] OpenStack Foundation. Open source software for building private and public clouds. [Online]. Available: http://www.openstack.org [cited in p. 36, 42]

[64] Apache Software Foundation. Apache CloudStack: Open Source Cloud Computing. [Online]. Available: http://cloudstack.apache.org/index.html [cited in p. 36]

[65] Eucalyptus Systems Inc. Eucalyptus: Open Source Private Cloud Software. [Online]. Available: https://www.eucalyptus.com/eucalyptus-cloud/iaas [cited in p. 36, 54]

[66] Parallels. Parallels Automation for Cloud Infrastructure. [Online]. Available: http://www.parallels.com/products/paci/ [cited in p. 36]

[67] Distributed Systems Architecture (DSA). Research and Innovation in Cloud Computing. [Online]. Available: http://dsa-research.org/doku.php [cited in p. 36]

[68] Toraldo, G., *OpenNebula 3 Cloud Computing*, ser. Community experience distilled. Packt Publishing„ 2012. [cited in p. 37, 40]

[69] C12G Labs. [Online]. Available: http://c12g.com/ [cited in p. 37]

[70] SQLite. About SQLite. [Online]. Available: http://www.sqlite.org/about.html [cited in p. 37, 40, 44, 47]

[71] MySQL. The world's most popular open source database. [Online]. Available: http://www.mysql.com/ [cited in p. 37, 40, 44, 47, 52]

[72] OpenNebula Community wiki. OpenNebula Ecosystem. [Online]. Available: http://community.opennebula.org/ecosystem [cited in p. 38]

[73] OpenNebula Archive. Scalable Architecture and APIs 4.0. [Online]. Available: http://archives.opennebula.org/documentation:archives:rel4.0:introapis [cited in p. 39]

[74] Amazon Elastic Compute Cloud. API Reference. [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/APIReference/Welcome.html [cited in p. 39]

[75] Amazon Web Services. Amazon EC2 API Tools. [Online]. Available: http://aws.amazon.com/developertools/Amazon-EC2/351 [cited in p. 39]

[76] Open Grid Forum (OGF). Open Forum - Open Standards. [Online]. Available: https://www.ogf.org/dokuwiki/doku.php [cited in p. 39]

[77] OGF. OCCI: About. [Online]. Available: http://occi-wg.org/about/ [cited in p. 39]

[78] OpenNebula Archive. OpenNebula OCCI User Guide 4.0. [Online]. Available: http://archives.opennebula.org/documentation:archives:rel4.0:occiug [cited in p. 39]

[79] IEEE, "IEEE Standard for Local and metropolitan area networks—Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks," *IEEE Std 802.1Q-2011 (Revision of IEEE Std 802.1Q-2005)*, pp. 1–1365, Aug 2011. [cited in p. 41]

[80] Open vSwitch. An Open Virtual Switch. [Online]. Available: http://openvswitch.org/ [cited in p. 41, 44]

[81] Housley, R. et al., "Internet X.509 Public Key Infrastructure Certificate and CRL Profile," January 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2459.txt [cited in p. 42]

[82] THE RACKSPACE BLOG & NEWSROOM. Opening The Rackspace Cloud. [Online]. Available: http://www.rackspace.com/blog/opening-the-rackspace-cloud/ [cited in p. 42]

[83] NASA. NASA Nebula Cloud Computing Platform. [Online]. Available: http://www.nasa.gov/open/plan/nebula.html [cited in p. 42]

[84] OpenStack. The OpenStack Open Source Cloud Mission. [Online]. Available: https://wiki.openstack.org/wiki/Main_Page [cited in p. 42]

[85] OpenStack Compute Administration Guide. Grizzly, Conceptual Architecture. [Online]. Available: http://docs.openstack.org/grizzly/openstack-compute/admin/content/conceptual-architecture.html [cited in p. 43]

[86] AWS. Amazon S3. [Online]. Available: https://aws.amazon.com/s3/ [cited in p. 44, 55, 145, 147]

[87] Ceph. Block Storage. [Online]. Available: http://ceph.com/ceph-storage/block-storage/ [cited in p. 44]

