TECHNICAL UNIVERSITY OF CRETE

School of Electronic and Computer Engineering (ECE)

Virtual Machine Deployment and Migration on Heterogeneous Cloud Platforms

by

Dimitrios G. Kargatzis

Dissertation Thesis

Thesis Committee:

Professor Euripides Petrakis

Assistant Professor Vasilis Samoladas

Research Collaborator Dr. Stelios Sotiriadis

Chania, 73100, Greece

Abstract

Cloud computing offers an innovative business model for organizations to adopt IT services at a reduced cost with increased reliability and scalability. Adopting a cloud solution means binding to a specific platform and vendor, using specific protocols, standards and tools of the cloud and finally, running into a vendor lock-in situation. The fear of vendor lock-in is often cited as a major impediment to cloud service adoption. If the provider decides to raise its prices or change its security policies, the customer may have to consider to move his workloads to another provider. In this work we focus on the automatic migration requirements in Openstack systems and as a use case we present a mechanism for Virtual Machine (VM) migration (or of their running instances) between Openstack and KVM virtualization and another cloud platform that runs a different Virtualization engine (Stratogen and VMware is our case). The standards approach is to freeze the running instance of a VM and restart it under the new environment. We examined the requirements for successful migration and the conclusion is that migration is not always fully automatic as it might need lots of parameter tuning depending on differences in the type of virtualization engines used at source and target environments (a task that can be performed by a specialized expert). To alleviate the requirement of human intervention in the loop, we suggest using containers as the underlying virtualization technology. We propose a mechanism that implements migration using containers in a few steps and we run a series of experiments to show proof of concept. As a conclusion, the later technology proved to be feasible and more promising although it is still not fully supported by infrastructure (operating system) tools that allow migration independent of the state of the underlying operating system kernel at the time of transfer.

Περίληψη

Το υπολογιστικό Νέφος προσφέρει ένα καινοτόμο επιχειρηματικό μοντέλο στους οργανισμούς για να υιοθετήσουν υπηρεσίες πληροφορικής μειώνοντας το κόστος, με μεγαλύτερη αξιοπιστία και επεκτασιμότητα. Επιλέγοντας μία συγκεκριμένη λύση στο υπολογιστικό Νέφος η επιχείρηση είναι άμεσα συνδεδεμένη με τον συγκεκριμένο πάροχο, τα πρωτόκολλα, τα πρότυπα και τα εργαλεία που χρησιμοποιεί με αποτέλεσμα να μη μπορεί να καταφύγει σε διαφορετική λύση στο μέλλον. Αυτός είναι ο κυριότερος λόγος για τον οποίο οι επιχειρήσεις δεν υιοθετούν υπηρεσίες υπολογιστικού Νέφους. Αν ο πάροχος αλλάξει την πολιτική ασφαλείας ή αυξήσει τις τιμές για τους πόρους που παρέχει, η επιχείρηση πρέπει να έχει τη δυνατότητα να μεταφέρει τις εργασίες της σε άλλον πάροχο. Σε αυτή τη δουλειά, εστιάσαμε στις απαιτήσεις της αυτοματοποιημένης μεταφοράς των εργασιών μίας επιχείρησης από έναν πάροχο σε κάποιον άλλον και συγκεκριμένα παρουσιάσαμε ένα μηχανισμό για τη μεταφορά εικονικών μηχανών από περιβάλλον Openstack σε VMware (Stratogen). Η βασική προσέγγιση είναι να πάρουμε ένα στιγμιότυπο μαις εικονικής μηχανής από ένα περιβάλλον και να το μεταφέρουμε σε ένα άλλο περιβάλλον. Εξετάσαμε τις απαιτήσεις για την επιτυχημένη μεταφορά και το συμπέρασμα είναι ότι η διαδικασία δεν είναι πλήρως αυτοματοποιημένη πάντα, απαιτώντας αρκετή παραμετροποίηση ανάλογα με τον τρόπο που γίνεται η εικονοποίηση στο κάθε περιβάλλον (μία διαδικασία που απαιτεί εξειδικευμένες γνώσεις). Για να γίνει όσο το δυνατόν περισσότερο αυτοματοποιημένη η διαδικασία, προτείνουμε τη χρήση ¨containers" στους παρόχους. Σχεδιάζουμε ένα μηχανισμό που υλοποιεί μεταφορά με τη χρήση ¨containers" σε λίγα μόνο βήματα και τρέχουμε μία σειρά από πειράματα για να αποδείξουμε αυτή την παραδοχή. Συμπεραίνοντας, η κανούργια τεχνολογία φαίνεται να ειναι πολλά υποσχόμενη αλλά ακόμα δεν υποστηρίζεται πλήρως από τα εργαλεία που στοχεύουν στη μεταφορά ανεξάρτητα του λειτουργικού συστήματος.

Contents

Chapter 1

Introduction

1.1 Cloud Computing

Cloud computing is a type of Internet-based computing that provides shared computer processing resources, services and data to computers and other devices on demand. Nowadays, 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 time. Possessing the hardware and software for their operation is not the optimal solution in many cases in terms of capital investment or in terms of human resources required for maintenance. In these cases, cloud computing offers a pay-per-use solution thus minimizing investment in terms of capital and human resources. Cloud technology can be also used in many other ways in order to provide a suitable and affordable solution in many use scenarios by exploiting features such as:

Elasticity: The ability of cloud computing adjusts 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 adjust his computing resources, without human interaction.

Broad network access: The ability to use the cloud over the internet, any time and from anywhere. The interaction between user and cloud is realized by means of Web 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.

Measurable services provision: A business model is applied and ensures clouds' sustainability. A business model applies a pricing model which 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 [1], 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 [2]:

Pay per use: The consumer can pay for exactly the resources he/she uses, meaning that he/she 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/she 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 lease computing power and storage space without risking with the procurement of expensive hardware, software or paying 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[1],

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

Google Drive[2], iCloud[3] etc.). The applications that are using cloud computing technology are many and their number keeps growing.

1.2 Cloud Computing for Enterprises

Cloud computing is becoming a game changer for Small-Medium Enterprises (SMEs) by offering scalable infrastructure and capabilities available as services [3]. In an enterprise that needs complex and expensive IT technology to support its business processes, cloud provides an attractive alternative by which the compute resources are made available at a fraction of the cost and without to get IT services without being concerned with the details of how this is done.

Adopting cloud computing can save money, but it is important to choose the right cloud hosting service and the right cloud solution for your business needs. Determining which provider is best for a business depends largely on what you the business needs in terms of services, time of use, degree of required control over hardware and software for application development or hosting. Additional factors that may affect a decision are provider's current security accreditations in which case an evaluations of the encryption options the company supports is mandatory prior to taking a decision to select a provider.

1.3 Problem Definition

Adopting a cloud solution means binding with the specific protocols, standards and tools of the cloud provider (vendor lock-in). Enterprises may leave the opportunity open to migrate to different clouds. If the provider decides to raise its prices or change its security policies, the customer may consider to move his workloads to another provider. However, the complexity of the problem depends on the way a cloud

[2] https://www.google.com/drive/
[3] https://www.icloud.com/

provider tweaks its infrastructure (heterogeneity), making migration difficult and expensive. Basic migration constraints between heterogeneous cloud providers are architecture, hypervisors, container formats and disk formats.

1.4 Proposed Solution

There are many platforms and tools to migrate from a cloud provider to another with the same infrastructure. In this work we focus on the problem of migrating workloads between different cloud infrastructures. We propose an implementation that will transfer and deploy an instance or a service from Openstack[4] to VMware[5]. The implementation consists of three modules, the first is responsible for downloading the instance from source cloud provider, the second is responsible for solving migration constraints and the last one is responsible for uploading and deploying the instance on the target cloud provider. We focus on the automatic migration requirements in Openstack systems and as a use case we present a mechanism for Virtual Machine (VM) migration (or of their running instances) between Openstack and KVM virtualization and another cloud platform that runs a different Virtualization engine (Stratogen and VMWARE is our case). The standards approach is to freeze the running instance of a VM and restart it under the new environment. We examined the requirements for successful migration and the conclusion is that migration is not always fully automatic as it might need lots of parameter tuning depending on differences in the type of virtualization engines used at source and target environments (a task that can be performed by a specialized expert). To alleviate the requirement of human intervention in the loop, we suggest using containers as the underlying virtualization technology. We propose a mechanism that implements migration using containers in a few steps and we run a series of experiments to show proof of concept. As a conclusion, the later technology proved to be feasible and more promising although it is still not fully supported by infrastructure (operating system)

[4] https://www.openstack.org/

[5] https://www.vmware.com/

tools that allow migration independent of the state of the underlying operating system kernel at the time of transfer.

