Technical University of Crete

School of Electronic and Computer Engineering

Σχεδιασμός και Υλοποίηση Μηχανισμού Μεταφοράς Λογισμικού σε Δεδομένα σε Περιβάλλον Υπολογιστικού Νέφους OpenStack

Implementation of the Cloud Software to Data Approach in an OpenStack Environment

Lenos Vakanas

Intelligent Systems Laboratory

Thesis Committee:

Professor Euripides Petrakis

Assistant Professor Antonios Deligiannakis

Research Collaborator Dr. Stelios Sotiriadis

September 2015

Περίληψη

Η τεχνολογία του «υπολογιστικού νέφους» (Cloud Computing) έχει καθιερωθεί ως μια τεχνολογία ανάπτυξης λογισμικού με πολλά πλεονεκτήματα με κυριότερο το χαμηλό κόστος κατασκευής, και συντήρησης συστημάτων εφαρμογών. Είναι εμφανής όμως η ελάχιστη αξιοποίηση αυτής της τεχνολογίας σε εφαρμογές που χαρακτηρίζονται από την χρήση ευαίσθητων δεδομένων πού δεν επιτρέπεται να μεταδίδονται στο διαδίκτυο ή να αποθηκεύονται σε απομακρυσμένες περιοχές του υπολογιστικού νέφους. Μια λύση αυτού του προβλήματος αναφέρθηκε για πρώτη φορά στο EC Cloud report 2010 [1] το οποίο εισάγει μια καινούργια έννοια του «υβριδικού» ή «αντίστροφου» υπολογιστικού νέφους (Hybrid or reverse cloud) που προτείνει ως λύση που εξασφαλίζει την ασφάλεια των δεδομένων, την μεταφορά του λογισμικού στα δεδομένα. Ως μία υλοποίηση της παραπάνω προσέγγισης σχεδιάσαμε και υλοποιήσαμε την μέθοδο S2D (Software to Data service) που λειτουργεί σε περιβάλλον OpenStack και επιτρέπει επιπλέον την παρακολούθηση και μέτρηση της χρήσης του λογισμικού (π.χ., χρόνος λειτουργίας) στο νέο περιβάλλον εγκατάστασης. Το τελευταίο θεωρείται σημαντικό για περιβάλλοντα υπολογιστικού Νέφους καθώς επιτρέπει την μίσθωσή ανάλογα με την χρήση (pay per use) που θεωρείται ένα από τα βασικά χαρακτηριστικά της τεχνολογίας υπολογιστικού Νέφους.

Abstract

Cloud computing offers a development platform with many benefits such as low cost application development and deployment along with minimization of maintenance and upgrades. Despite the technology's numerous advantages, health care and other application fields related with sensitive and confidential information have been reluctant to seize its offerings. This is because of the requirement for data processing data on remote cloud datacenters and therefore the transferring of sensitive data over the Internet. A solution to this problem is the reverse cloud approach that allows software to be transferred near to the data source and to be instantiated into a new cloud environment in order to eliminate the problems of processing sensitive data remotely. This encompasses the concept of virtual machine migration for cloud federations that are composed by various OpenStack systems. To achieve this we developed an innovative software to data service that allows virtual machines in the form of running instances or images to be migrated between OpenStack environments. Further, the service allows easily reconfiguration (regarding hardware features) along with monitoring and calculating of virtual machine use in the OpenStack federation.

Table of Contents

Chapter 1 – Introduction 1

1.1. Cloud Computing 2

1.2. Cloud Deployment Models 3

1.3. Motivation 5

1.4. Proposed Solution 6

1.5. Working Environment 7

Chapter 2 – Background and Related Work **9**

2.1. Cloud Computing Architecture Layers 9

2.1.1. Service Models 9

2.1.2. Virtualization 11

2.1.3. Hypervisor 12

2.2. Openstack Cloud Environment 13

2.2.1. Openstack Components 13

2.2.2. Openstack and OCCI 15

2.2.3. Openstack Technology 15

2.3. Representational State Transfer Application Programming Interface (REST API) 17

2.4. Migration In Cloud Computing 17

2.5. Software to Data 18

Chapter 3 – Software to Data (S2D) Service **20**

3.1. Service Functionality 21

3.2. Service Architecture 24

3.2.1. Front End (User) 24

3.2.1.1. User Interface (UI) 24

3.2.1.2. Instance Migration Tool (UI) 24

3.2.1.3. Instance Monitoring Tool (UI) 25

3.2.2. Back End (Server) 26

3.2.2.1. Instance Migration Tool (Server) 26

3.2.2.2. Instance Monitoring Tool (Server) 27

3.2.2.3. XML API Call 27

Chapter 4 – Implementation **28**

4.1. User Interface Implementation 28
4.2. Back End Implementation 28
4.2.1. Jersey - RESTful Web Services in Java 28
4.2.2. JavaScript Object Notation (JSON) 29
4.2.2.1. Jettison – A JSON StAX Implementation 29
4.2.3. Apache HttpComponents 29
4.2.4. Instance Migration Tool Implementation 29
4.2.5. Instance Monitoring Tool Implementation 33
4.2.6. XML API Call Implementation 34
4.2.7. Apache Tomcat 7 36
4.2.8. Launching an Instance In Intellicloud 37
4.2.8.1. Network Topology of an Instance 40
4.2.8.2. Implementation Details 41
Chapter 5 – Conclusions and Future Work 43
5.1. Conclusions 43
5.2. Future Work 44
5.2.1. Multiple Migrations at Once 44
5.2.2. Alternative Ways of Monitoring 45
5.2.3. XML API Functionality 45
References 47

Chapter 1

Introduction

Cloud computing took many shapes and names over the years before it evolved to what we know today. The idea dates back to 1961 when computing pioneer John McCarthy[1] stated that:

"Computation may someday be organized as a public utility"

The path that guides us to today's cloud computing is as follows:

- 1960 Supercomputers[2]
- 1997 High Performance Computing (HPC) - High Throughput Computing (HTC)[3].
- 1999 Volunteer Computing[4]
- 1999 Parallel Computing[5]
- 1999 Grid Computing[6]
- 2000 Utility Computing[7]
- 2006 Cloud Computing[8]

The cloud computing shares many similarities with its predecessors technologies and in simple words one may regard cloud computing as the combination of grid and utility computing. We chose one of many possible definitions given to cloud computing over the years [2]:

"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."

[1] http://en.wikipedia.org/wiki/John_McCarthy_%28computer_scientist%29
[2] http://en.wikipedia.org/wiki/Supercomputer
[3] http://rc.arizona.edu/high-performance-computinghigh-throughput-computing
[4] http://en.wikipedia.org/wiki/Volunteer_computing
[5] http://en.wikipedia.org/wiki/Parallel_computing
[6] http://en.wikipedia.org/wiki/Grid_computing
[7] http://en.wikipedia.org/wiki/Utility_computing
[8] http://en.wikipedia.org/wiki/Cloud_computing

Beside the benefits that cloud computing technology offers, there are some key factors that attracted the attention of the industry and motivated research over the last few years [2, 3, 6]:

- Decrease in hardware costs
    - Enabled the existence of Personal Computers (PCs) in almost every home hence the rapid spreading of the internet.
- Increase of computing power and advent of multi – core architecture
    - PCs were getting more and more powerful and combined with the reduced cost in hardware, building a network of PCs became cheaper than a supercomputer but equally powerful.
- The exponentially growing data volumes in modern application fields and the internet
    - More organizations needed the capabilities of a supercomputer (without its cost), with organizations being either businesses or scientific organizations.
- Emergence of Web 2.0 applications and Web Services
    - The rapid growth of the World Wide Web (WWW) in recent years has generated the need for tools and mechanisms which automatically handle tasks that are typically handled manually by humans (e.g. shopping online). Eventually, these task became more complicated and more demanding in terms of computing power, storage or network capacity.
- Existence of large – scale datacenters
    - It is a direct outcome of the above, meaning companies needed more computing power to satisfy their growing number of customers and their service demands.
- Virtualization
    - It enabled the creation of more efficient, scalable services and datacenters. The emerge of cloud technology paid special attention to the way computing services and resources are offered (e.g. leased to users) and maintained.

With the introduction of Elastic Compute Cloud[9] (EC2) and Simple Storage Service[10] (S3) in 2006, Amazon was the first to launch its own publicly available cloud. Soon after, more companies and organizations followed this paradigm including Microsoft (Azure[11] in 2009) and Google (App Engine[12] in 2008).

1.1 Cloud Computing

A very commonly asked question is "What is Cloud Computing". The answer in its most simple form is that cloud computing is a distributed network - based system that provides resources, such as hardware, software and network services, over the internet without exposing the user to the complexity of the system [5]. Previously, a supercomputer was the solution, mainly for scientific purposes, but in recent years supercomputers are replaced by what we call cloud technology. Furthermore, cloud technology is not only for scientific use but also for commercial use. Companies or individuals may need large amounts of computing power or storage for limited. Possessing the hardware and software for their operation is not the optimal solution in many cases in terms of investment, capital or human resources. In these cases, the optimal solution is cloud computing offering a pay-per-use solution of computing resources over the internet [6]. Cloud technology can be also used in many other ways in order to provide a suitable and affordable solution in use case scenarios [1, 4].

Elasticity: The ability to adjust computing resources (i.e. CPU, memory, bandwidth) to the actual (possibly varying in time) needs of an application and apply a business model to take this into account. A consumer can be charged by the resources he consumes and is able adjust the amount of resources according to his needs.

On-demand self-service: A consumer can perform various tasks automatically, including adjusting his computing resources, without human interaction.

Broad network access: The ability to use the cloud with any device connected to the internet. The interaction between user and cloud is realized by means of Web

[9] http://aws.amazon.com/ec2/
[10] http://aws.amazon.com/s3/
[11] http://azure.microsoft.com/en-us/
[12] https://cloud.google.com/

interfaces and APIs and other established technologies adopted by existing client platforms like smartphones, laptops etc.

Resource pooling: This enables serving multiple customers by dynamically allocating the resources according to demand. Typically, customers allocate computing power over the Web without caring about the ownership or physical location of the resources. The offered services become more reliable due to the fact that if a physical component fails, the system dynamically switches to another.

Measured service: A business model is applied and ensures clouds' sustainability. A business model applies a pricing model which, in turn, is realized by means of additional tools for resource utilization and resource monitoring (e.g., utilization of CPU power and memory).

Quality of Service (QoS): Except reliability, which is an important aspect of Quality of Service [15], other criteria must be met by a cloud provider ensuring the quality of services to customers including, response time, throughput, packet loss frequency, CPU load etc. Furthermore, Cloud systems are a valid choice for a wide scope of applications not only for offering desirable computational characteristics but also for its economics [1]:

Pay per use: The consumer can pay for exactly what he uses, meaning that he can scale up or down according to his needs without the risk of over or under pricing.

Cost Reduction: A consumer achieves cost reduction, not only with the pay per use scheme but also, without having to maintain or upgrade the infrastructure or software that he uses. There are no idle machines (virtual or not) due to the automatic scaling of allocated resources according to processing load.

Computing is ideal for a wide audience of consumers. For example a small or startup business can rent computing power and storage needed without risking excessive hardware, software procurement or maintenance costs. Apart from large or small business, there are also examples of individuals that can be also benefited from cloud technology (e.g., by using cloud storage such as Dropbox[13], Google Drive[14], iCloud[15] etc.). The applications that are using cloud computing are many and keep growing.