[88] Mongo DB. GridFS. [Online]. Available: http://docs.mongodb.org/manual/core/gridfs/ [cited in p. 44]

[89] Mongo DB. Agile and Scalable. [Online]. Available: http://www.mongodb.org/ [cited in p. 44]

[90] Open Networking Foundation. OpenFlow. [Online]. Available: https://www.opennetworking.org/sdn-resources/onf-specifications/openflow [cited in p. 44]

[91] Cisco. [Online]. Available: http://www.cisco.com/ [cited in p. 44]

[92] AWS. Amazon EBS. [Online]. Available: http://aws.amazon.com/ebs/ [cited in p. 45, 56]

[93] OpenStack Compute Administration Guide. Grizzly, Logical Architecture. [Online]. Available: http://docs.openstack.org/grizzly/openstack-compute/admin/content/logical-architecture.html [cited in p. 46]

[94] OpenStack. OCCI. [Online]. Available: https://wiki.openstack.org/wiki/Occi#Summary [cited in p. 46]

[95] OpenStack. Configuring Object Storage with the S3 API. [Online]. Available: http://docs.openstack.org/grizzly/openstack-object-storage/admin/content/configuring-openstack-object-storage-with-s3_api.html [cited in p. 47]

[96] RabbitMQ. Messaging that just works. [Online]. Available: http://www.rabbitmq.com/ [cited in p. 47]

[97] Apache. Apache Qpid. [Online]. Available: http://qpid.apache.org/ [cited in p. 47]

[98] OpenStack. Block Storage Service Administration Guide. [Online]. Available: http://docs.openstack.org/grizzly/openstack-block-storage/admin/content/index.html [cited in p. 47]

[99] Microsoft. Hyper—V. [Online]. Available: http://technet.microsoft.com/en-us/windowsserver/dd448604.aspx [cited in p. 47]

[100] OpenStack Compute Administration Guide. Grizzly, Selecting a Hypervisor. [Online]. Available: http://docs.openstack.org/grizzly/openstack-compute/admin/content/selecting-a-hypervisor.html [cited in p. 47]

[101] SQLAlchemy. The Python SQL Toolkit and Object Relational Mapper. [Online]. Available: http://www.sqlalchemy.org/ [cited in p. 47]

[102] OpenStack Manuals. Architecture: Physical network diagram. [Online]. Available: http://docs.openstack.org/grizzly/basic-install/yum/content/basic-install_architecture.html [cited in p. 48]

[103] Apache. The Apache Software Foundation. [Online]. Available: http://www.apache.org/foundation/ [cited in p. 49]

[104] The Channel. (2010, May) Cloud.com takes on virty infrastructure. [Online]. Available: http://www.channelregister.co.uk/2010/05/04/cloud_com_launch/ [cited in p. 49]

[105] Citrix. Citrix Systems, Inc. [Online]. Available: http://www.citrix.com/ [cited in p. 49]

[106] Apache. The Apache Software Foundation Incubator. [Online]. Available: http://incubator.apache.org/ [cited in p. 49]

[107] Sabharwal, N., *Apache CloudStack Cloud Computing*, ser. Community experience distilled. Packt Publishing„ 2013. [cited in p. 50, 52]

[108] Citrix. XenServer. [Online]. Available: http://www.citrix.com/products/xenserver/overview.html [cited in p. 51]

[109] Citrix. CloudPlatform. [Online]. Available: http://www.citrix.com/products/cloudplatform/overview.html [cited in p. 51]

[110] Soheil Eizadi. (2013, August) CloudStack, Development 101. [Online]. Available: https://cwiki.apache.org/confluence/display/CLOUDSTACK/Development+101 [cited in p. 51]

[111] Citrix. NetScaler Application Delivery Controller. [Online]. Available: http://www.citrix.com/products/netscaler-application-delivery-controller/overview.html [cited in p. 51]

[112] Apache. Apache Tomcat. [Online]. Available: http://tomcat.apache.org/ [cited in p. 52]

[113] Joe Brockmeier. (2013, March) Deploying Apache CloudStack from API to UI. [Online]. Available: http://www.slideshare.net/jzb/cloud-stack-from-api-to-ui [cited in p. 53]