Chapter 2

Background and Related Work

2.1 Cloud Computing Architecture

Cloud computing architecture refers to the layers and components required for cloud computing. Cloud architecture can be divided into 4 general layers that map to the available business models: the hardware layer, the infrastructure layer, the platform layer and the application layer (Figure 1). For research and developing purposes, each layer can be divided into sub layers. Infrastructure is the lowest layer and is a means of providing processing, storage, networks, and other fundamental computing resources as standardized services. Cloud providers' clients can deploy and run operating systems and software for their underlying infrastructures. The platform layer provides higher abstractions and services to develop, test, deploy, host, and maintain applications in the same integrated development environment. This layer provides a runtime environment and middleware to deploy applications using programming languages and tools the cloud provider supports. The application layer is the highest layer and features a complete application offered as a service [4].

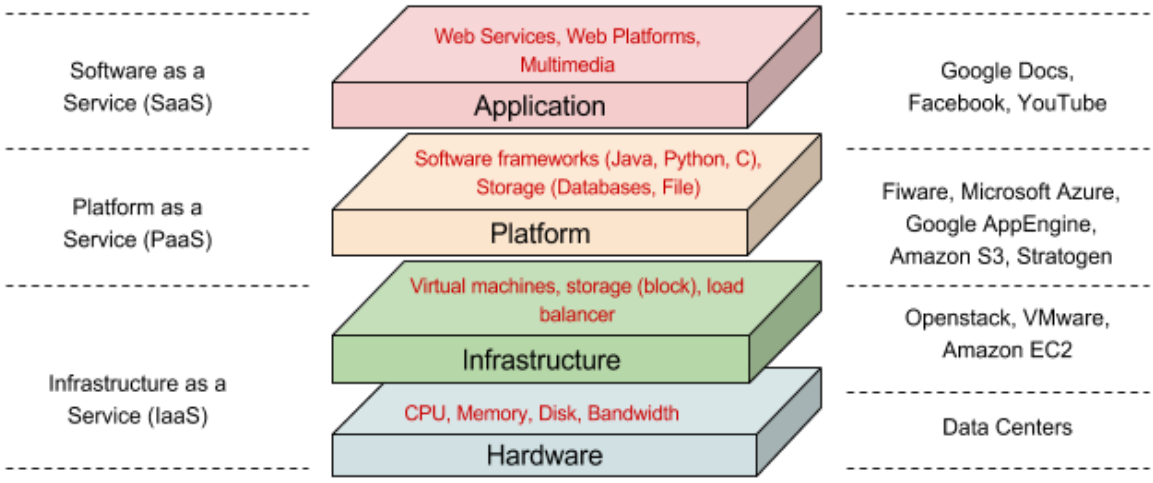

Figure 1 – Cloud Computing Layers

2.1.1 Service Models

Cloud computing services are offered in three different service models:

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

The consumer leases 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[6], SQL Azure.

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

The cloud provider provides a software platform (with all basic tools and software services such as Generic Enablers in the case of FIWARE-LAB[7]) for deploying their applications. The consumer gets a functional virtual machine with an operating system of his choice and he can use it for deploying services without worrying about upgrades or maintenance. Examples of such services are Google App Engine[8],

[6] https://aws.amazon.com/s3/
[7] https://account.lab.fiware.org/
[8] https://cloud.google.com/appengine/

Windows Azure[9] (Platform) and FIWARE-LAB.

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

This is the most usable cloud service. It allows a consumer to use services provided by a 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.

2.1.2 Virtualization

Virtualization provides a layer of abstraction between the hardware and the software. Hardware or platform virtualization refers to the creation of a virtual machine (VM) that acts like a real computer with an operating system. The main difference from the tradition computer is that it allows definition of multiple VMs with different operating systems over the same hardware. The host machine is the actual machine on which the virtualization takes place and the guest machine is the virtual machine created by hypervisor (Virtual Machine Manager).

A cloud provider typically has a specific amount of computing resources to share but virtualization enables optimal sharing and use of resources among 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.

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. Also known as Virtual Machine Manager (VMM), is the program that allows multiple

[9] https://azure.microsoft.com/

operating systems to share a single hardware host [5]. Each virtual machine appears to have the host's processor, memory and other resources all for itself. However, it is actually controlling the host processor and other resources, allocating what is needed to each operating system and making sure that the guest machines (virtual machines) can't disrupt each other. It can be one of two types (Figure 2). This type of hypervisor is referred to as type 1 hypervisor also known as native or bare metal hypervisor. Typically, there is a "light" software operating directly on the system hardware to control the hardware and to manage guest operating systems. Instead, type 2 hypervisors, need a full host operating system to run onto but this affects the overall performance of the hypervisor.

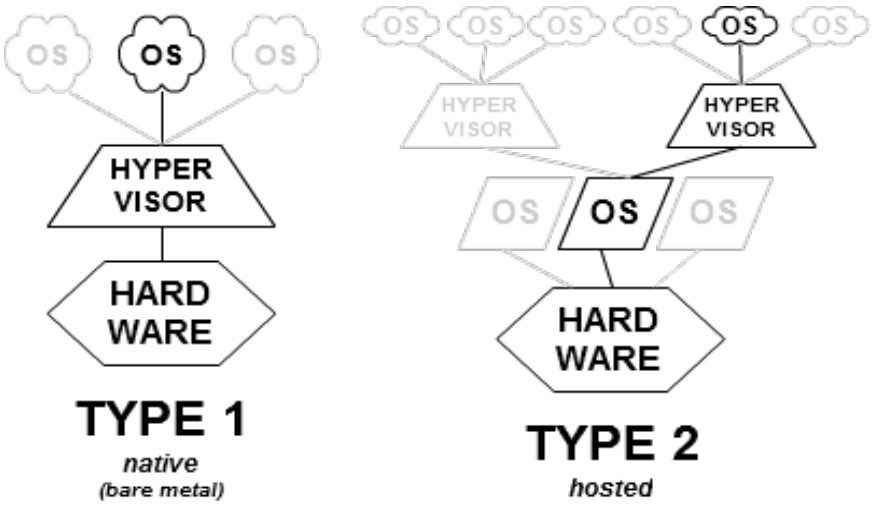

Figure 2 – Hypervisor types

The following are the main hypervisors of choice in use.

- **KVM**

  KVM[10] runs on most Linux distributions today and is perceived as the default hypervisor to be used in all virtualization and cloud products offered by most

[10] https://www.linux-kvm.org/page/Main_Page

Linux vendors, probably making it one of the most widely used hypervisors in the world. KVM is an open source hypervisor.

- **XEN**

Xen[11] an open source hypervisor, The project started in University of Cambridge, then moved to Xensource, then acquired by Citrix, and finally to its current place of residence – the Linux Foundation. Amazon Web Services (AWS)[12] is the biggest cloud provider that uses Xen today. Xen offers a number of advantages over KVM such as the efficiency of paravirtualization (an efficient and lightweight virtualization technique), which exceeds what is available in KVM due to the closer access Xen has to the physical hardware, and the fact that it is a more mature product. Xen is not actually part of the Linux Operating system, whereas KVM is part of the Linux kernel.

- **ESXI**

ESXI[13] is a product of VMware. It is the feature-rich hypervisor that many enterprises use is ESXI (vSphere). It supports any operating system, be it Linux or Windows, with almost any kind of distribution that you could imagine covered by ESXi.

- **Hyper-V**

Hyper-V[14] is a Microsoft product. However, there are free versions available but with many limitations built-in. Hyper-V and Microsoft have always feud with VMware. Over the past few years, they have managed to cut away from VMware's market share by providing a native hypervisor that does most of what vSphere (a VMware product) can do and at a more attractive price.

2.1.4 Image

An image is a virtual hard disk file that is used as a template for a virtual machine (VM). An image is a template because it doesn't have the specific settings that a

[11] https://www.xenproject.org/
[12] https://aws.amazon.com/
[13] http://www.vmware.com/products/vsphere-hypervisor.html
[14] https://technet.microsoft.com/en-us/library/mt169373(v=ws.11).aspx

configured virtual machine has, such as the computer name, network, or user account settings. In simple words it is a software implementation of a machine (i.e. a computer) that executes programs like a physical machine. Image may contain a boot loader, an operating system and a root file system that is necessary for starting an instance, data files and applications just like your personal computer.

Snapshot is a "point in time image" of a virtual guest operating system (VM). That snapshot contains an image of the VMs disk, RAM, and devices at the time the snapshot was taken [6]. With the snapshot, you can return the VM to that point in time. All changes made after the snapshot was taken may be based on that snapshot information (they are incremental changes). You can take snapshots of your VMs, no matter what guest OS you have and the snapshot functionality can be used for features like performing image level backups of the VMs without ever shutting them down. Snapshots can be taken in just about every virtualization platform available.