[13] https://www.dropbox.com/

1.2 Cloud Deployment Models

The deployment models describe the cloud's operators and consumers.

Public Cloud: Usually owned and managed by enterprises or organizations this cloud model is intent for the general public. Usage of this cloud model can be free or in a pay-per-use. Examples of this model are Amazon, Google Apps, Windows Azure etc.

Private Cloud: Exclusively owned and operated by a single entity, this cloud model is used for satisfying the needs of the owner enterprise and sometimes it can provide functionality to customers if the owner chooses to. An accurate example is eBay[16].

Hybrid Cloud: This cloud model is the combination of a public and a private cloud. It provides the ability, for an entity, to own and manage a private cloud and use functionality offered by public cloud providers, with the advantages of both models. It comes as a solution to problems found in public cloud models, such us the problem with patient data in e-Health environments [7, 10].

Community Cloud: It is intended to serve consumers (e.g., organizations or individuals) sharing common interests or concerns. It can be owned and managed by someone in the community or by a third party.

1.3 Motivation

In recent years cloud computing has been fostered as the technology to offer virtualized resources to Internet users on a bespoke manner. This includes hardware, software and platform that could be delivered as a service. For many areas such as industry, agriculture etc. this has been served as an efficient approach with regards to the minimization of operational costs and increased elasticity; yet not in the healthcare domain. Data stored in cloud are usually available over the Internet and could contain confidential and private health information. Today there are various standards, regulations and recommendations such as national legislation, ISO standards (ISO

[14] https://www.google.com/drive/
[15] https://www.icloud.com/
[16] http://www.ebay.com/

80001[17]) and the need to comply with security standards (ISO 27000[18]), thus there are severe restrictions to data transfer and storage. As a result, cloud computing that is profoundly based on the Internet and openness, becomes a hurdle to its adoption in health care. Along with health care, more application owners with similar concerns are:

- Businesses or Government related organizations

Entities that manage sensitive or classified data are reluctant to cloud based solutions due to security risks of transferring data over the internet. Remote storage of data might be a problem as well.

- Scientific or Research organizations

For these entities, the major problem can be the amount of data that have to be transferred over the internet (i.e. it may cost too much or it may take too long to transfer big amounts of data). In general, when remote storage or transferring data over the internet is a concern cloud technology runs into a problem.

1.4 Proposed Solution

The solution to this problem comes by utilizing hybrid cloud technology [1, 8]. Also referred to as "reversed cloud approach" [7, 9, 10] builds upon the idea of bringing the software to the data rather than transferring the data to the public cloud where the software is installed. To achieve this, all we need is a public cloud which is the software provider, a private cloud which is the consumer and a service that will transfer and deploy the software from the public to the consumer's cloud. Afterwards, the consumer can process his data locally and pay the usage fees to the provider. By using this solution, a health care provider can use the technology and offerings of public clouds, without breaking the restrictions related to data privacy.

In our work we propose an implementation of the Software to Data (henceforth called S2D) service that will transfer, deploy and monitor the software from a public cloud provider to the private cloud and data owner. The implementation consists of two

[17] http://www.iso.org/iso/catalogue_detail.htm?csnumber=44863
[18] http://www.iso.org/iso/catalogue_detail?csnumber=63411

modules, the first being responsible for transferring and deploying the software between the two clouds and the second one responsible for monitoring the usage of the software e.g., up-time and for providing this information to service provider (e.g., for billing the customer). The software to data approach removes any restrictions or concerns in regards to data usage in clouds as described in the previous section, by deploying the software in the location of the data so that the processing takes place locally. The parties, as described in the previous section, no longer have to worry about data privacy or data size. Naturally, this solution has some drawbacks that are not necessarily major. For example, the elasticity or scalability properties of the cloud depend only on the consumer's private cloud properties. Following this line of thinking, all the properties or benefits of cloud technology are only as good as the consumers cloud, though some of them may be better, such us reliability and QoS metrics like response time considering that the processing takes place locally. Either way, we believe that the solution of the software to data approach enables the use of cloud technology, by application owners that previously needed to possess their own hardware and software infrastructure for their work. Other requirements are that the two clouds, public and private, must be compatible with each other and that the private cloud must meet the hardware requirements of the transferred software.

The proposed solution proposes a service which through an easy to use GUI (Graphical User Interface) or a REST API call with an XML document will offer easy to transfer images (Openstack compatible), automatized configuration and easy deployment and thus achieving migration of a running instance and re-instantiation between two Openstack clouds.

1.5 Working Environment

Our proposed S2D solution was deployed using the Intellicloud[19] and FI-LAB[20] infrastructures as test bed. Both clouds are running Openstack[21] and FI-WARE[22] on top of Openstack so they are compatible with each other. We are communicating with the two clouds using a REST API provided by the Openstack environment

[19] http://cloud.intellicloud.tuc.gr/
[20] https://cloud.lab.fiware.org/
[21] https://www.openstack.org/
[22] http://www.fiware.org

FI-WARE is a European initiative that aims to provide an open platform for developing and deploying Future Internet applications and services. Application development is realized by means of a set of pre-defined basic services called Generic Enablers (GEs) and which can be viewed as building blocks for an application. For example, there is a GE for authenticating users and developers of the cloud environment so that, they do not need to develop this service on their own. GEs offer their functionality through a REST API or a web interface (or both) to its users. Overall, FI-WARE aims to provide an innovative and cost-effective way of building cloud applications while maintaining high quality of service and security.

Chapter 2

Background and Related Work

2.1 Cloud Computing Architecture Layers

The layers which compose cloud's system architecture can be viewed as five different layers (Figure 2). The first one of course is the hardware layer, which contains all the hardware needed for the cloud to work. Processors, RAM, storage, network and others all belong to this layer. On top of the hardware layer is the virtualization layer which is responsible for managing the hardware, deploying the virtual machines (VMs) and dynamically allocating capacity to them according to demand. The remaining three layers are also called service models and they describe the functionality offered by the cloud.

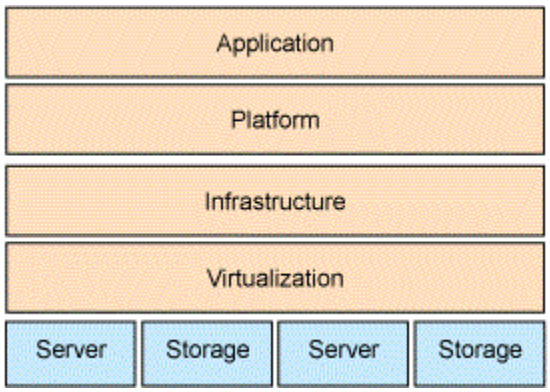

Figure 2 – Cloud Computing Layers[23]

2.1.1 Service Models

Cloud computing services are offered in three different service models:

- **Infrastructure as a Service (IaaS)**

[23] Source: http://www.ibm.com/developerworks/aix/library/au-cloud_apache/

It is the ability to rent hardware such as storage, computing power or network. The consumer is responsible for installing and maintaining the operating system and other software, but the responsibility of upgrading or maintaining the hardware resides to the provider. Examples of storage clouds are Amazon S3, SQL Azure.

- **Platform as a Service (PaaS)**

The cloud's provision for consumers who need a software platform (providing all basic tools and software services such as Generic Enablers in the case of FIWARE-LAB) for deploying their applications. The consumer gets a functional virtual machine with an operating system of his choice and he can use it as a developing platform or as a platform for deploying services without worrying about upgrades or maintenance of hardware. Examples of such services are Saleforce[24], Google App Engine, Windows Azure (Platform) and FIWARE-LAB.

- **Software as a Service (SaaS)**

This is the most usable cloud service. It allows a consumer to use services provided from the cloud provider or even other consumers (e.g. Google Docs). The consumer has no control over the service's software or hardware and he can only use it thought provided APIs or interfaces by the service provider. In addition, he is not required to maintain or upgrade the hardware or the software. Examples are Google Docs, Dropbox.

As showed in Figure 3, cloud's service models aim to satisfy a consumer's needs by providing different levels of responsibility in resource management and thus allowing him to focus on the part that he really wants without additional burden.

[24] http://www.salesforce.com/

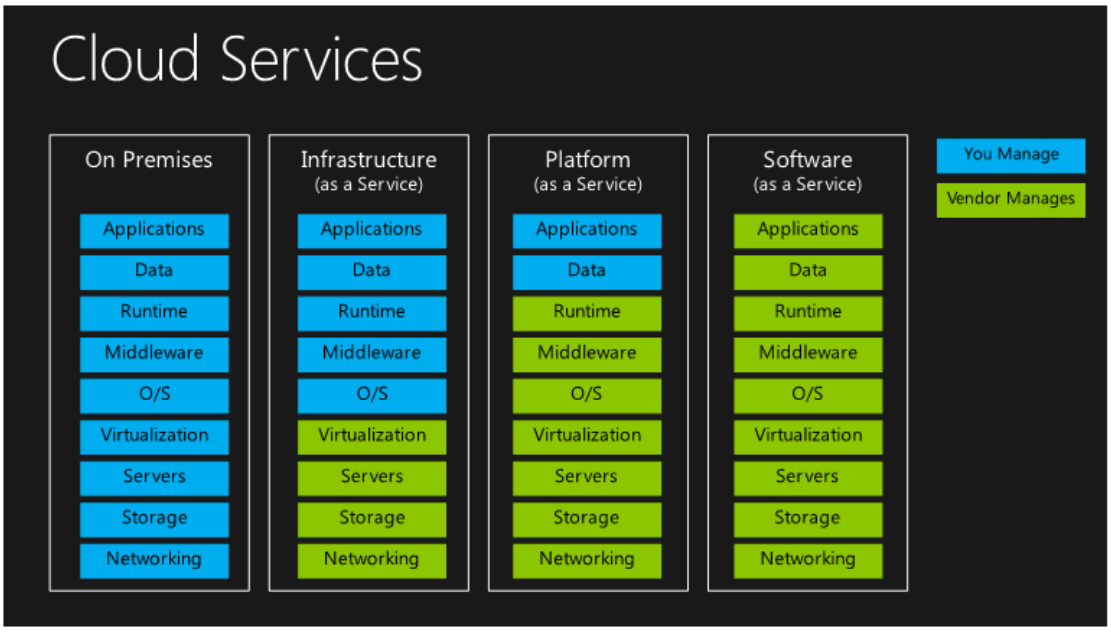

Figure 3 – Cloud Services[25]

2.1.2 Virtualization

Although this technology was not intended to be used in cloud computing it is vital for today's clouds. It dates back to 1970's where IBM needed a technology for running multiple applications on a mainframe [11]. Virtualization provides layer of abstraction between the hardware and the software. The product of virtualization is called Virtual Machine (VM) and it is representing a fully functional computer. The main difference from the tradition computer is that allows definition of multiple VMs with different operating systems over the same hardware.

A cloud provider may have certain amount of servers and network bandwidth but with virtualization allows serving a large number of consumers with diverse service demands (e.g. each one may use different operating system. As a result of virtualization, clouds are able to efficiently exploit their computing power. Also, combining operating system and software into a virtual machine achieves easier backups and faster re-deployment since everything living within a virtual machine can be fully backed up at any state.