[114] Eucalyptus. The Eucalyptus Story. [Online]. Available: https://www.eucalyptus.com/about/story [cited in p. 54]

[115] Eucalyptus. Eucalyptus Cloud Computing Architecture. [Online]. Available: https://www.eucalyptus.com/eucalyptus-cloud/iaas/architecture [cited in p. 55]

[116] Eucalyptus. Eucalyptus Components. [Online]. Available: https://www.eucalyptus.com/docs/eucalyptus/3.2/ig/euca_components.html#euca_components [cited in p. 56, 58]

[117] Eucalyptus. Overview of Euca2ools. [Online]. Available: https://www.eucalyptus.com/docs/euca2ools/3.0/euca2ools-guide/euca2ools_oview.html#euca2ools] [cited in p. 56]

[118] DeKoenigsberg, Greg, "Eucalyptus Architecture," 2012. [Online]. Available: https://github.com/eucalyptus/eucalyptus/wiki/documents/eucalyptus-detailed-architecture-v1.6.2.pdf [cited in p. 57, 59]

[119] "Parallels Automation for Cloud Infrastructure (PACI)," Data Sheet, Parallels, November 2011. [cited in p. 60, 62]

[120] "Parallels Automation for Communication Service Providers," Data Sheet, Parallels, January 2013. [Online]. Available: http://www.parallels.com/fileadmin/media/hcap/pa/documents/PA_Datasheet_for_Carriers_Ltr_EN_01092013.pdf [cited in p. 61]

[121] Parallels. Supported Third-Party Products in Parallels Operations Automation. [Online]. Available: http://kb.parallels.com/en/8769 [cited in p. 62]

[122] "Parallels Automation 5.4: Hardware Requirements," Data Sheet, Parallels, February 2013. [cited in p. 63]

[123] Roy Thomas Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, UNIVERSITY OF CALIFORNIA, IRVINE, 2000. [cited in p. 67]

[124] Berners-Lee, T. et al., "Uniform Resource Identifier (URI): Generic Syntax," January 2005. [Online]. Available: http://www.ietf.org/rfc/rfc3986.txt [cited in p. 68]

[125] Masse, M., *REST API Design Rulebook*. O'Reilly Media, 2011. [cited in p. 68]

[126] Fielding, R. et al., "Hypertext Transfer Protocol – HTTP/1.1," June 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt [cited in p. 68, 145, 206]

[127] Laurent, S.S. and Johnston, J. and Dumbill, E., *Programming Web Services with XML-RPC*. O'Reilly Media, Incorporated, 2001. [cited in p. 69]

[128] OpenNebula. XML-RPC API 4.2. [Online]. Available: http://opennebula.org/documentation:rel4.2:api [cited in p. 70]

[129] OpenNebula. Virtual Machine Definition File 4.2. [Online]. Available: http://opennebula.org/documentation:rel4.2:template [cited in p. 71, 72]

[130] OpenNebula. Image Definition Template 4.2. [Online]. Available: http://opennebula.org/documentation:rel4.2:img_template [cited in p. 78]

[131] OpenNebula. Virtual Network Definition File 4.2. [Online]. Available: http://opennebula.org/documentation:rel4.2:vnet_template [cited in p. 81]

[132] OpenNebula. The LVM Datastore 4.2. [Online]. Available: http://opennebula.org/documentation:rel4.2:lvm_ds [cited in p. 84]

[133] OpenNebula. The VMFS Datastore 4.2. [Online]. Available: http://opennebula.org/documentation:rel4.2:vmware_ds [cited in p. 84]

[134] OpenNebula. The Ceph Datastore 4.2. [Online]. Available: http://opennebula.org/documentation:rel4.2:ceph_ds [cited in p. 84]

[135] OpenNebula. The System Datastore 4.2. [Online]. Available: http://opennebula.org/documentation:rel4.2:system_ds [cited in p. 84]

[136] OpenNebula. The Filesystem Datastore 4.2. [Online]. Available: http://opennebula.org/documentation:rel4.2:fs_ds [cited in p. 84]

[137] OpenNebula. The iSCSI Datastore 4.2. [Online]. Available: http://opennebula.org/documentation:rel4.2:iscsi_ds [cited in p. 84]