2.1.4.1 Disk Formats

The disk format of a virtual machine image is the format of the underlying disk image.

RAW format has the advantage of being simple and easily exportable to all other emulators [7]. It has no metadata associated and it is as fast as possible.

ISO was created by the International Standards Organization's 9660 standard. An ISO archive is a CD/DVD image. Creating a package as an ISO image allows you to install a pre-configured virtual machine image using a CD ROM drive [8].

VHD (**Virtual Hard Disk**) is another file format which represents a virtual hard disk drive (HDD). It may contain what is found on a physical HDD, such as disk partitions and a file system, which in turn can contain files and folders. It is typically used as the hard disk of a virtual machine [9]. It is supported by Hyper-V and Xen

hypervisors and was initially used by Microsoft Azure and thereafter by Rackspace[15] and other cloud providers.

VMDK (Virtual Machine Disk) is one of the disk formats used in the Open Virtualization Format (OVF) for virtual appliances. Initially developed by VMware for its virtual appliance products like VMware Workstation[16] or VirtualBox[17].

Qcow used by a virtual machine monitor (QEMU[18]). An image format like Qcow has the largest overhead compared to raw images, when it needs grow, the image. This allows for smaller file sizes than raw disk images, which allocate the whole image space to a file, even if parts of it are empty. Qcow2 [10] is an updated version of the Qcow format.

2.1.4.2 Container Formats

There are several container formats for packaging and distributing a pre-configured virtual machine image (virtual appliance) to run on a hypervisor. The container format refers to whether the virtual machine image is in a file format that also contains metadata about the actual virtual machine such as architecture and hypervisor type [11].

Bare indicates there is no container or metadata envelope for the image. It is safe to specify bare as the container format if you are unsure about image metadata.

Open Virtualization Format (OVF) describes an open, standard, secure, portable, efficient and extensible format for the packaging and distribution of software to be run on VMs. It is not tied up with any particular hypervisor. OVF consists of several files placed in one directory and contains exactly one OVF descriptor (XML) which describes metadata about virtual machine image, such as name, hardware

15 https://www.rackspace.com/
16 http://www.vmware.com/products/workstation.html
17 https://www.virtualbox.org/
18 http://www.qemu-project.org/

requirements and references to the other files in the OVF package. May typically contain one or more disk images and optionally certificate files. Furthermore, OVA is a tar file with the OVF directory inside [12].

Amazon Machine Image (AMI) provides the information required to launch an instance. AMI[19] includes a template for the root volume for the instance (an operating system and applications) and a block device mapping that specifies the volumes to attach to the instance when it's launched.

Docker container format isn't a format for packaging and distributing virtual machine images. Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run, code, runtime, system tools, system libraries [15]. It's a different architectural approach that is not tied with any specific infrastructure.

2.1.5 Cloud Hosting

Openstack and VMware are the industry's most popular full infrastructure suite that deliver comprehensive virtualization, management, resource optimization, application availability and operational automation capabilities is an integrated offering.

Some Cloud Hosting enterprises build their environment with Openstack and VMware infrastructures to provide compute capacity in the cloud. Furthermore, there are Cloud Hosting enterprises which build their own software for virtualization, management and resource optimization to provide compute capacity in the cloud, such as Amazon[20], Microsoft and Google[21].

The following figure (3) shows the standards of most popular Cloud Providers.

[19] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html
[20] https://aws.amazon.com/ec2/
[21] https://cloud.google.com/

Provider	Disk Format	Container	Hypervisor	CLI (command line tools)	Rest API	XML/JSON
Amazon EC2	AMI	AMI	Xen	✓	✘	XML
Microsoft Azure	VHD	BARE	Hyper-V	✓	✓	XML
Google Compute Engine	RAW, AMI, VDI	BARE	KVM	✓	✓	JSON
Rackspace (Openstack)	VHD	BARE	Xen	✓	✓	XML/JSON
Fi-ware (Openstack)	QCOW2, AMI	OVF, AMI, BARE	KVM, Xen	✓	✓	JSON

Figure 3 – Most popular Cloud Providers standards

2.1.6 Image Conversion Tools

QEMU is a generic and open-source machine emulator (also called software virtualizer, emulate the complete hardware in software) and virtualizer (also called hardware virtualizer, both the host and guest share some of physical hardware). It also provides a set of tools to create and convert disk images and supports many image file formats that can be used with virtual machines as well as with any of the tools (like qemu-img) [13], including

- QCOW2 (KVM, Xen)
- Raw
- VDI (VirtualBox)
- VHD (Hyper-V)
- VMDK (VMware)

OVF Tool is a command line utility that helps users to generate OVA packages (is a part of OVF standard and contains all the files of a virtual machine) and convert formats supported by vSphere[22], vCloud[23] Director, VMX hypervisors or OVF to any format supported by the hypervisors above [8].

[22] http://www.vmware.com/products/vsphere.html

[23] http://www.vmware.com/products/vcloud-suite.html

Supported File and Package Types for OVF Tool Input and Output

- OVF
- OVA
- VMX
- VMDK
- ISO

2.2 Containers in Cloud Computing

Containers are an attempt to abstract applications from the underlying OS to enable faster development and easier deployment. And unlike virtual machines, containers execute directly on the host OS, sharing the kernel with other containers. Container manager allocates resources to containers dynamically and they are able to operate with the minimum amount of resources to perform the task they were designed for, that means you can buy less hardware, build or rent less data center space.

2.2.1 Container-based virtualization

Container-based virtualization, also called operating-system-level virtualization, is an approach in which the virtualization layer runs as an application within the operating system. In traditional hardware virtualization, a hypervisor (either software or bare metal) can run one or more guest operating systems. Each operating system acts as if it is in control of the entire machine. In container-based virtualization approach, the operating system's kernel runs on the hardware node with several isolated guest containers. Unlike with hypervisors, there is no emulation layer, just a thin layer controls resource access.

The key point is that hypervisors are an abstraction at the hardware level and containers are an abstraction at the OS kernel level. Anything you can do with hardware, you can do with a hypervisor but some things are incredibly hard to do with hardware, like memory hot unplug. With containers we actually virtualizing at the level of the kernel, that means in a containerized system everything that runs shares

the same kernel. Resource management becomes easy because a single kernel is managing the memory in the system.

The following figure (4) shows the architecture differences between hardware and container-based virtualization.

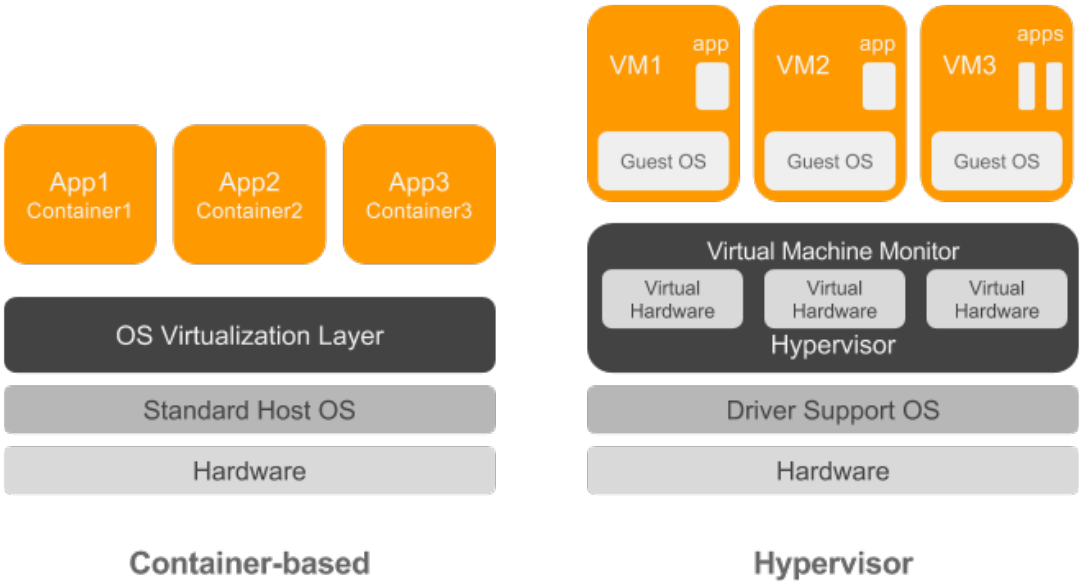

Figure 4 – Virtualization Types

2.2.2 Docker

Is an open-source project that automates the deployment of applications inside software containers. Docker[24] kicked off the container trend by standardizing the packaging and APIs for Linux containers, providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux. Docker uses the resource isolation features of the Linux kernel to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines. Docker containers wrap a piece of software in a

[24] https://www.docker.com/

complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries [15]. You may package your application into standardized unit to run on all major Linux distributions, on Microsoft Windows and on top of any infrastructure.