[25] Source: http://www.cetancorp.com/solutions/cloud/private-public-cloud/

2.1.3 Hypervisor

Hypervisor is the software responsible for creating the virtual environment where the virtual machines operate and for dynamically allocating hardware resources to them. It is also known as Virtual Machine Monitor (VMM) and it can be one of two types. Clouds use type 1 hypervisor also known as native or bare metal hypervisor. It is usually a "light" software operating directly above the hardware resources which results in good performance. Its responsibilities are, as described above, creation of the virtual environment and allocating the necessary hardware resources to the virtual machines [12]. There is also type 2 hypervisor which needs a host operating system to operate and usually results in lower performance [11].

Overall, virtualization is a vital part of cloud. As showed in Figure 4, visualization lays directly above the hardware resources and enables the use of cloud's service models with the functionality each one contains combined in a VM or several VMs in the case of SaaS. Cloud consumers have the illusion of infinite hardware resources available to them on demand [3].

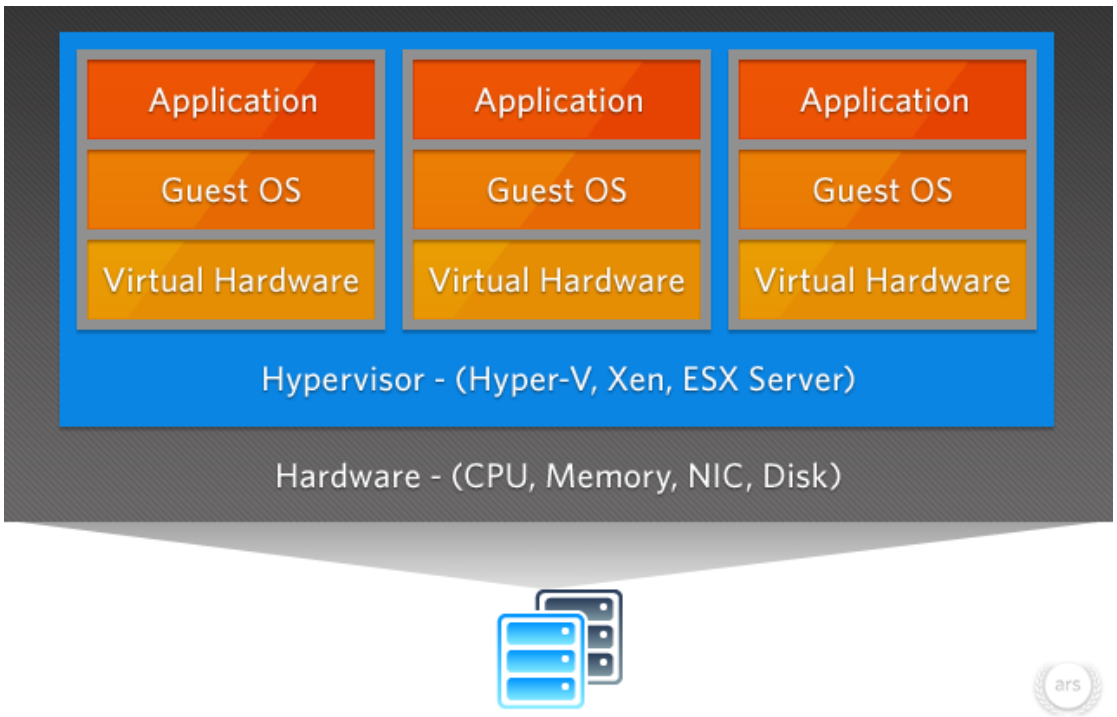

Figure 4 – Virtualization[26]

[26] Source: http://arstechnica.com/business/2011/02/virtualization-in-the-trenches-with-vmware-part-1-basics-and-benefits/

2.2 Openstack Cloud Environment

Openstack[27] is an open source project for managing a large pool of resources such us computing power, storage and network. It is an operating system for clouds [13]. The initial contributors of this project, back in 2010, was Rackspace and NASA but since then, more than 200 companies have joined the project while the list includes names like IBM, Cisco, Dell, HP and many more. Openstack's software is written in Python and it is cross platform.

2.2.1 Openstack Components

Openstack components are the software or systems which manage the cloud's resources. A user can use them either by the dashboard (Graphical User Interface provided by Openstack) or directly by the API each component provides for communication. All of Openstack components are showed in Figure 5.

- Nova - Compute Service

Designed to manage the pool of compute resources, Nova is the computing resources controller. It is written in Python and can work with type 1 or bare metal hypervisors used in virtualization technologies.

- Quantum - Networking Service

Quantum is the system that manages the clouds networks and IP addresses. Users can use this system to create their own networks or new Floating IPs which are used by the virtual machine for internet access.

- Glance - Image Service

Glance is the managing service of images in the cloud. Cloud's images are the operating system installed on a virtual machine and can also be a backup of a virtual machine with both operating system and additional software call a Snapshot.

- Keystone - Identity Service

[27] http://www.openstack.org/

Keystone is the authentication system of the cloud. It manages the cloud's users both in and out of the cloud as well as within the cloud depending on each user's permissions.

- Horizon - Dashboard or UI Service

The User Interface where you can control and use the above mentioned services. Horizon is a web-based dashboard where administrators and users can manage and view their resources.

- Cinder

Cinder is the block storage service used in Openstack. It allows the users to request and consume those resources via a self-service API.

These Openstack components are still under development.

- Ceilometer

It is a metering and monitoring service aimed to provide all the necessary measurements to establish a reliable and accurate billing system

- Heat

Heat is the orchestration engine under development for Openstack which will provide the ability to launch multiple cloud applications based on templates.

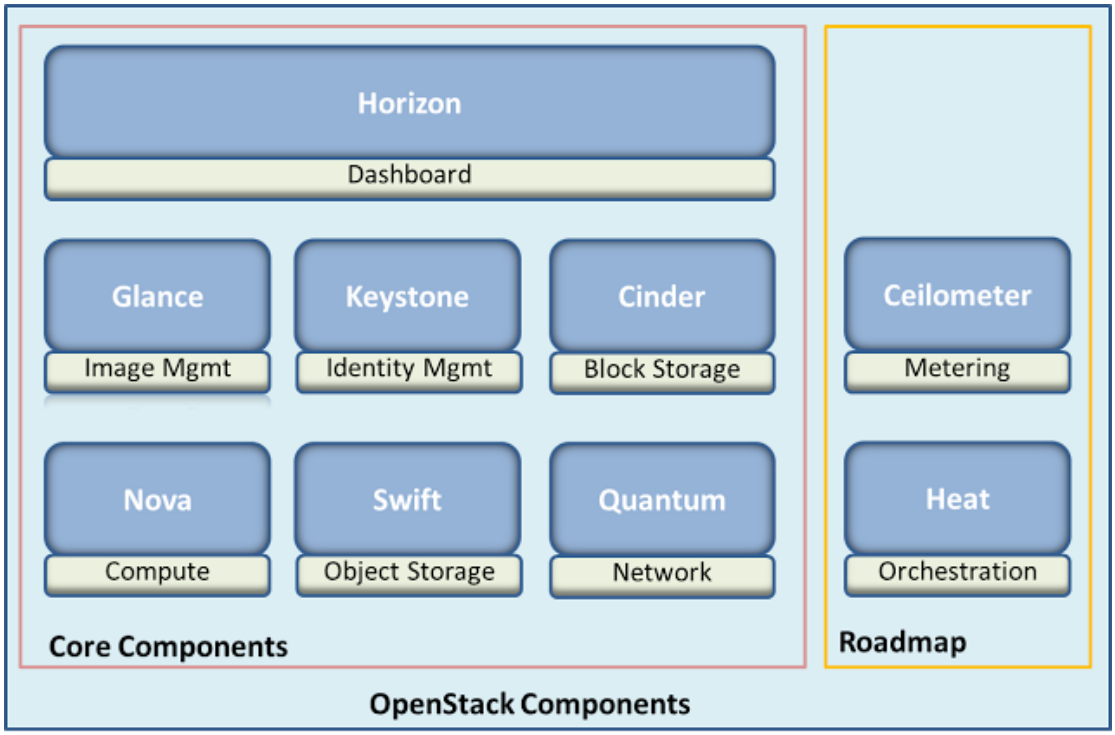

Figure 5 – Openstack Components[28]

2.2.2 Openstack And OCCI

OCCI stands for Open Cloud Computing Interface and it is a RESTful – based protocol for managing tasks in cloud. Initially, this protocol was intended for IaaS service model but now it supports all three of the service models. It contributes to the development of interoperable tools used for development, automatic scaling and monitoring. Openstack has implemented this standard aiming to exploit more their properties for integration, portability, interoperability and innovation [14].

2.2.3 Openstack Technology

Before continuing with the functionality details we provide the following clarifications and terminology definitions concerning the Openstack cloud environment [16].

[28] Source: http://applycloud.blogspot.com/2013/05/openstack-components.html

Cloud A: The "source" cloud of the software. Meaning the cloud we want to migrate the software from. In a real life scenario this would be the public cloud.

Cloud B: The "target" cloud for the software. Meaning the cloud we want to migrate the software to. Again, in a real life scenario this would be the private cloud.

Call or API call: A REST API call. The service uses calls for communicating with the user interface and the cloud's operating system (Openstack). All the information about the call to the cloud's system can be found in the Openstack REST API documentation[29].

Instance: A virtual machine that runs on the cloud. In order to launch an instance you need to provide instance name, image name and flavor.

Flavor: Predefined virtual hardware templates defining number of CPU cores and size of RAM and disk space. There are five different templates in order to satisfy any consumer. Flavors are simulating the physical hardware that a consumer would buy in order to build a PC or a server.

Images: An image is the operating system used to launch and run a virtual machine.

Snapshot: Snapshot is the image created of a running instance. Simply put, taking a snapshot of an instance will back up the current disk state of that instance. As a result, snapshots are also images that can be used as virtual machine templates because they preserve any additional software that the instance had.

Security Groups: A collection of IP filters (Ports) that are being applied to an instance's networking. The owner of the instance must configure correctly the security groups in order to be able to communicate with the instance and all of its software. For example, port 22 is used for secure communication (SSH) with the instance and port 80 is used for communication of an Apache server[30]. These IP filters or ports are called **Rules** of a security group.

Floating IP: The public IP used to connect and communicate with an instance. The consumer can create limited number of floating IPs and only one is allocated per

[29] http://developer.openstack.org/api-ref-compute-v2.html
[30] http://httpd.apache.org/

instance. Apart from this IP, each instance has one more IP, local IP, which is used for communication with the cloud system only.

Keypair: The keypair allows you to access via SSH your instance. Each instance can have only one keypair registered and the owner must provide the same key in order to be authenticated and be granted access to the instance. On the contrary, a keypair can belong to several instances.

2.3 Representational State Transfer Application Programming Interface (REST API)

REST is an architectural style of designing APIs which are used for communication between client – server or two applications. REST is a collection of guidelines and best practices that allows you to create a web service that is lightweight, scalable, language and platform independent [18]. Most commonly, REST runs over the HTTP protocol and makes use of the HTTP verbs (GET, POST, PUT, DELETE, etc.) to send and receive data and often those data are composed in JSON or XML format. Using REST APIs as a way of communication is increasing in the web services world and is particularly the preferred way of communication in cloud computing. For example, in Openstack, the most of the GEs are communicating with their users and each other via REST APIs and the use JSON or XML to format their data.