[138] OpenNebula. The Kernels & Files Datastore 4.2. [Online]. Available: http://opennebula.org/documentation:rel4.2:file_ds [cited in p. 84]

[139] OpenStack. OpenStack API Complete Reference. [Online]. Available: http://api.openstack.org/api-ref.html [cited in p. 86]

[140] Apache CloudStack. CloudStack API Developer's Guide. [Online]. Available: http://cloudstack.apache.org/docs/en-US/Apache_CloudStack/4.0.2/html/API_Developers_Guide/index.html [cited in p. 102]

[141] Apache CloudStack. Apache CloudStack API Documentation (v4.1.0). [Online]. Available: http://cloudstack.apache.org/docs/api/apidocs-4.1/TOC_User.html [cited in p. 102]

[142] "Parallels Operations Automation 5.4: PACI RESTful API Programmer's Guide," Data Sheet, Parallels, January 2013. [Online]. Available: http://download.pa.parallels.com/poa/5.4/doc/pdf/POA%20RESTful%20API%20Guide/paci-restful-api-guide-5.4.pdf [cited in p. 126, 171]

[143] Apache. Apache CXF: An Open-Source Services Framework. [Online]. Available: http://cxf.apache.org/ [cited in p. 141]

[144] Apache. Apache Wink. [Online]. Available: http://wink.apache.org/ [cited in p. 141]

[145] Apache. Welcome to Apache Axis2/Java. [Online]. Available: http://axis.apache.org/axis2/java/core/ [cited in p. 141]

[146] Deltacloud. Deltacloud Framework. [Online]. Available: http://deltacloud.apache.org/ [cited in p. 143]

[147] jClouds. The Java Multi-Cloud Toolkit. [Online]. Available: http://jclouds.apache.org/ [cited in p. 143]

[148] Apache Libcloud. One Interface To Rule Them All. [Online]. Available: http://libcloud.apache.org/ [cited in p. 143]

[149] DMTF. Cloud Management Initiative. [Online]. Available: http://dmtf.org/standards/cloud [cited in p. 144]

[150] Deltacloud. About Deltacloud. [Online]. Available: http://deltacloud.apache.org/about.html [cited in p. 144]

[151] Deltacloud. Deltacloud drivers. [Online]. Available: http://deltacloud.apache.org/drivers.html#drivers [cited in p. 144, 145]

[152] Rackspace. Rackspace Cloud Files. [Online]. Available: http://www.rackspace.com/cloud/files/ [cited in p. 145, 147]

[153] Fog. The Ruby cloud services library. [Online]. Available: http://fog.io/ [cited in p. 145]

[154] jClouds. Providers. [Online]. Available: http://jclouds.apache.org/reference/providers/ [cited in p. 146]

[155] Apache. Apache Ant. [Online]. Available: http://ant.apache.org/ [cited in p. 146]

[156] Apache. Apache Maven Project. [Online]. Available: http://maven.apache.org/ [cited in p. 146]

[157] jClouds. Apache jclouds 1.6.2–incubating API. [Online]. Available: http://demobox.github.io/jclouds-maven-site-1.6.2/1.6.2-incubating/jclouds/apidocs/ [cited in p. 147]

[158] Dasein. DASEIN–CLOUD–CORE. [Online]. Available: http://www.dasein.org/api/dasein-cloud/ [cited in p. 147]

[159] Williams, A. (2010, December) Another Giant Gets Another Sexy Startup: Rackspace Acquires Cloudkick. [Online]. Available: http://readwrite.com/2010/12/15/another-giant-buys-another-sex#awesm=~oE40TX0mteRPFf [cited in p. 147]

[160] Libcloud. Supported Providers. [Online]. Available: http://libcloud.apache.org/supported_providers.html [cited in p. 147]

[161] Libcloud . Drivers. [Online]. Available: https://github.com/StratusLab/libcloud-drivers [cited in p. 148]

[162] Libcloud. Registering a third party driver. [Online]. Available: https://libcloud.readthedocs.org/en/latest//other/registering-a-third-party-driver.html [cited in p. 148]

[163] Aeolus. Manage Your Cloud Deployments with Ease. [Online]. Available: http://www.aeolusproject.org/ [cited in p. 148]