2.2.3 Kubernetes

Is also an open-source system for automating deployment, scaling, and management of containerized applications. It is a project created to manage a cluster of Linux containers as a single system, managing and running Docker container across multiple hosts, offering co-location of containers, service discovery, and replication control [16]. It was started by Google and now is supported by RedHat, Microsoft, IBM, and Docker.

Kubernetes[25] serves two purposes: It scales and starts containers across multiple Docker hosts, balancing the containers across them. It also adds a higher level API to define how containers are logically grouped, allowing defining pools of containers and loading balancing.

2.3 Migration in Cloud Computing

The process of moving our workload between cloud providers is referred as migration in cloud computing. There are three types of migration, virtual machine migration, process migration and container-based migration. There are techniques and tools to achieve each type of migration but also there are limitations for each one.

2.3.1 Virtual Machine Migration

The process of moving a running instance from one cloud providers to another called virtual machine migration. In virtual machine migration, transferred data contains everything an application would find in host, including the OS, middleware and virtual versions of the devices. Because of virtual machine applications are depended

[25] https://kubernetes.io/

on cloud provider specifications, virtual machine migration is difficult. Cloud provider specifications may differ in hypervisor type, image format, container format, tools and APIs. A virtual machine is tied up with cloud provider specifications and we can't migrate to a cloud provider with an alternate infrastructure. To achieve virtual machine migration between heterogeneous cloud environments we have to use a middleware to bridge the specification gaps between cloud providers.

2.3.2 Process Migration using CRIU

The process of moving a running application to a different host referred as process migration. CRIU[26] is a software tool for Linux operating system to freeze a running application (or part of it) and checkpoint it as a collection of files on disk [17]. Specifically, this tool lets you store a state of a process and restore it as it was during the time of freeze on the same or another host with the initial PID. CRIU separates the application from the underlying OS and freeze all the processes which are associated with this application. All information related to the process is stored in one or more image files. These image files contain information, such as memory maps, pipes, file descriptors, inter-process communication, etc.

The initial purpose of this tool was to avoid application data loss caused by system failures. In this thesis, we examined how CRIU tool could achieve a successful process migration from one host to another.

Process migration is a kernel level migration and requires Linux kernel v3.11 or newer, with some specific options set. Depending on application functionality, some kernel configurations are required. Assuming that our system supports all dependencies of process migration we have to use CRIU command tool with various parameters. Checkpoint process is not a straightforward process, differs depending on the application we want to migrate and requires knowledge of operating system to use the appropriate parameters.

Beyond kernel configuration and the appropriate checkpoint parameters, CRIU has

[26] https://criu.org/Main_Page

some limitations. The basic limitation is that for a successful migration both the source and destination system must have the same versions of libraries. Also restore process fails if the PID is in use on destination system. Furthermore, checkpoint process freezes the process and its process tree, we can't checkpoint and restore a process on its own [18].

2.3.3 Container-based Migration

Container-based migration is also a software level migration. In contrast virtual machine applications, the host OS and some middleware are shared and transferred data in container-based migration contains only the application and some system libraries. Applications designed for containers are forced to be compatible in most systems that deploy applications in containers. Docker uses CRIU tool to checkpoint and restore a container [19].

Blue outline in figure (5) below shows the data that transferred during virtual machine and container-based migration process.

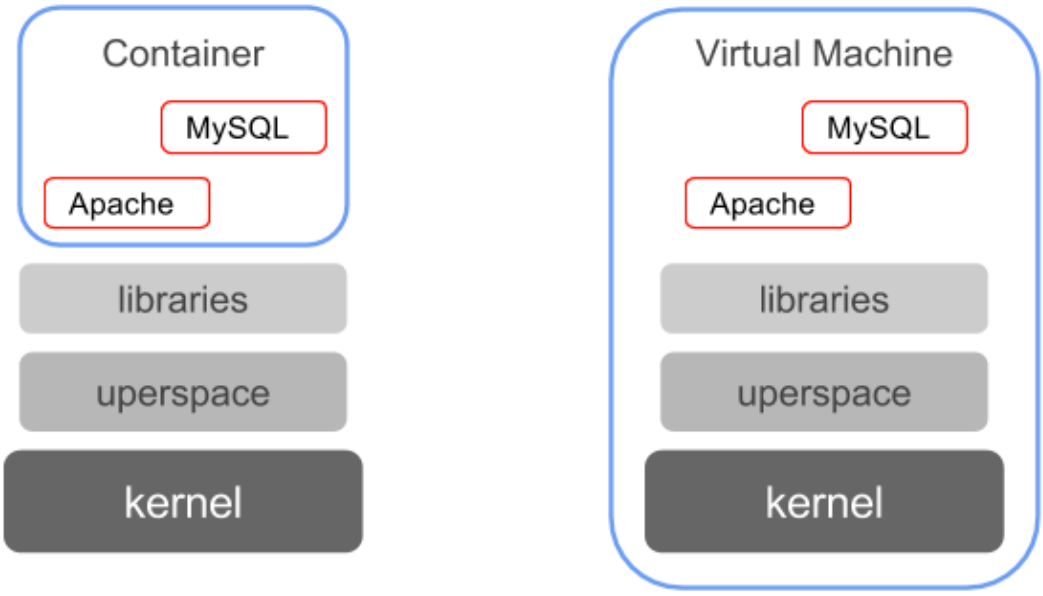

Figure 5 – Transferred data during virtual machine and container-based migration respectively

2.4 Other Technologies

2.4.1 Representation State Transfer (REST)

REST is an architectural style based on a set of principles that describe how data objects or resources can be defined and addressed on the internet. Clients and servers are separated from REST operations and communicate by transferring representations of resources through an interface, which improve client code portability. REST's decoupled architecture and light weight communications between server and client, make REST a popular building style for cloud-based APIs. REST runs over Hypertext Transfer Protocol (HTTP) and has constraints such as stateless existence, cache and layered system leverage.

2.4.2 Client URL Library (cURL)

cURL is a library for transferring data using various protocols. Users use this library in order to simplify the development of RESTful web services in PHP. We used this library to make HTTP requests to achieve client-server communication.

2.4.3 Extensible Markup Language (XML)

In computing, XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The design goals of XML empathize simplicity, generality and usability across the internet. It is textual data format and it is widely used for the presentation of arbitrary data structures such as those used in web services.

2.4.4 JavaScript Object Notation (JSON)

An alternative to XML, Json is also 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 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.

Chapter 3

Virtual Machine Migration

3.1 Migration between Cloud Providers

Cloud to cloud (C2C) migration is the process of moving physical or virtual machines along with their associated configuration, operating systems, applications and storage from a cloud environment to another. In any case, successful migration to another environment may require the use of middleware, such as a cloud integration tool.

3.1.1 Homogeneous Migration

Virtual machine migration from a node / zone to another in any cloud environment or virtual machine migration between two cloud environments with the same architecture is a straightforward process. There are no gaps because of the use of the same protocols and standards. We have to move a running instance between clouds or nodes in a cloud while maintaining its hardware, software and network configurations [20].

3.1.2 Heterogeneous Migration

Heterogeneous migration process is more complicated. In this case, we have to move an instance from a cloud environment to another with different architecture, protocols and standards. So have to use a middleware to bridge the gaps between these environments. The migration restrictions between heterogeneous cloud platforms may are on hypervisor, image, virtual machine description (meta-data) they use.

3.2 Migration Service

This section presents our approach to virtual machine migration between homogeneous but also heterogeneous cloud environments. Our service utilizes the Openstack's and VMware's REST API to allow users to perform migration.

3.2.1 Service Model

The service is designed as a modular cloud PaaS in order to allow easy deployment using API interfaces. The user uses a web interface to provide required information which will be used by the service for a successful migration. The migration service is responsible for transferring a running instance from one cloud environment to another. In heterogeneous case, the service is also responsible for configuring the instance to be portable on target cloud environment.

The model composed by the user that interacts with the service through the front-end interface (using GUI), the service that performs VM migration include all needed actions and the back-end system that includes the source and target cloud as demonstrated in Figure 6.