2.4 Migration In Cloud Computing

Cloud migration is the execution of a set of job tasks to relevant resources which could offer improved load balancing and higher throughput of jobs with as little to none downtime (Downtime: time from process state running to suspending to running again). Migration is categorized into two generic procedures known as process and live migration [19].

Process migration, the oldest technique of the two going back to 1980's, is the procedure of transferring a process between two machines. In the level of operating systems, process migration was achieved by letting programs communicate to each

other in a cluster of machines via appropriate kernel design. Such operating systems are Accent, ChorusOS, Amoeba but they do not support clusters of independent machines running on different operating systems something that limited the interest in this technology [20].

A step forward from operating system level migration was user level indented for long-running applications running on cluster machines with unmodified commercial operating systems. Such systems are Condor, CoCheck and MPVM although running on commercial operating systems they could support only specific applications as they required processes which did not support common operating system services such as inter-process communications [19, 20].

In object level migration, several systems designed as middleware and programming languages offered object-based migration. Abacus, Emerald, Globus, Legion and Rover being some of those systems could reduce the amount of state needed to be recorded and moved in order to achieve the migration of an application. But, applications needed to be rewritten using new programming language environments thus they could not move legacy applications [19, 20].

A more promising but challenging migration technique is VM migration or live migration which provides the ability to move an entire VM from a host resource to a target resource. Strong sell-points of this technique is that the migration is performed without pre-emptying execution and any perceived degradation and the fact that the whole VM is being transferred which does not affect downtime. Although being a promising technique, VM migration faces significant challenges. Wide area environments require large amount of bandwidth and memory capacity due to the large amount of data being transferred [20], something that challenges the effectiveness of this technique on today's systems and networks.

2.5 Software To Data

A concept discussed in detail in FI-STAR project [7, 9, 10] that aims to eliminate restrictions which are responsible for the minimal usage of cloud technology by various application fields especially health care. Software to data refers to the idea of

transferring the software to the data location and thus avoiding the transferring of data over the internet which is illegal for health care. In order to accomplish this goal, hybrid cloud technology is utilized. As a result, the provider offers his services via public cloud and the consumer uses these services in his private cloud achieving maximum security over his data (patient data or otherwise).

As shown in [7], such system can be implemented by establishing two sides, the provider's edge and consumer's edge. Provider's edge represents the public cloud which will be providing software or GEs and the consumer's edge where a private cloud will exist and these GEs will get instantiated. A single link will be preserved through the two clouds and its purpose will be to allow software to be downloaded from the public to the private cloud. By using this set-up, no sensitive data are transferred over the internet thus this technology could help reluctant fields to seize the offerings of cloud computing.

Alongside proposals like [7], others are focusing on network communication research as it is a key factor of cloud computing. Software Defined Network (or SDN) is a network that could adapt to the needs in real time. As explained in [21], hybrid cloud technology needs an equally smart network that could dynamically adapt to the bandwidth needs. Hybrid clouds and Software to data could take advantage of such network due to the large amount of data that need to be transferred over the internet as VMs can reach sizes of several gigabytes. SDNs are using software to monitor the network traffic and dynamically switch the traffic where is most needed thus making the use of network more efficient.

Chapter 3

Software to Data (S2D) Service

In the current chapter we present the design or the S2D service.

The service is utilizing Openstack's REST API to allow its users to migrate a virtual machine between two clouds. Users can perform the migration using the web-based user interface or by sending an xml document containing all the needed information. Should a user chooses to perform migration through the Web interface, we should follow the eight steps of the interface in order for the service to gather all needed information. The user provides information about his credentials, the virtual machine he wants to migrate, configuration of the newly deployed machine and so on. Apart from the web interface, the user can also send directly to the service all this information in an xml document. The xml document has a predefined structure validated by an xml schema and the user has to fill all the fields which will be used by the service.

The migration service executes the three modes of operation referred to above, the first being responsible for transferring and deploying the software, the second one is responsible for monitoring the usage of the software (e.g. up-time) while the third one is an alternative to the user interface but only for the first module, meaning it offers to the user the alternative for using the migration service as an API call containing all the needed information in an XML document.

We would define a successful migration as the process of moving a running instance (and its software) from cloud A, deploy it in cloud B while maintaining its hardware, software and network configurations. So, when the process is finished you would find in cloud B a running instance including your software but it will be of the same flavor, same rules in the security group and a public IP for you to use. Obviously, the IP number will not be the exact same as before because it varies based on geographical location, organization and so on.

3.1 Service Functionality

To achieve VM migration, the service guides the user through the process and at the same time performs some automated actions resulting in a less complex and time consuming process. Currently, the service offers this functionality for Intellicloud[31] and FI-Lab[32] with both clouds using Openstack.

Now, we describe the functionality provided by the service through both the user interface and the API call.

Authentication: The user provides his tenant ID, username and password in order to be authenticated by the cloud. Authenticating generates a token which is being used in every action the user (or the service itself) performs on the cloud. The token is unique for every user and it has a lifecycle predefined by the cloud. For example, Intellicloud's token is valid for 24 hours.

Get Instances: The service retrieves a detailed list of the instances registered to the user.

Get Images: The service retrieves the images registered to the user including snapshots.

Get Instance's Details: Retrieves the information which describes an Instance.

Create Snapshot: Creates a snapshot of the running instance which the user selected for migration. In addition, the service stores all the properties of this instance which will be later be used for launching the new instance in cloud B with the same configurations such as security group rules or flavor. For this action, the user provides a name for the snapshot and the name of the instance he wants to snapshot.

Download Snapshot: The service downloads and stores temporarily the previously created snapshot.

Upload Snapshot: Because a snapshot is also an image, the service creates a new a new image containing the data of the previously downloaded snapshot. The user provides the name of the new image, the format and if it is going to be public or

[31] http://cloud.intellicloud.tuc.gr/
[32] https://cloud.lab.fiware.org/

private (access). The service will perform the following actions to achieve this result without exposing the user to each action.

1. Create a new blank image with the given name
2. Update the blank image according to the format (e.g. .qcow2) and access (e.g. public) that the user specified.
3. Upload the data the snapshot's data to the new image. The snapshot's data contain information about the instance's operating system and software which will be used to launch the same instance in the target cloud.

Keypair Actions: The keypair must be allocated to the user and provided to the new instance before its creation. As a result, the service will save the name of the keypair that the user chooses.

- **Create Keypair:** The user provides the name of the new keypair and the service will create a new one with that name and allocate it to the user.
- **Import Keypair (from Cloud A):** The user selects the keypair from cloud A and the service imports it in cloud B. For this action to be completed, the service firsts retrieves the list of user's keypairs from cloud A and after the selection of the user, imports it in cloud B.
- **Select Keypair (from Cloud B):** The service retrieves the list of keypairs allocated to the user and the user selects one.

Launch Instance: The user provides the name of the instance and its security group name. Afterwards, the service will set the following.

- **Keypair:** Keypair of the instance will be the previously selected by the user.
- **Image:** Image will be the new image which the service created from the data of the snapshot.
- **Flavor:** The service will fetch it from Cloud A and set the same flavor in Cloud B.
- **Security Group:** The service will either create a new security group or will allocate to the instance the default one depending on the users input. If the user sets the name as "default" or if he leave it empty, the service will select the default security group. Now, if the user sets a different name, the service

will generate a new security group with that name. Independent of the user's input, the service will:

1. Get the security group that the instance had in Cloud A.
2. Get all of the rules inside that security group.
3. Insert all of the rules in the security group that the user selected for the instance in Cloud B.

The rationale behind all these actions is that we want the instance to be launched in Cloud B and immediately be operational (including all of its software) without any further configuration by the user.

IP Number Actions: The user can choose from a list of free IPs or chooses to create a new one to allocate to the new instance. Free IP is one that belongs to the specific user but it is not allocate to any other of his instances. If the user chooses to create a new IP, the service will

1. Create a new IP
2. Allocate it to the user's account
3. Allocate the new IP to the instance.

Instance Overview: The service fetches all the information about the instance which the user created. This is also a way for the user to check that the process was successful (the other way is directly through the cloud's dashboard). The fetched information is directly from the cloud's system and not by any information given by the user during the process ensuring the success of the process.

Get Instance's Usage Data: The user provides his cloud credentials and after his successful authentication by the cloud's system, the service fetches the usage data of his instances. The usage data includes the total of CPU hours used, total GB of storage, RAM and so on.

Reset System: This action allows the user to reset the state of the service. This means that the user's session will be deleted along with any information he provided till that point, including any images that he downloaded to the service's server. However, any actions that had already been executed on any of the two clouds are not affected meaning if the user wants an action done on a cloud reverted, he has to do it through the cloud's dashboard (e.g. deleting a snapshot). This action is only available when

the user uses the migration tool of the service because in the case of the monitoring tool the user does not provide anything more than his cloud credentials.

3.2 Service Architecture

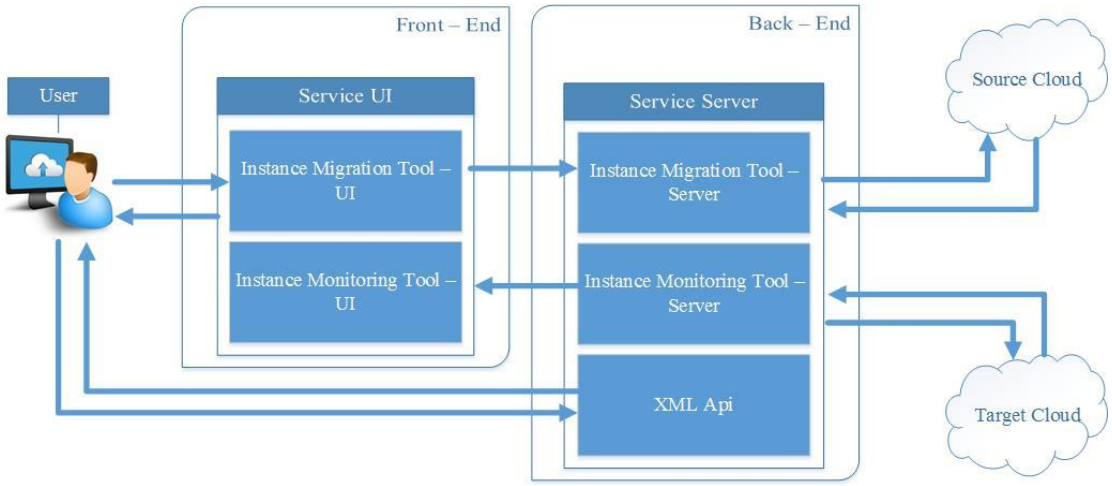

Figure 7 – Service Architecture

3.2.1 Front End (User)

The front end of the service is what the user can see and control which are the User Interface and the information being provided to the service. Information is transferred from the front end to the back end by REST API calls being performed by the UI on the service REST API.

3.2.1.1 User Interface (UI)

The user interface allows the user to interact with the service and achieve his goal which in our case is to migrate and monitor an instance. The user, through the UI, uses the instance migration tool in order to migrate an instance and the monitoring tool in order get the usage data of his instances.