[164] mOSAIC. Open source API and platform for multiple clouds. [Online]. Available: http://www.mosaic-cloud.eu/ [cited in p. 149]

[165] IBM. The Simple Cloud API. [Online]. Available: http://www.ibm.com/developerworks/library/os-simplecloud/ [cited in p. 149]

[166] RADICAL Research. SAGA Project. [Online]. Available: http://saga-project.github.io/ [cited in p. 149]

[167] Contrail. Open computing infrastructures for elastic services. [Online]. Available: http://contrail-project.eu/ [cited in p. 149]

[168] OGF. XtreemOS. [Online]. Available: http://www.xtreemos.eu/ [cited in p. 149]

[169] Ruby. [Online]. Available: https://www.ruby-lang.org/en/ [cited in p. 151]

[170] Matsumoto, Y., *Ruby in a Nutshell*. O'Reilly Media, 2001. [cited in p. 151]

[171] RubyGems. [Online]. Available: http://rubygems.org/ [cited in p. 151]

[172] Blake Mizerany. Sinatra. [Online]. Available: http://www.sinatrarb.com/about.html [cited in p. 153]

[173] Sinatra-rabbit. [Online]. Available: http://rubydoc.info/gems/sinatra-rabbit/1.1.6/frames [cited in p. 153]

[174] Rack. Rack Documentation. [Online]. Available: http://rack.rubyforge.org/doc/ [cited in p. 153]

[175] Thin. Thin: A fast and very simple Ruby web server. [Online]. Available: http://code.macournoyer.com/thin/ [cited in p. 153]

[176] David Heinemeier Hansson. Rails. [Online]. Available: http://rubyonrails.org/ [cited in p. 153]

[177] Harris, A. and Haase, K., *Sinatra: Up and Running.* O'Reilly Media, 2011. [cited in p. 153]

[178] cURL. [Online]. Available: http://curl.haxx.se/ [cited in p. 153, 154]

[179] VIM. [Online]. Available: http://www.vim.org/ [cited in p. 154]

[180] Wireshark. [Online]. Available: http://www.wireshark.org/ [cited in p. 154]

[181] Deltacloud. Deltacloud API. [Online]. Available: https://deltacloud.apache.org/rest-api.html#rest [cited in p. 155]

[182] OpenNebula. OpenNebula: How to Interact with OpenNebula Using Deltacloud. [Online]. Available: http://wiki.opennebula.org/deltacloud [cited in p. 168]

[183] Deltacloud. OpenStack RubyGem. [Online]. Available: https://rubygems.org/gems/openstack/versions/1.1.2 [cited in p. 168]

[184] Childers, C. CloudStack Driver for Deltacloud. [Online]. Available: https://github.com/chipchilders/deltacloud/tree/cloudstack-driver/server/lib/deltacloud/drivers/cloudstack [cited in p. 169]

[185] Deltacloud. Deltacloud: Write a provider driver. [Online]. Available: https://deltacloud.apache.org/write-new-driver.html [cited in p. 169]

[186] Lunacloud. Lunacloud: Cloud server pricing. [Online]. Available: http://www.lunacloud.com/en/cloud-server-pricing [cited in p. 173]

[187] REXML. Overview. [Online]. Available: http://www.germane-software.com/software/rexml/ [cited in p. 175]

[188] ERB. [Online]. Available: http://ruby-doc.org/stdlib-2.1.1/libdoc/erb/rdoc/ERB.html [cited in p. 175]

[189] cURL. The man page. [Online]. Available: http://curl.haxx.se/docs/manpage.html [cited in p. 205]

[190] OpenNebula. OpenNebula Market place: ttylinux - kvm. [Online]. Available: http://marketplace.c12g.com/appliance/4fc76a938fb81d3517000003 [cited in p. 207]

[191] Cybercom. OpenStack driver for Deltacloud. [Online]. Available: http://mail-archives.apache.org/mod_mbox/deltacloud-dev/201404.mbox/browser [cited in p. 209]

[192] Aaron Patterson. Nokogiri. [Online]. Available: http://nokogiri.org/ [cited in p. 222]