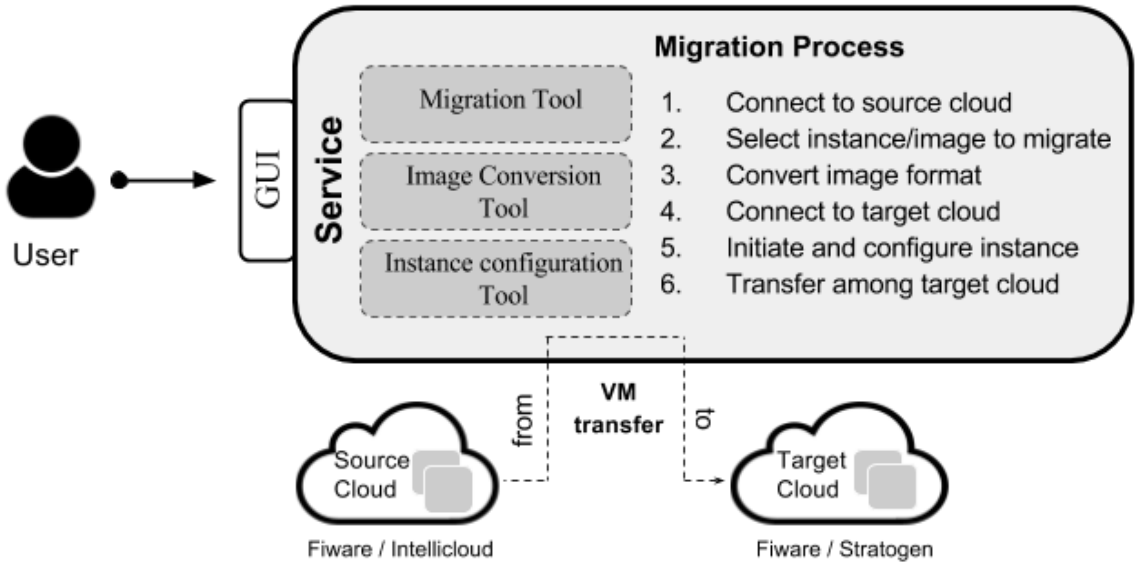

Figure 6 – Service Model

The following list details the specification of each component.

- The UI allows user to interact with the service to provide information and achieve a successful migration. The user, through the GUI, uses the migration tool in order to transfer an instance among the target cloud.

- The migration tool composed by two modules, one for the download procedure from source cloud and one for upload procedure for target cloud. The migration tool guides the user to perform an instance migration between two homogenous or heterogeneous clouds in a few steps. On each step, the user provides required information and the migration tool performs the corresponding action on the cloud.

- The image conversion tool is responsible to convert the downloaded image to a portable format for the upload procedure on target cloud.

- The instance configuration tool performs an action on target cloud to initiate the virtual machine. Specifically, the tool uploads a XML file (OVF descriptor) on target cloud with initial instance details.

- The back-end of the service offers all the functionality of the service. The back-end stores temporarily the image until to be portable by the image conversion tool and uploaded on target cloud.

3.3 Service Functionality

The service guides the user to migrate virtual machines or services with a few easy steps. Currently, it offers this functionality for Fi-Lab, Intellicloud[27] and Stratogen[28] environments. Fi-Lab and Intellicloud built their environment based on Openstack infrastructure and Stratogen built its environment based on VMware infrastructure. At the same time, the service performs automated actions to bridge the gaps between these infrastructures.

[27] http://cloud.intellicloud.tuc.gr/

[28] http://www.stratogen.net/

With this service anyone can migrate an existing virtual machine from Intellicloud to Fi-Lab and vice versa or migrate an existing virtual machine from one of these Openstack environments to Stratogen. It does not require any special knowledge other than the basic process to launch a new instance from Openstack dashboard or VMware dashboard. User Interface guides the user with the steps below to achieve the migration, we present one use case scenario for homogeneous cloud environments (Intellicloud and Fi-Lab) and one for heterogeneous cloud environments (Fi-Lab and Stratogen).

3.3.1 Homogeneous Migration

1. Authentication: User provides his username, password and tenant name to be authenticated on Intellicloud. Authenticating generates a unique token for every user for which is being in every action user performs on the source cloud.

2. Get Instances: The service retrieves all existing instances registered to this user.

3. Get Images: The service retrieves a list of public images and images registered to this user including snapshots.

4. Create Snapshot: User selects an existing running instance for migration. Snapshot makes a copy of this instance, also contains all the hardware configuration of the instance.

5. Download Image: User selects an existing public image or a snapshot for downloading.

6. Authentication: User provides his username, password and tenant name as the source cloud authentication to be authenticated on Fi-Lab cloud.

7. Select Fi-Lab Region: User selects the region which wants to transfer his virtual machine.

8. Upload: The user provides provides the name of the new image and the service performs actions automatically to upload the image or snapshot which user

downloaded earlier.

- It creates a new blank image with the name provided by the user.
- It uploads the data of downloaded image to the blank image which created above and the initial instance is read to deploy on Fi-Lab cloud.

3.3.2 Heterogeneous Migration

1. Authentication: User provides his username, password and tenant name to be authenticated on Fi-Lab cloud. Authenticating generates a unique token for every user for which is being in every action user performs on the source cloud.

2. Select Fi-Lab Region: User selects the region which has his workload.

3. Get Instances: The service retrieves all existing instances registered to this user.

4. Get Images: The service retrieves a list of public images and images registered to this user including snapshots.

5. Create Snapshot: User selects an existing running instance for migration. Snapshot makes a copy of this instance, also contains all the hardware configuration of the instance.

6. Download Image: User selects an existing public image or a snapshot for downloading.

7. Authentication: User provides his username, password to be authenticated on Stratogen cloud. Authenticating generates a unique token for every user for which is being in every action user performs on the source cloud.

8. Create vApp: User creates a new vApp which is necessary to deploy a new virtual machine.

9. Upload OVF descriptor: VMware vApps operate on the Open Virtualization Format (OVF) and also are exported in OVF format. User uploads the OVF descriptor to initiate this vApp template which contains virtual machine's meta-data such as

hardware configuration and virtual machine image size.

10. Upload Image: If the previous step is done successfully user uploads the reference file (VMDK image) and user's virtual machine is ready to deploy on Stratogen source cloud.

- The service performs automatically the conversion of downloaded image to VMDK format.