3.2.1.2 Instance Migration Tool (UI)

The migration tool provides the necessary functionality that the user will need to migrate his instance and guides him through the process. The layout of the migration tool is nine numbered steps which in step 1 the process starts and ends at step 9 with an overview of the process. During those steps the user has to provide the required

information of each step in order for the service to be able to perform the actions on the cloud. The nine steps are composed into tabs where the user can find the fields where he has to insert the information along with navigation buttons, action buttons and information on each step and its fields. The user has to follow the steps sequential in order to have a successful migration.

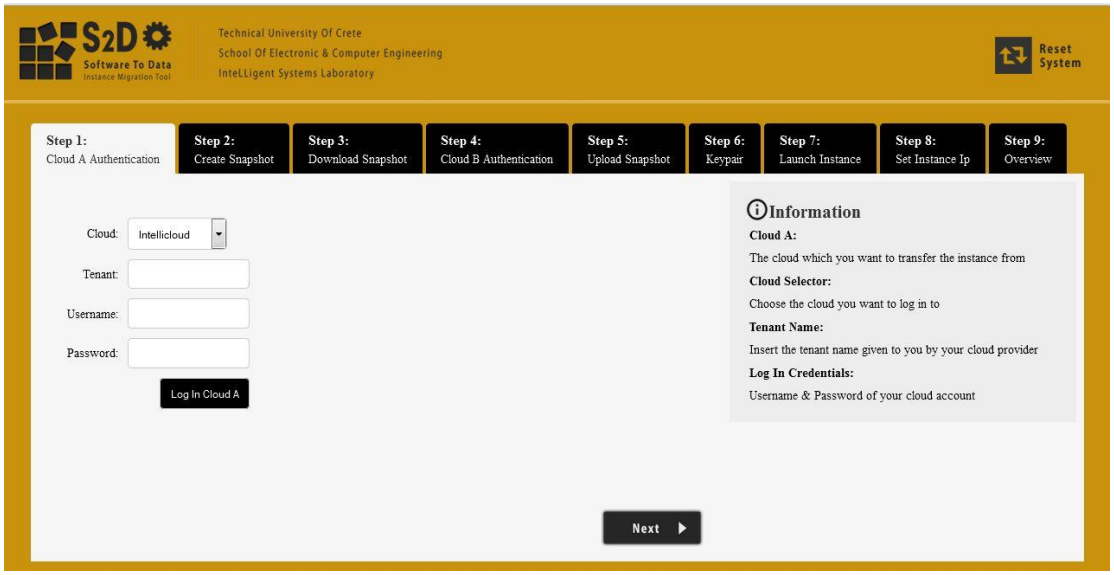

Figure 8 – Migration Tool

3.2.1.3 Instance Monitoring Tool (UI)

This tool allows the user to get the usage data of every instance he owns. The user only needs to provide the information in order to be authenticated by the cloud's system and then the service fetches the usage data of his instances. The usage data includes up-time of each instance, number of cores in use, CPU hours, storage used and other useful metrics.

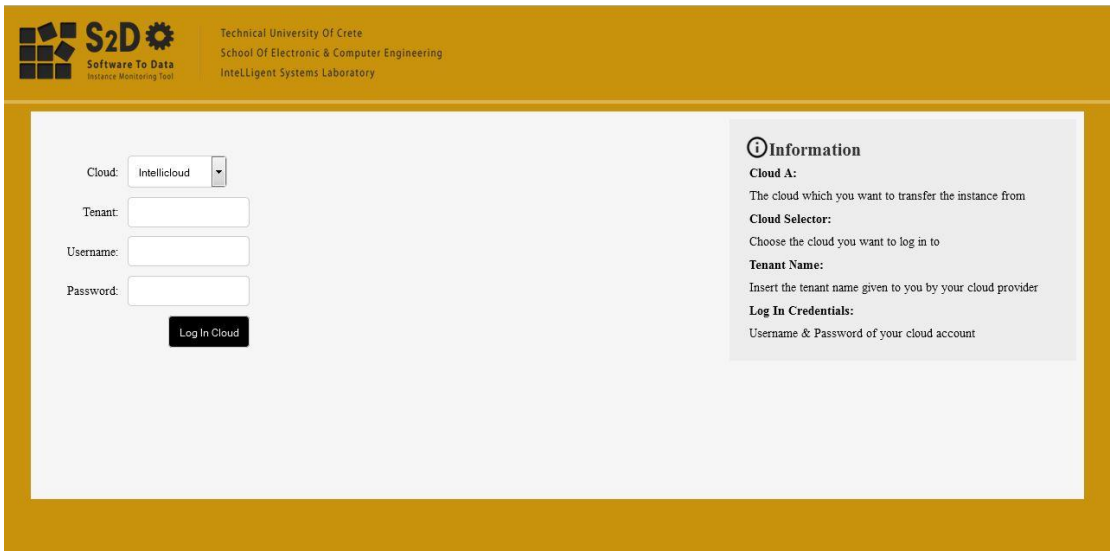

Figure 9 – Monitoring Tool

3.2.2 Back End (Server)

The back end of the service is composed by three main tools which offer all the functionality of the service. Each tool is responsible of a specific task and information required for the success of that task is provided by the UI or an XML document. The back end stored temporarily in the server the image which the user downloads with the intension of uploading it later on the second cloud. Also, the service stores some information on the user's session in case that information is needed later and avoids asking the user for providing information he already provided.

3.2.2.1 Instance Migration Tool (Server)

The migration tool contains the necessary functionality to perform an instance migration between the two Openstack clouds. As discussed above, the user completes 9 steps for a successful migration with each step asking for the required information for a specific action on the cloud. The migration tool processes the information provided by the user on each step and performs the corresponding action on the cloud. First, the migration tool will do any processing needed on the information before performing the request on the cloud and after a successful response it may save temporarily some information of the request and/or response for future use.

3.2.2.2 Instance Monitoring Tool (Server)

The user can get the usage data of his instances by providing the authentication credentials. This tool performs two API calls to the cloud's system, first is the authentication of the user and second is to retrieve from the system all the usage data of the user's instances.

3.2.2.3 XML API Call

Besides the user interface, the service provides to the user the ability to migrate his instance by performing an API call containing an XML document. This xml document must have a specific syntax and must contain all the information which the service will need to perform the necessary actions on the two clouds and migrate the user's instance. As mentioned above, this API call contains the functionality to validate the xml document, extract the information from it and feeds that information to the migration tool. The process follows the same path as the user does from the UI, meaning this tool performs API calls on the migration tool in the same order and information. Though, the user cannot use the XML API call to monitor his instances.

Chapter 4

Implementation

In order to develop the functionality of our service we used several technologies. In this chapter we will list the technologies used in our service and how in order to achieve our goal.

4.1 User Interface Implementation

The UI offers an easy and simple way of migrating a VM. It does not require any special knowledge other than the basics of an openstack environment because the information required for a successful migration is mainly found on the user's openstack account. For the development of the UI, we used HTML[33] and CSS[34] for the Graphical User Interface (GUI) and JQuery[35] for dynamic manipulation of data on the screen and for communication with the service REST API. Furthermore, the communication with the service is done by JQuery Ajax[36] calls on the service REST API.

4.2 Back End Implementation

The abstract flow of process of the back end is that it gets feed information by the front end or by direct communication with the user through the XML API. It uses that information in order to perform several actions on the two clouds by communicating with the cloud's REST API. For the implementation of the back end we used as programming language JAVA with several open source software.

4.2.1 Jersey - RESTful Web Services in Java

Jersey[37] is a framework for developing RESTful web services in java. More detailed, Jersey is a reference implementation of JAX-RS (Java API for RESTful Web Services) which is using annotations, such as @GET, @Produces, in order to simplify the development of RESTful web services in java. We used Jersey in order to develop our REST API which is used for client – server communication.

[33] http://en.wikipedia.org/wiki/HTML
[34] http://en.wikipedia.org/wiki/Cascading_Style_Sheets
[35] https://jquery.com/
[36] http://en.wikipedia.org/wiki/Ajax_%28programming%29
[37] https://jersey.java.net/

4.2.2 JavaScript Object Notation (JSON)

An alternative to XML, Json[38] is a human – readable data format that is mainly used for communication between client – server or between web applications. Json is a language independent data format which originated from JavaScript [39]and is consisting of attribute – value pairs same as JavaScript objects. Json is easy to read and write also it is shorter than XML because it does not require end tags which are some advantages over XML.

4.2.2.1 Jettison – A JSON StAX Implementation

Jettison[40] is what STaX[41] and DOM is for XML, meaning is a collection of java APIs that help developers parse and manipulate Json Objects in java. Jettison is also an open source plugin of java and it provides an easy way of processing Json. We are using jettison to process out data in the service since Json is the format of our data.

4.2.3 Apache HttpComponents

Apache HttpComponents[42] offers a toolset for creating and maintaining everything that has to do with the HTTP protocol in java. It is also an open source project of Apache Software Foundation and our use of this toolset comes for our need to perform HTTP calls on the, provided by Openstack, cloud's REST API.

4.2.4 Instance Migration Tool Implementation

The migration tool uses all the previously mentioned technologies in order to provide the functionality for communication with both the UI and the clouds. Furthermore, the communication with the clouds is done by performing calls on some of the Openstack components which are Nova compute service, Glance image service, Keystone identity service and Quantum networking service. Each component can be reached through a URL consisting of the cloud's IP and the port which the component listens to, but these numbers are not the same of every cloud as a result we are not listing the exact numbers.

[38] http://en.wikipedia.org/wiki/JSON
[39] http://en.wikipedia.org/wiki/JavaScript
[40] http://jettison.codehaus.org/
[41] http://en.wikipedia.org/wiki/StAX
[42] http://hc.apache.org/

Step 1: Cloud A Authentication

- **Parameters:** Cloud, Tenant, Username, Password
- **Description:** The service prepares and performs the call for authentication by the cloud which the user specified and after successful authentication the service displays the log in details of the user (Username, Tenant, Source Cloud).
- **Keystone - Identity:** POST, http://{IP}:{port}/v2.0/tokens

Step 2: Create Snapshot

- **Parameters:** Selected instance, Snapshot Name
- **Description:** The service retrieves and displays the instance list owned by the user. Then, the user selects one instance which he wants to create a snapshot of and presses the Create Snapshot button. After the cloud's system has successfully created it, the service will display the name of the snapshot and instance name but behind the scenes it will also save the instance's security group details and flavor details for later use in the creation of the new instance at the target cloud.
- **Nova - Compute:** Instance list: GET, http://{IP}:{port}/v2/{Tenant_ID}/servers/detail
- **Nova - Compute:** Create snapshot: POST, http://{IP}:{port}/v2/{Tenant_ID}/servers/{Instance_ID}/action
- **Nova - Compute:** Security Group: GET, http://{IP}:{port}/v2/{Tenant_ID}/servers/{Instance_ID}/os-security-groups
- **Nova - Compute:** Flavor: GET, http://{IP}:{port}/v2/{Tenant_ID}/flavors/{Flavor_ID}

Step 3: Download Snapshot

- **Parameters:** Selected Image
- **Description:** The service retrieves and displays the image list owned by the user including snapshots. The user selects the snapshot he previously created and presses the Download Image button. After the download is performed, the service displays the name of the image and its size. The downloaded image is stored temporarily in the service.

- **Glance - Image:** POST, http://{IP}:{port}/v2/images/{Image_ID}/file

Step 4: Cloud B Authentication

- **Parameters:** Cloud, Tenant, Username, Password
- **Description:** The service prepares and performs the call for authentication by the cloud which the user specified. The service displays the log in details of the user (Username, Tenant, Target Cloud).
- **Keystone - Identity:** POST, http://{IP}:{port}/v2.0/tokens

Step 5: Upload Image

- **Parameters:** Name, Format, Access
- **Description:** After the user fills the parameters and presses the Upload Image, the service will create an empty image with the specified name, update the image by the specified format and access and then upload the data of the snapshot which is temporarily stored in the service. At the success of this process, the service will display the newly created image's details.
- **Glance - Image:** Create Empty Image: POST, http://{IP}:{port}/v2/images
- **Glance - Image:** Update Image: PATCH, http://{IP}:{port}/v2/images/{Image_ID}
- **Glance - Image:** Upload Image Data: PUT, http://{IP}:{port}/v2/images/{Image_ID}/file

Step 6: Keypair

- **Parameters:** User's Selection
- **Description:** This step is for configuring the keypair which will be used for the instance. If the user wants to choose an existing keypair in cloud b, the service will display him the list of his key pairs, but if he wants to create a new one he must provide the name of the new keypair. Also, the user can import a keypair from cloud A and by selecting that, the service will display him the list of his keypairs in cloud A. Depending on the user's choice, the service will select the appropriate keypair for the new instance. In the end, the service displays the selected keypair name.

- **Nova - Compute:** Get Keypair: GET, http://{IP}:{port}/v2/{Tenant_ID}/os-keypairs
- **Nova - Compute:** Import/Create Keypair: POST, http://{IP}:{port}/v2/{Tenant_ID}/os-keypairs

Step 7: Launch Instance

- **Parameters:** Name, Security Group
- **Description:** The user sets the name and security group of the new instance and presses Create Instance. The service then, will create a new instance with the specified name, the image in step 5, the keypair in step 6 and will automatically set the flavor the same as the snapshotted instance had. The security group can be empty or "default" meaning that all the security rules of the snapshotted instance will be copied in the default security group. In case the user specifies a different name, a new security group will be created with that name and again will have the security rules of the snapshotted instance in cloud A. At creation end, the service displays the name and security group of the new instance.
- **Quantum - Network:** Create Security Group: POST, http://{IP}:{port}/v2.0/security-groups
- **Nova - Compute:** Get Default Security Group: GET, http://{IP}:{port}/v2/{Tenant_ID}/os-security-groups
- **Quantum - Network:** Create Security Group Rules: POST, http://{IP}:{port}/v2.0/security-group-rules
- **Nova - Compute:** Create Instance: POST, http://{IP}:{port}/v2/{Tenant_ID}/servers

Step 8: Set Instance IP

- **Parameters:** IP, Create New One
- **Description:** The user can select between an existing IP from the list or create a new one to be allocated to the new instance. The IP list only contains available IPs and after the user's choice the service displays the IP number allocated to the instance.

- **Nova - Compute:** IP List: GET, http://{IP}:{port}/v2/{Tenant_ID}/os-floating-ips
- **Nova - Compute:** Tenant's Pool: GET, http://{IP}:{port}/v2/{Tenant_ID}/os-floating-ip-pools
- **Nova - Compute:** Allocate IP To Tenant: POST, http://{IP}:{port}/v2/{Tenant_ID}/os-floating-ips
- **Nova - Compute:** Allocate IP To Instance: POST, http://{IP}:{port}/v2/{Tenant_ID}/servers/{Instance_ID}/action

Step 9: Overview

- **Parameters:** N/A
- **Description:** As a last step, the service will provide to the full overview of the instance as it is fetched by the cloud's system (not from saved data). The overview includes instance name, access, flavor, image and so on.
- **Nova - Compute:** Instance Details: GET, http://{IP}:{port}/v2/{Tenant_ID}/servers/{Instance_ID}
- **Nova - Compute:** Flavor Details: GET, http://{IP}:{port}/v2/{Tenant_ID}/flavors/{Flavor_ID}
- **Nova - Compute:** Image Details: GET, http://{IP}:{port}/v2/{Tenant_ID}/images/{Image_ID}

4.2.5 Instance Monitoring Tool Implementation

This tool allows the consumer to get the usage data of every instance he owns. The user only needs to provide the information in order to be authenticated by the cloud's system and then the service fetches the usage data of his instances. The usage date includes up-time of each instance, number of cores in use, CPU hours, storage used and other useful metrics. The performed call for getting the usage data is:

- **Nova - Compute:** Tenant Usage Data: GET, http://{IP}:{port}/v2/{Tenant_ID}/os-simple-tenant-usage/{Tenant_ID}

In our implementation, the consumer and the provider of the software must both have access to the account that the VM is deployed after the migration. This gives the ability to control the VM to both the consumer and the provider which essentially means that the user can monitor, use and suspend the VM as per his needs and the provider of the VM can also monitor the usage in order to get some kind of payment, agreed from both, and also suspend the usage of his software in case the user is not honoring their deal. A similar scenario would be if the consumer was the administrator of a private cloud and the provider was given an account there to migrate the VM. Again both have the ability to control and monitor the software. But a more complex scenario would be if the consumer was a government agency or an army one. In that case, giving an account to the provider could be problematic and this scenario would require a more specific, elegant and probably more complex solution.

4.2.6 XML API Call Implementation

Similar to the UI, the xml API call performs calls to the instance migration tool within the service in order to migrate a VM. The difference is that the information used for migration is not gathered along the nine steps of the UI but is given at once in an XML document created by the user. This XML document must follow a specific syntax that is predefined in order to avoid information being missing. The validity of the document is verified against an XML schema and with successful validation the document is passed to the service processing. The service will extract the information and pass it to the migration tool with the same way as the UI does.

The xml API call is focused on the migration process and as a result you cannot monitor your instance through it. Instead the consumer should use the provided UI for monitoring his instance. Furthermore, when a consumer creates a new keypair, the cloud's system provides the keypair file immediately after creation as a result this option is disabled in the API call.

The Xml syntax follows this schema:

<MigrationSpecs>

<CloudAlogIn>

<cloud> Cloud A **</cloud>**

<tenant> Cloud A: Tenant Name **</tenant>**

<username> Cloud A: Username **</username>**

<password> Cloud A: Password **</password>**

</CloudAlogIn>

<SnapshotToCreate>

<InstanceID> Instance To Take Snapshot Of **</InstanceID>**

<SnapshotName> The Name For The Created Snapshot **</SnapshotName>**

</SnapshotToCreate>

<CloudBlogIn>

<cloud> Cloud B **</cloud>**

<tenant> Cloud B: Tenant Name **</tenant>**

<username> Cloud B: Username **</username>**

<password> Cloud B: Password **</password>**

</CloudBlogIn>

<ImageToUpload>

<ImageName> The Name Of The New Image**</ImageName>**

<Format> Format Of New Image**</Format>**

<Access> (Public =) True / False **</Access>**

</ImageToUpload>

<Keypair>

<Name> The Keypair's Name For The New Instance**</Name>**

<PublicKey> The Public Key Of The Keypair (Only when importing) **</PublicKey>**

</Keypair>

<InstanceToLaunch>

<InstanceName> Name Of The New Instance **</InstanceName>**

<SecurityGroup> New Security Group Name (Empty or "default" for default security group) **</SecurityGroup>**

</InstanceToLaunch>

<AllocateIP>

<IP> IP Of The New Instance (insert "new" for creating a new IP for the instance) **</IP>**

</AllocateIP>

</MigrationSpecs>

4.2.7 Apache Tomcat 7

Tomcat 7[43] is an open source software implementation of the Java Servlet and JavaServer Pages technologies provided by the Apache foundation. Despite being free, tomcat is one of the best software for deploying java web applications and we used it to properly test our service locally. In addition, we also installed it in a VM running Ubuntu 12.04 and used it as the server of our service deployed in intellicloud[44]. Launching an instance in intellicloud is explained in detail below and after configuring correctly our new instance we install Tomcat 7 via the Ubuntu console.

[43] http://tomcat.apache.org/
[44] http://cloud.intellicloud.tuc.gr/horizon/auth/login/

4.2.8 Launching An Instance in Intellicloud

Intellicloud is a private cloud located in Technical University of Crete as a result in order to get access you need to ask the cloud's administrators. After getting access to the cloud we launched an instance called S2D_Service_Server with medium flavor meaning it has 4GB RAM, 2 VCPUs and 40GB Disk as displayed in Figures 10, 11, 12. Apart from the instance's name and flavor, we also need to select the image that we want to make an instance of, the security group, keypair and network of our new instance. As soon as the cloud's system launches our new instance, which will look like in Figure 13, we need to configure the security groups and the floating IP in order to be able to connect to our instance. Why we need to configure the floating IP, can be explained by looking at Figure 13 under the IP addresses tab which shows the IPs currently associated with our new instance. Our instance does not have a public IP yet, so we cannot access it remotely over the internet. We can partially resolve this by allocating a new IP to our project (or tenant) and then associating that IP with our instance (Figure 14) which will allow us to access our instance remotely. We mentioned that allocating an IP will partially resolve the problem because we also need to configure the security groups to allow internet traffic to our instance. Inside the security groups we can create rules which consist of IP protocol, range of the port and the source of the traffic. For example, port 22 is used for the ssh authentication that allows us to connect remotely to our instance and port 8080 is used by the Apache Tomcat 7 (Figure 15). After configuring the security groups and floating IP we can access our instance via any ssh and telnet client like PuTTY[45].