3.5 User Interface

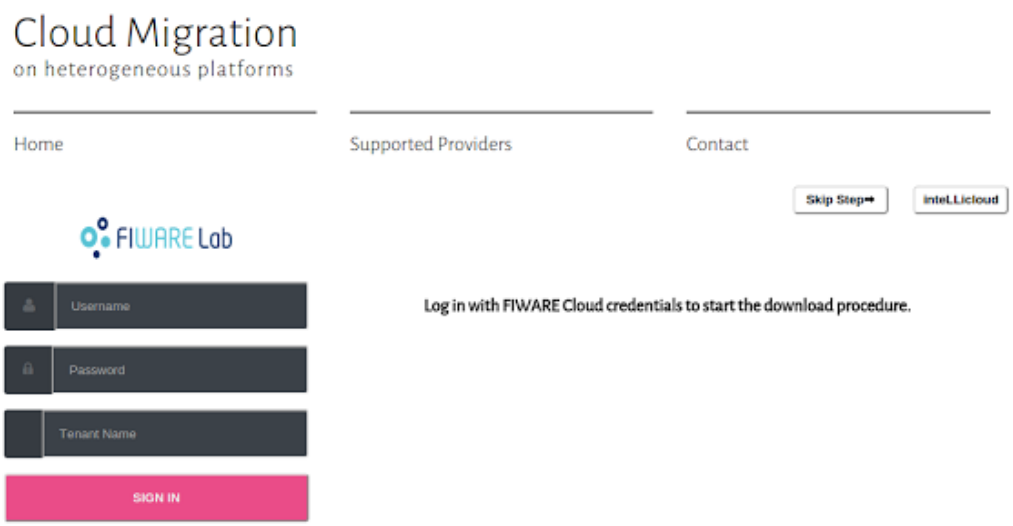

Figure 7 – Fi-Lab Authentication

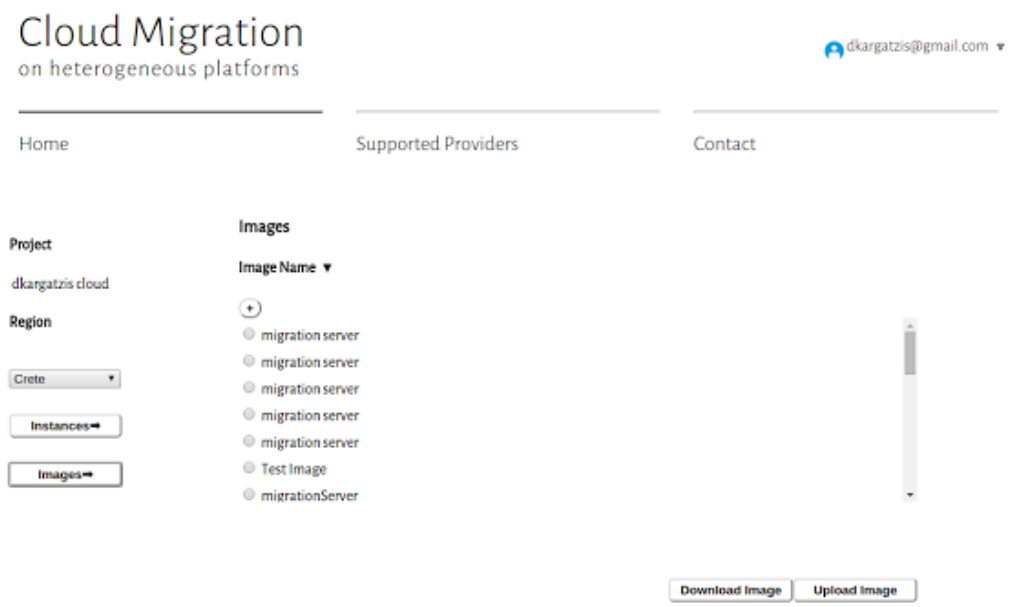

Figure 8 – List of images

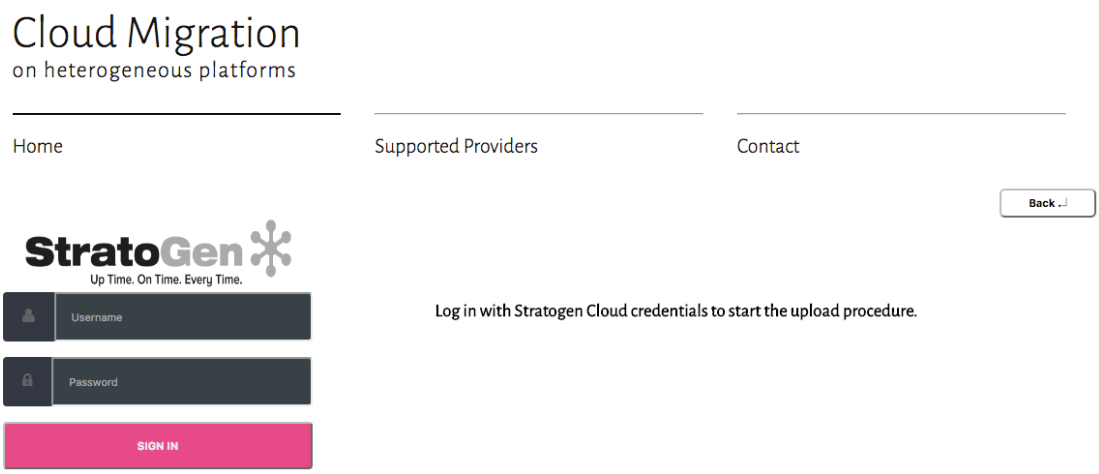

Figure 9 – Stratogen Authentication

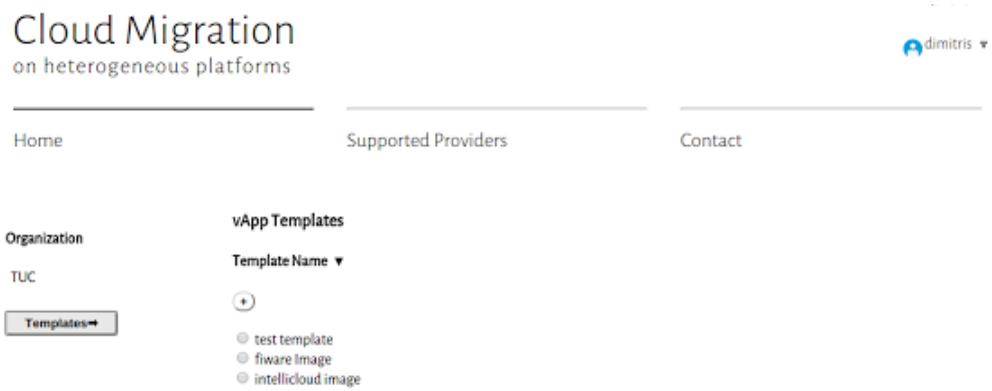

Figure 10 – List of vApp templates

Figure 11 – Upload OVF descriptor

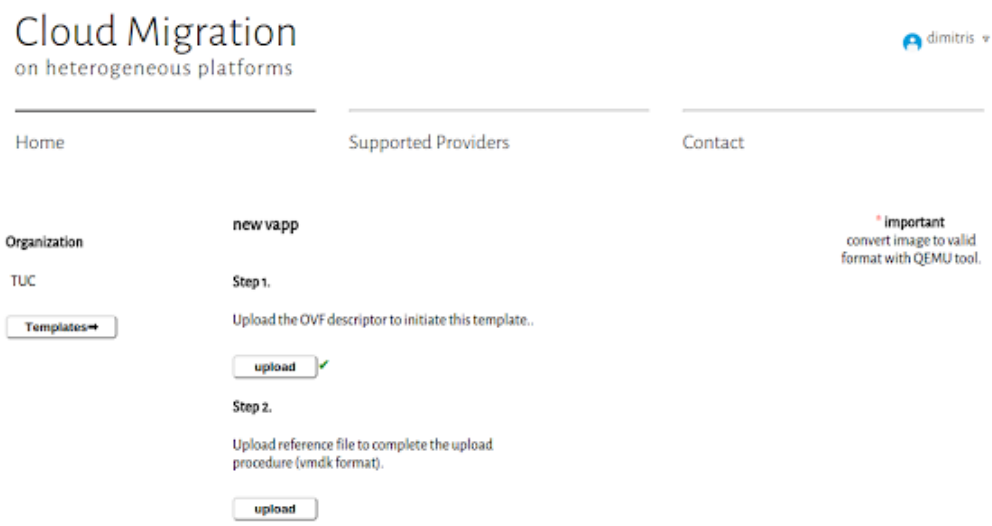

Figure 12 – Upload File reference

Chapter 4

Implementation

The abstract flow of process of the service is that it interacts with source and target environments through XML or JSON API. It uses information in order to perform actions on two clouds by communicating with the cloud's REST API.

4.1 Implementing Homogeneous Migration

The communication with the clouds is done by performing calls on Openstack APIs.

STEP 1. Intellicloud Authentication

REQUEST TYPE	**POST**
URL	http://cloud.lab.fi-ware.org:4730/v2.0/tokens
HEADERS	{“Content-Type”: “application/json”}

BODY	{
	"auth": {
	"tenantName": "user cloud",
	"passwordCredentials": {
	"username": "user@mail.com",
	"password": "********" }
	}
	}

Description: The service prepares and performs the call for authentication by the Intellicloud. The Openstack identity service generates and returns a token that represents the authenticated identity of a user and grants authorization on a specific project or domain.

STEP 2. Retrieve the list of instances

REQUEST TYPE	**GET**
URL	http://147.27.50.1:8774/v2/$tenant-id/servers
HEADERS	{“Content-Type”: “application/json”, “X-Auth-Token”: “$token”}

Description: The service retrieves and displays the list of instances owned by the user.

STEP 3. Create Snapshot

REQUEST TYPE	POST
URL	http://147.27.50.1:8774/v2/$tenant_id/servers/$server_id /action
HEADERS	{“Content-Type”: “application/json”, “X-Auth-Token”: “$token”}

BODY	{
	"createImage":
	{"name": "Snapshot", "metadata": {}}
	}

Description: The user selects one instance (running or idle) which he wants to migrate and creates a snapshot of.

STEP 4. Retrieve the list of images

REQUEST TYPE	GET
URL	http://147.27.50.1:8774/v2/$tenant_id/images
HEADERS	{“Content-Type”: “application/json”, “X-Auth-Token”: “$token”}

Description: The service retrieves and displays the list of images. This list contains public images that published either from other users or from cloud environment and user’s private images.

STEP 5. Download Snapshot

REQUEST TYPE	GET
URL	http://147.27.50.1:9292/v2/images/$image_id/file

HEADERS	{“Content-Type”: “application/json”, “X-Auth-Token”: “$token”}

Description: The user selects an image which he wants to download, in our case the snapshot that created earlier.

STEP 6. Fi-Lab Authentication

REQUEST TYPE	**POST**
URL	http://cloud.lab.fi-ware.org:4730/v2.0/tokens
HEADERS	{“Content-Type”: “application/json”}
BODY	{ "auth": { "tenantName": "user cloud", "passwordCredentials": { "username": "user@mail.com", "password": "********" } } }

Description: The service prepares and performs the call for authentication by the cloud. The Openstack identity service generates and returns a token that represents the authenticated identity of a user and grants authorization on a specific project or domain.

STEP 7. Create new Image

REQUEST TYPE	**POST**
URL	http:// 147.27.60.1:9292/v2/images

HEADERS	{“Content-Type”: “application/json”, “X-Auth-Token”: “$token”}

BODY	{
	"name": "$name ",
	"container_format": "bare",
	"disk_format": "qcow2",
	"visibility": "public"
	}

Description: The cloud creates a new image with the properties we include in the request body and returns the image id.