[45] http://www.chiark.greenend.org.uk/~sgtatham/putty/

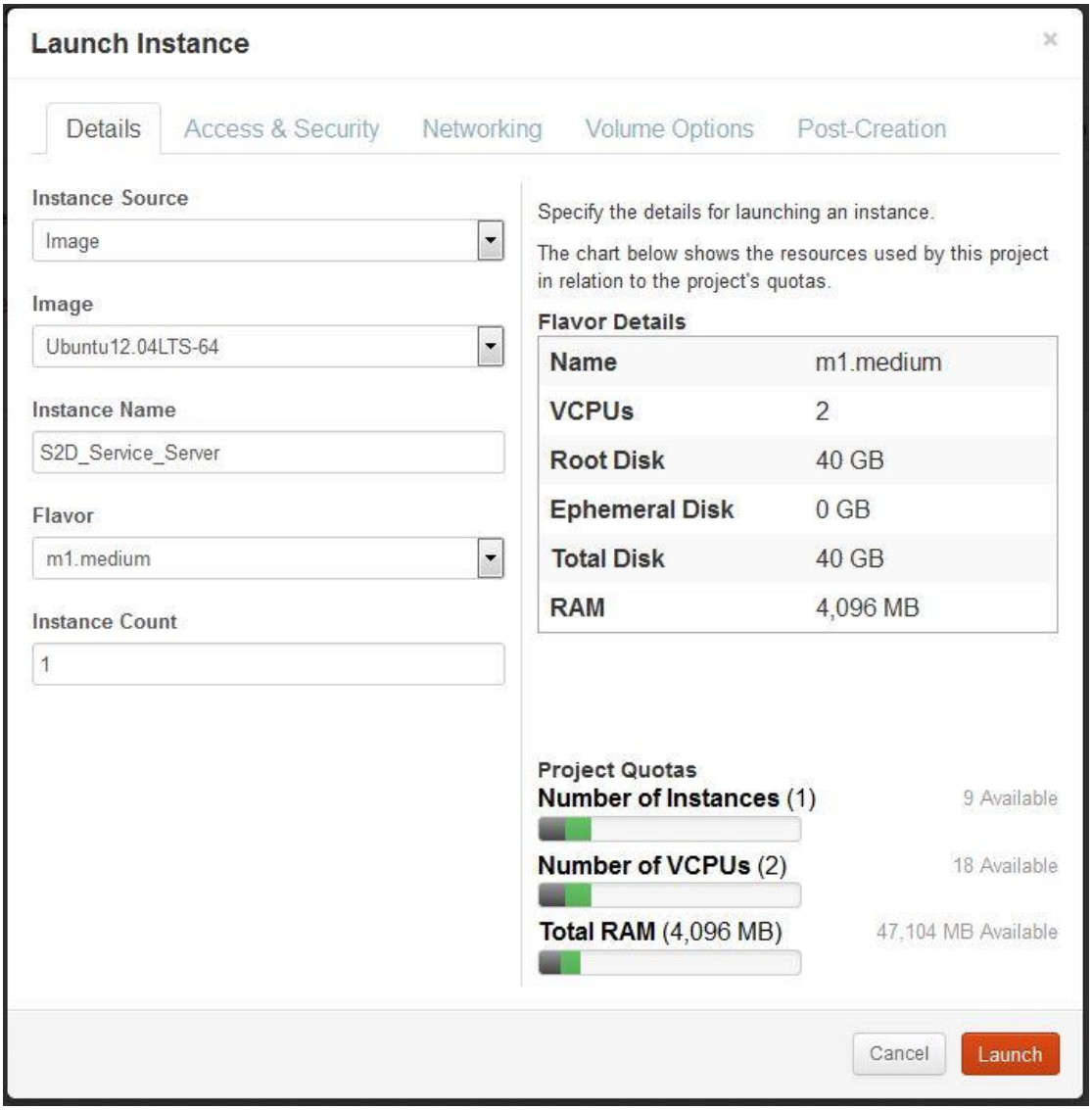

Figure 10 – Launch Instance – Step 1

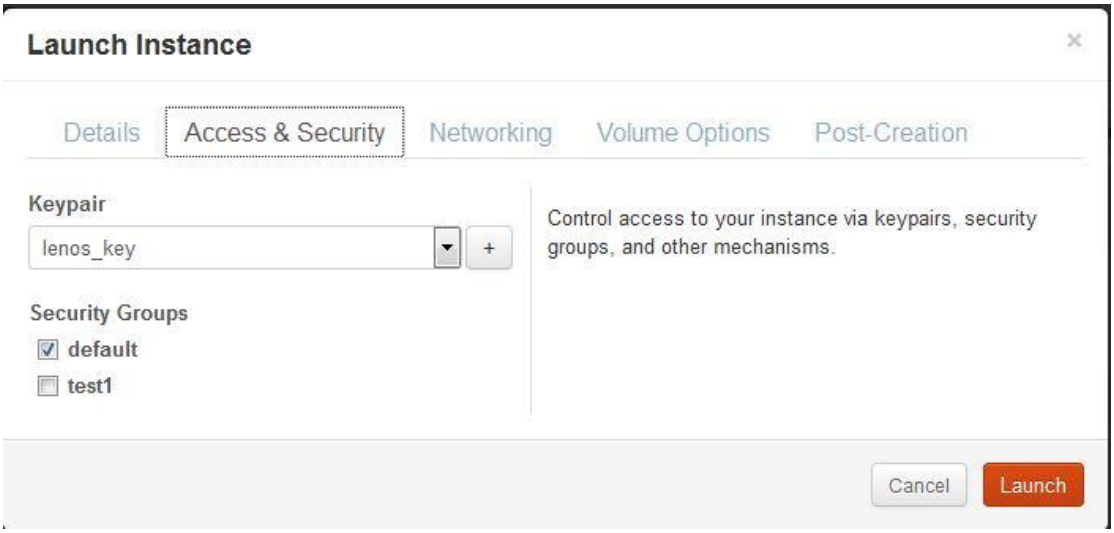

Figure 11 – Launch Instance – Step 2

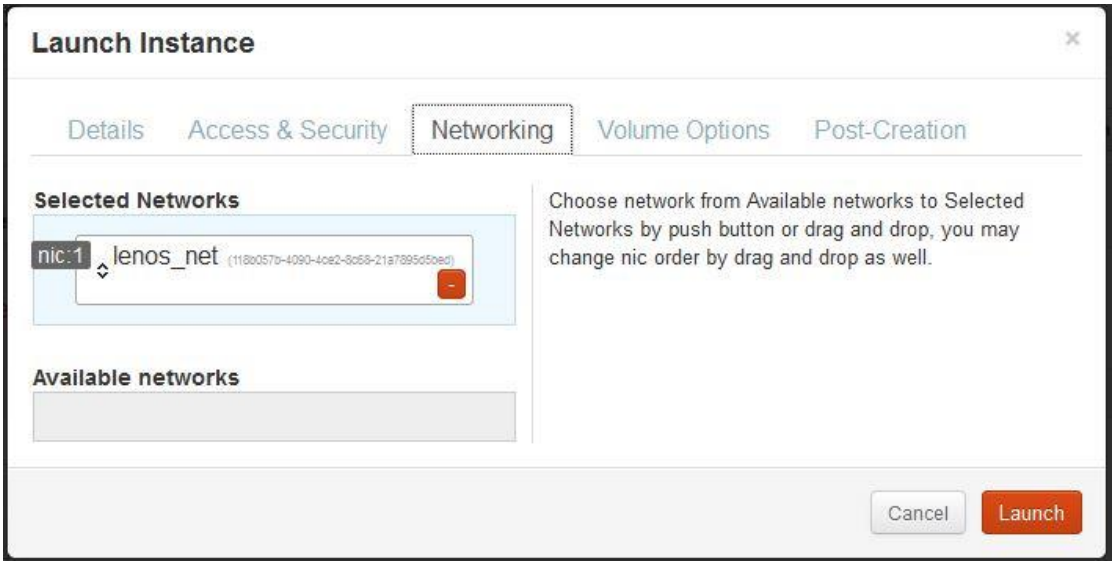

Figure 12 – Launch Instance – Step 3

Figure 13 – New Instance

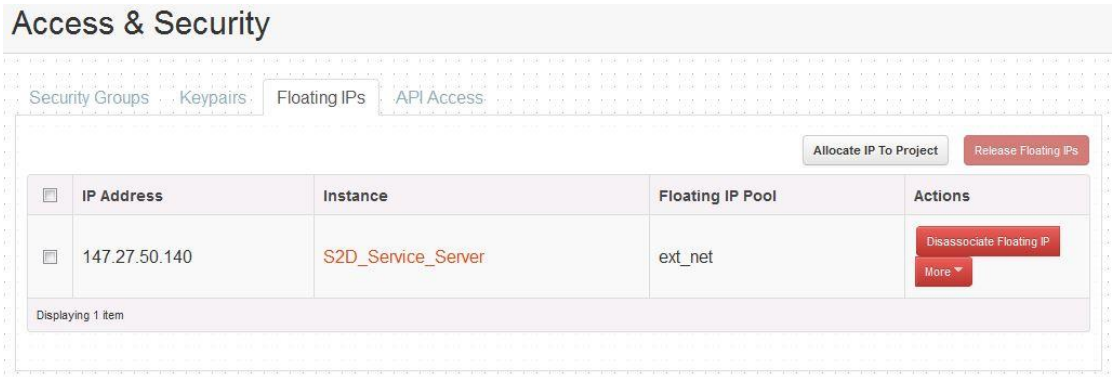

Figure 14 – Configuring Floating IP

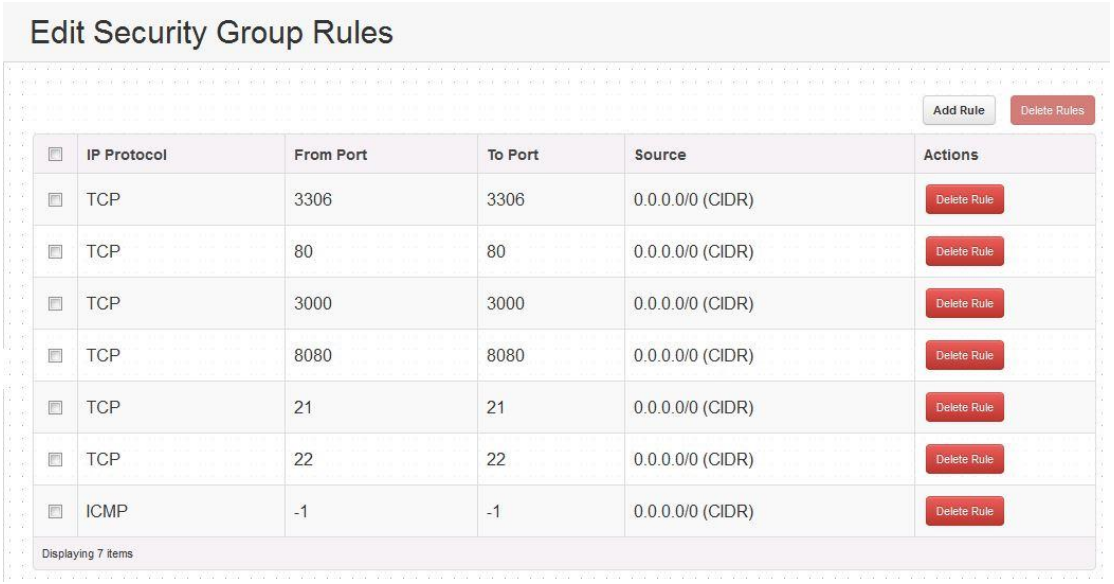

Figure 15 – Configuring Security Group

4.2.8.1 Network Topology Of An Instance

By reviewing the network topology of an instance in the cloud can help us understand where the configuring of security groups and floating IP is used. This topology exist for every instance in the cloud so for better understating we will use the new instance created in the previous section as an example. As displayed in Figure 16, our new instance, S2D_Service_Server, is connected to a virtual network named lenos_net (blue line) and has a local IP. In order to be able to connect to our instance remotely we need to connect it with the ext_net which is the external network. This is done by associating a (public) floating IP through the virtual router lenos_router to our instance. In addition, by setting the rules in the security group we are saying to the

virtual router which internet traffic is allowed to pass to our instance from external sources. So a data flow from an outside source will come from the external network (green line) to our virtual router, there it will be decided to which instance it is supposed to go through the IP to which the date was sent and then the rules inside the security group will determine if the data are allowed to pass through our local network (blue line) to their destination which in this example is the S2D service server.

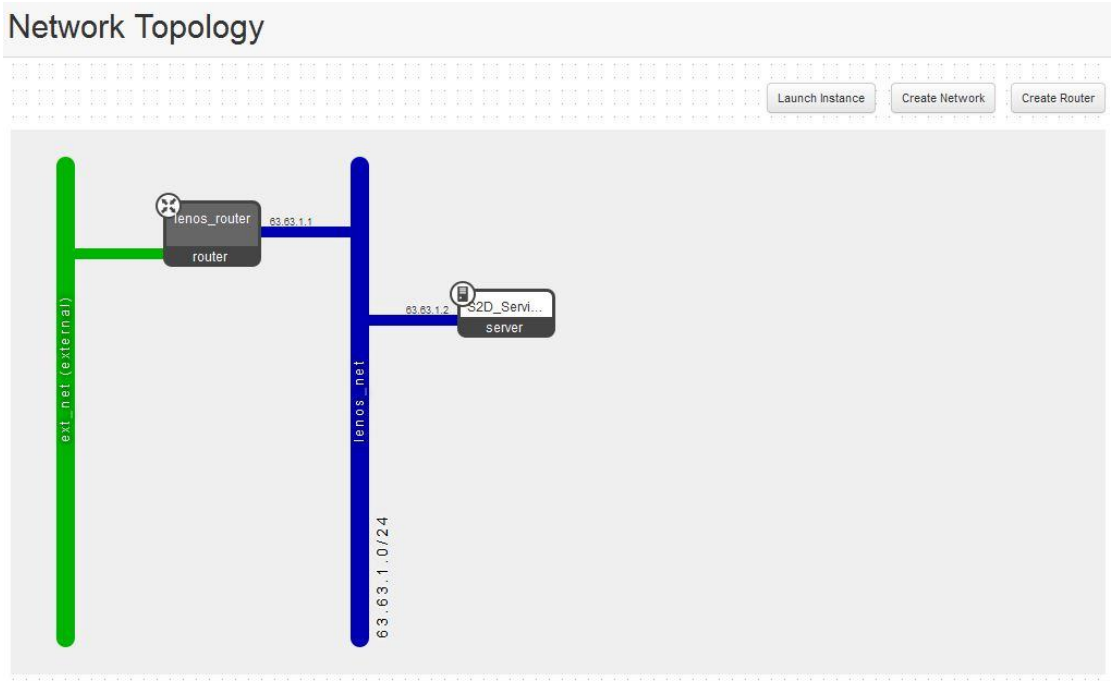

Figure 16 – Network Topology

4.2.8.2 Implementation Details

Our service allows easy migration of VMs between Openstack environments and thus achieving interoperability between Openstack – based clouds. VMs can be transferred while running without any downtime and the process is easy thanks to a wizard type GUI. What's more, when the VM is transferred it does not need to be fully reconfigured. In an effort to make the migration as automatized as possible some configurations are taken directly from the source cloud such us the VM's flavor and security group rules and other configurations are available through the service such as setting floating IP and keypair.

Furthermore, the migration can be performed through an XML document making the migration fully automatized as the user only needs to provide the required information and the rest are taken care from the service. The information needed can be found through the two cloud's dashboard such as security groups and keypair names. Also, the XML itself is self-explained and human – readable (definition of XML) as can be seen in section 4.2.6.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Looking back, the problem we tried to address was the cloud's lack of use by some applications fields which possess sensitive data and they are restricted by them or simply concerned for their data privacy. A solution that could eliminate these concerns or restrictions that we proposed in this work builds-upon the idea of reverse-cloud approach. In simple words, the solution revolves around the idea that application fields which cannot move to public clouds and take advantage of its offerings can still enjoy some of those offerings by transferring the software they want to use in their own private clouds. This merging of public and private cloud was named as a hybrid cloud.

Following the approach of hybrid cloud technology, we developed a service which could transfer and deploy a VM along with its software between two Openstack clouds. This works by using the REST API provided by the Openstack system which allows the service to perform all the necessary actions required by this process. Our proposed S2D (Software to Data) service implementation creates a "backup" or snapshot, as it is called, which contains all the information needed to deploy an identical VM containing the same operating system and software. Then it transfers the snapshot and deploys it to its destination cloud. When this process is done, we have the software that was located on a public cloud and we were unable to use now running on our own private cloud and we are using it without exposing our data to the outside world.

We hope that S2D services (such as our proposed one in this work) will pave the way to application fields with strong privacy or security concerns so that they can still benefit from the advantages of clouds and at the same time meet their restrictions or satisfy their concerns regarding the privacy of their data.

5.2 Future Work

Having the potential for expansion and upgrading, our service is a good candidate for continuing the original work and we will list some of the possible work which could be done in the future [22].

5.2.1 Multiple Migrations At Once

As it stands right now, migration of multiple VMs at once is not possible. A consumer can migrate multiple VMs only by going through the process multiple times. Even if the time that the services takes in order to migrate a VM stays the same, the time that the user takes to provide the information of each VM is significant and by eliminating it would result in a faster migration. This could be achieved by converting all the variables which hold the information of the VM in to arrays and repeat the migrations steps for each raw of the array. Furthermore, after enabling this functionality to work with the UI, the only thing left would be to upgrade the XML schema to support multiple elements in the XML document.

Migrating multiple VMs produces another challenge. For example, if you have two VMs that communicate with each other through their IP, migrating them would result in a change of their IPs and thus break the communication between them. A solution to this could result by carefully designing the software to be easily configurable without requiring code changes through a configuration file. Another idea for a possible solution to this would be the developing of a system similar to the Domain Name System[46] (or DNS) used in World Wide Web[47]. More detailed, each service or application housed in a VM installed in a cloud somewhere should register its IP with a unique name like is done for webpages where one IP is register for one domain name. This would result in the use of that service or application by its unique name and thus enabling the owner of that service to migrate it anywhere without breaking anyone else's service which communicates with his, only by changing the IP registered to its unique name. For example, if everyone that uses Keyrock[48] as his authentication manager, used it with the unique name Keyrock IDM, the owner of Keyrock could migrate his service anywhere without breaking the services of others

[46] http://el.wikipedia.org/wiki/Domain_Name_System
[47] http://en.wikipedia.org/wiki/World_Wide_Web
[48] http://catalogue.fiware.org/enablers/identity-management-keyrock

only by registering the new IP with the Keyrock IDM name as opposed to making everyone change the code of their service anywhere they use the old IP of Keyrock.

5.2.2 Alternative Ways Of Monitoring

Monitoring the use of your software when transferred to another cloud could be done by several ways. One of them, the one we use, is when both consumer and provider have access to the VM where the software is housed and thus both have control over it. A different way of monitoring, not the most secure though, would be if the provider did not have access to the VM but was reported on a timely loop by the consumer about the usage of its software. A better, but complex, way of monitoring would be if the software was pre-designed to send usage information back to the provider or by designing the software to work for a prepaid time subscription. Both of the last two ways of monitoring are more complex and require additional components such us a server to send back the usage information or validate that the prepaid subscription is still valid. Also, the software itself would require additional developing to support such functionality.

5.2.3 XML API Functionality

Possible ways to enrich the XML API functionality could be to implement the support of monitoring or the ability to create a new keypair through an XML API call. Supporting monitoring could become a complex solution due to the fact that monitoring could be done with many ways, simple or complex. But as the service stands right now, monitoring through the XML API could be resolved by getting the usage data from the cloud's system in JSON format, converting them to XML and returning them to the user. Besides this, allowing the user to create and download a new keypair by using the XML API could be a possible future work. This could be resolved by returning both the XML response as well as the new keypair. The only restriction here is that the new keypair must be downloaded right and that the download only occurs ones, at the begging of every keypairs lifecycle.

References

[1] Schubert L, Jeffery K, Neidecker-Lutz B (2010) "The Future of Cloud Computing – Opportunities for European cloud computing beyond 2010", European Commission, http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf

[2] Ian Foster, Yong Zhao, Ioan Raicu, Shiyong Lu, "Cloud Computing and Grid Computing 360-Degree Compared", http://arxiv.org/ftp/arxiv/papers/0901/0901.0131.pdf

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing", http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf

[4] Peter Mell, Timothy Grance, "The NIST Definition of Cloud Computing", http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf

[5] Cloud Computing Presentation, Stelios Sotiriadis

[6] Jon Kuroda, "Cloud Computing - What Is It (Good For)? ", http://www.eecs.berkeley.edu/~jkuroda/talks/clouds/#%281%29

[7] Tom Pfeifer, Stefan Covaci, "Active Protection of Patient Data by Reverse Cloud Approach"

[8] Open Networking Foundation, Mitch Auster (Editor), Nabil Damouny, John Harcourt, "OpenFlow™-Enabled Hybrid Cloud Services Connect Enterprise and Service Provider Data Centers", https://www.opennetworking.org/

[9] Christoph Thuemmler ,Thomas Magedanz, Thomas Jell, Julius Mueller, Stefan Covaci, Stefano de Panfilis, Armin Schneider, Klinikum Rechts der Isar, Anastasius Gavras, "Applying the Software-to-Data Paradigm in Next Generation E-Health Hybrid Clouds"

[10] Stelios Sotiriadis, Euripides G.M. Petrakis, Stefan Covaci, Paolo Zampognaro, Eleni Georga, Christoph Thuemmler, "An architecture for designing Future

Internet (FI) applications in sensitive domains: Expressing the Software to data paradigm by utilizing hybrid cloud technology"

[11] Charles David Graziano, "A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project", http://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=3243&context=etd

[12] VMware, Inc., "Virtualization Overview", http://www.vmware.com/pdf/virtualization.pdf

[13] Atul Jha, Johnson D, Kiran Murari, Murthy Raju, Vivek Cherian, Yogesh Girikumar, "OpenStack Beginner's Guide", http://arccn.ru/knowledge-base?pdf=50f6707855f16.pdf

[14] Andy Edmonds, Thijs Metsch, "Open Cloud Computing Interface - RESTful HTTP Rendering", https://www.ogf.org/documents/GFD.185.pdf

[15] Amid Khatibi Bardsiri, Seyyed Mohsen Hashemi, "QoS Metrics for Cloud Computing Services Evaluation", http://www.mecs-press.org/ijisa/ijisa-v6-n12/IJISA-V6-N12-4.pdf

[16] Openstack org., "OpenStack Operations Guide", http://docs.openstack.org/openstack-ops/content/

[17] Mark Massé, "REST API Design Rulebook", http://shop.oreilly.com/product/0636920021575.do

[18] Roy Thomas Fielding, "Architectural Styles and the Design of Network-based Software Architectures", http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[19] Stelios Sotiriadis, Nik Bessis, Pawel Gepner, Nicolas Markatos, "Analysis of requirements for virtual machine migration in dynamic clouds"

[20] Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh , "The Design and Implementation of Zap: A System for Migrating Computing Environments", http://www.cs.cmu.edu/~sosman/publications/osdi2002/osdi2002_zap.pdf

[21] Mitch Auster, Nabil Damouny, John Harcourt, “OpenFlow-Enabled Hybrid Cloud Services Connect Enterprise and Service Provider Data Centers”, https://www.opennetworking.org/solution-brief-openflow-enabled-hybrid-cloud-services-connect-enterprise-and-service-provider-data-centers#page-top

[22] Vakanas, L., Sotiriadis, S. and Petrakis, E. (2015) "Implementing the Cloud Software to Data approach for OpenStack environments", Adaptive Resource Management and Scheduling for Cloud Computing, Held in conjunction with PODC-2015, Donostia-San Sebastián, Spain, on July 20th, 2015, http://sotiriadis.gr/s2d.pdf