STEP 8. Upload the image reference file

REQUEST TYPE	**PUT**
URL	http://147.27.60.1:9292/v2/images/$image_id/file
HEADERS	{“Accept:application/octet-stream”, “X-Auth-Token”: “$token”}

Description: The service uses the image id that returned in the previous step to upload the reference file for the snapshot.

4.2 Implementing Heterogeneous Migration

In this case the procedure is different. The communication with the clouds is done by performing calls on Openstack APIs, furthermore, the service has to perform actions such as image conversion and instance details configuration.

STEP 1. Fi-Lab Authentication

REQUEST TYPE	**POST**

URL	http://cloud.lab.fi-ware.org:4730/v2.0/tokens
HEADERS	{“Content-Type”: “application/json”}

BODY	{
	"auth": {
	"tenantName": "user cloud",
	"passwordCredentials": {
	"username": "user@mail.com",
	"password": "********" }
	}
	}

Description: The service prepares and performs the call for authentication by Fi-Lab cloud. The Openstack identity service generates and returns a token that represents the authenticated identity of a user and grants authorization on a specific project or domain.

STEP 2. Retrieve the list of instances

REQUEST TYPE	**GET**
URL	http://147.27.60.1:8774/v2/$tenant-id/servers
HEADERS	{“Content-Type”: “application/json”, “X-Auth-Token”: “$token”}

Description: The service retrieves and displays the list of instances owned by the user on a specific region. Specifically, the first part of URL (147.27.60.1) specifies the region which the service performs requests.

STEP 3. Create Snapshot

REQUEST TYPE	POST
URL	http://147.27.60.1:8774/v2/$tenant_id/servers/$server_id /action
HEADERS	{“Content-Type”: “application/json”, “X-Auth-Token”: “$token”}

BODY	{
	"createImage":
	{"name": "Snapshot", "metadata": {}}
	}

Description: The user selects one instance (running or idle) which he wants to migrate and creates a snapshot of.

STEP 4. Retrieve the list of images

REQUEST TYPE	GET
URL	http://147.27.60.1:8774/v2/$tenant_id/images
HEADERS	{“Content-Type”: “application/json”, “X-Auth-Token”: “$token”}

Description: The service retrieves and displays the list of images. This list contains public images that published either from other users or from cloud environment and user’s private images.

STEP 5. Download Snapshot

REQUEST TYPE	GET
URL	http://147.27.60.1:9292/v2/images/$image_id/file

HEADERS	{“Content-Type”: “application/json”, “X-Auth-Token”: “$token”}

Description: The user selects an image which he wants to download, in our case the snapshot that created earlier.

STEP 6. Stratogen Authentication

REQUEST TYPE	**POST**
URL	http://mycloud.statogen.net/api/sessions
HEADERS	{“Accept”: “application/*+xml;version=5.1”}
BASIC AUTH HEADERS	{username@organization_name:password}

Description: The first thing user needs to do is login and get the authentication token. The service needs to use this authentication token as a header in all subsequent API calls.

STEP 7. Create a new vApp

REQUEST TYPE	**GET**
URL	http://mycloud.statogen.net/api/org/$organization_id
HEADERS	{“Accept”:“application/*+xml;version=5.1”,“x-vcloud-authorization”: “$token” }

Description: The service explodes the organization id from the authentication response body and makes a get request to the cloud with this organization id as URL parameter. The response body provides links to various attributes and actions to do with the organization.

REQUEST TYPE	**POST**
URL	http://mycloud.statogen.net/api/$new_vdc_link
HEADERS	{“Accept”:“application/*+xml;version=5.1”,”Content-Type”: “application/vnd.vmware.vcloud.uploadVAppTemplateParams+xml”,“x-vcloud-authorization”: “$token” }
BODY	<?xml version="1.0" encoding="UTF-8"?> <UploadVAppTemplateParams name="$vapp_name" xmlns="http://www.vmware.com/vcloud/v1.5" xmlns:ovf="http://schemas.dmtf.org/ovf/envelope/1"> <Description>Ubuntu vApp Template</Description> </UploadVAppTemplateParams>

Description: The service explodes from a previous response body the link for the new vApp and makes a POST request to initiate this new vApp with some properties.

STEP 8. Upload OVF Descriptor

REQUEST TYPE	**PUT**
URL	http://mycloud.statogen.net/api/$ovf_link

HEADERS	{“Accept”:“application/*+xml;version=5.1”,“x-vcloud-authorization”: “$token” }

Description: The response body of vApp creation request contains the link for the OVF descriptor. The OVF descriptor is a XML file which contains initial instance properties such as virtual disk name and size, virtual disk information (capacity), virtual machine information (name, OS type) and virtual hardware requirements (resources, controllers, network).

STEP 9. Convert Image

```
$ qemu-img convert -f qcow2 -O vmdk image.qcow2 image.vmdk
```

Description: The service executes the above command to convert the downloaded image to a portable format for the target cloud.

STEP 10. Upload Reference File

REQUEST TYPE	**PUT**
URL	http://mycloud.statogen.net/api/$file_link
HEADERS	{“Accept”:“application/*+xml;version=5.1”,“x-vcloud-authorization”: “$token” }

Description: The response body of vApp creation request also contains the link for the reference file. If the upload of OVF descriptor succeeded, user needs to upload the reference file and the snapshot is ready to deploy on Stratogen cloud.

4.2 Performance Analysis

The migration service implemented its functionality in homogeneous environments but also in heterogeneous environments as shown in section above. Specifically, homogeneous migration implemented its functionality in Openstack environments with Intellicloud of the Technical University of Crete[29] as source cloud and Fi-Lab as target cloud and heterogeneous migration implemented its functionality in Fi-Lab as source cloud and Stratogen as target cloud. Stratogen infrastructure is based on VMware platform. The performance evaluation of the service involves two experimental use cases that demonstrate the time required for each instance migration procedures, homogeneous and heterogeneous.

For homogeneous procedure, the assumption is that the user performs a migration of an instance from Intellicloud to Fi-Lab system. Figure 13 demonstrates the variation among the required calls. We used an image of Centos 7 (958.4 MB) and an image of Ubuntu 12.04LTS-64 (243.6 MB) to deploy two small flavors virtual machines.

[29] http://www.tuc.gr/index.php?id=5397

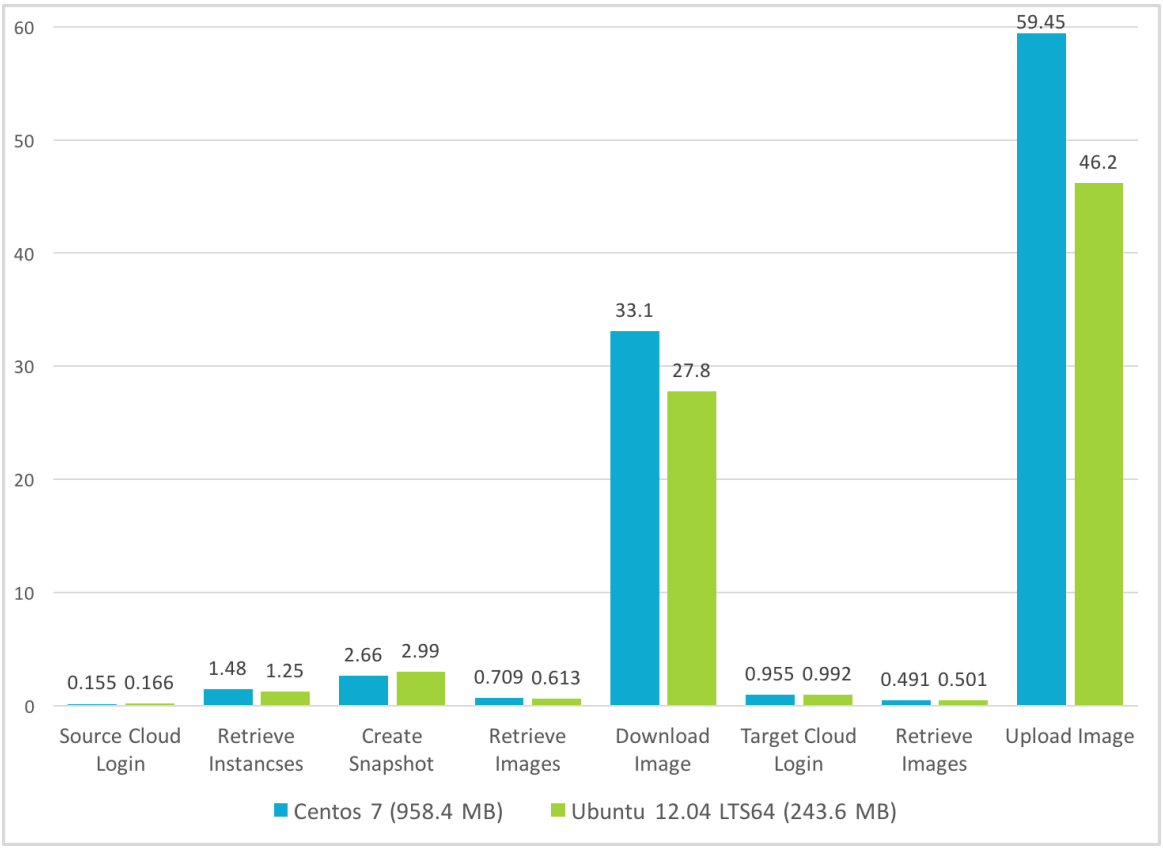

Figure 13 – Homogeneous migration performance

For heterogeneous procedure, the assumption is that the user performs a migration of an instance from Fi-Lab to Stratogen system. Figure 14 demonstrates the variation among the required calls. We used an image of Centos 7 (896.6 MB) and an image of Ubuntu 12.04 (458.8 MB) to deploy two small flavors virtual machines.

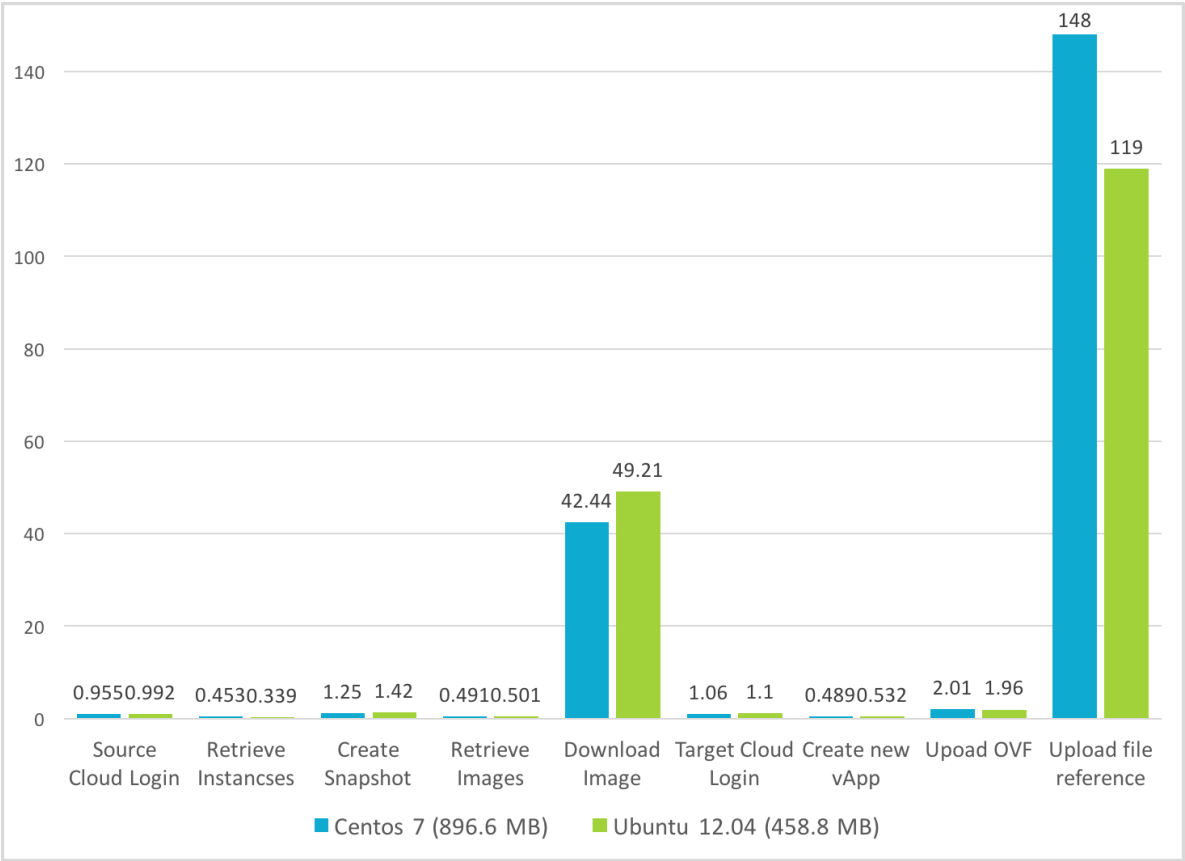

Figure 14 – Heterogeneous migration performance

We observed that the size of the snapshot is bigger than the size of the initial image. Specifically, in Fi-Lab the initial size of Centos 7 image is 896.6 MB and the size of the snapshot is 1.9 GB and the initial size of Ubuntu 12.04 image is 458.8 MB and the size of the snapshot is 1.3 GB.

Finally, it should be mentioned that the service executes most of the APIs calls related with configurations but the time required for downloading and uploading increases significantly the total time of migration, actions that depend to the image size and on the bandwidth speed.

Chapter 5

Conclusions & Future Work

5.1 Conclusions

Looking back, the problem we tried to address was the “vendor lockin”. In this thesis we present an implementation of virtual machine migration with two case scenarios and the problem still exists because there are many heterogeneous cloud environments. Also if one of these cloud providers decides to update its tools and API or change standards, specific protocols, our implementation may not achieve a successful migration. So, it’s not effective to add more use case scenarios in this implementation because the procedure for each use case scenario may differs and image conversion may be impossible. Process migration abstracts applications from the underlying OS and may bridge the heterogeneity gaps. Because of these limitations we describe in section 2.3.2, we can’t achieve a process migration successfully. As a conclusion, the later technology proved to be feasible and more promising although it is still not fully supported by infrastructure (operating system) tools that allow migration independent of the state of the underlying operating system kernel at the time of transfer.

5.2 Future Work

5.2.1 Container-based migration

We want compare hardware virtualization and container-based (operating-system-level) virtualization and examine how container-based virtualization could avoid the heterogeneity in cloud computing. Applications designed for containers are forced to be compatible in most systems that deploy applications in containers. We built containers such as mongodb[30] and Cassandra[31] on Google compute engine[32] and we achieved a successful transfer of our workload to a different zone through Google

[30] https://www.mongodb.com/
[31] http://cassandra.apache.org/
[32] https://cloud.google.com/compute/

compute engine console. In a future work, we will propose a mechanism that implements migration using containers in a few steps and we will run a series of experiments to show proof of concept. As a conclusion, the later technology proved to be feasible and more promising.

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] 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

[3] The Open Group, "Maximizing the Value of Cloud for Small-Medium Enterprises", http://www.opengroup.org/cloud/cloud_sme/

[4] Linkedin , "Cloud Computing Architecture", https://www.linkedin.com/pulse/20140621112709-142734032-cloud-computing-part-3-architecture

[5] Wikipedia, "Hypervisor", https://en.wikipedia.org/wiki/Hypervisor

[6] Virtualization Admin, "What is the Snapshot", http://www.virtualizationadmin.com/faq/snapshot.html

[7] Qemu Project "Disk Image File Formats", http://download.qemu-project.org/qemu-doc.html - disk_005fimages_005fformats

[8] VMware "OVF Tool User's Guide", https://www.vmware.com/support/developer/ovf/ovf420/ovftool-420-userguide.pdf

[9] Wikipedia "VHD File Format", https://en.wikipedia.org/wiki/VHD_(file_format)

[10] KVM "Qcow2", http://www.linux-kvm.org/page/Qcow2

[11] Openstack "Disk and Containers Formats",

https://docs.openstack.org/developer/glance/formats.html

[12] Wikipedia "Open Virtualization Format", https://en.wikipedia.org/wiki/Open_Virtualization_Format

[13] Openstack "Converting between Image Formats", https://docs.openstack.org/image-guide/convert-images.html

[14] Openstack "Converting between Image Formats", https://docs.openstack.org/image-guide/convert-images.html

[15] Docker "What is a Container", https://www.docker.com/what-container

[16] Kubernetes "Building High-Availability Clusters", https://kubernetes.io/docs/admin/high-availability/

[17] CRIU "Checkpoint / Restore", https://criu.org/Checkpoint/Restore

[18] Redhat "Checkpoint / Restore in User Space", https://access.redhat.com/articles/2455211

[19] CRIU "Docker", https://criu.org/Docker

[20] Vakanas, L., Sotiriadis, S. and Petrakis, E. (2015) "Implementing the Cloud Software to Data approach for OpenStack environments", http://www.intelligence.tuc.gr/~petrakis/publications/ARMS-CC2015.pdf