8-2016

Cloud Abstraction Libraries: Implementation and Comparison

Udit Agarwal
Clemson University

Cloud Abstraction Libraries: Implementation and Comparison

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Science

by
Udit Agarwal
August 2016

Accepted by:
Dr. Amy Apon, Committee Chair
Dr. Jim Martin
Dr. Hongxin Hu

ABSTRACT

Vendor lock-in makes it difficult for an organization to port their services, application or data. Cloud providers are in race to provide the best-in-class storage, networking and compute resources. Many organizations are moving towards micro-services and cloud services architecture. It is very important for an infrastructure platform to offer a high-quality cloud computing environment consistently across multiple cloud platforms. To enable this, a collaborative yet an independent cloud abstraction service is required. The cloud abstraction library should support the basic use cases of delivery pipeline, service management, cloud operations and security service.

Cloud interoperability standards helps to improve availability and scalability by providing cross organizational or vendor independent projects. An important aspect of cloud interoperability is development of standardized APIs to send and receive data, irrespective of the underlying cloud implementation. Cloud interoperability helps application and data portability between public clouds and private clouds. This thesis explores the role of open source libraries to use cloud specific features. Our work is to qualitatively and quantitatively evaluate Dasein cloud and jClouds against Amazon EC2 and Google Compute Engine. We believe that cloud standardization can be accelerated by implementations based on open source and open standards.

ACKNOWLEDGMENTS

It feels wonderful writing this page and I feel privileged to have access to such a great academic community at Clemson University. I would like to take this opportunity to express my gratitude to the people without whose support this thesis would not have been possible. I am deeply grateful to my thesis advisor, Dr. Amy Apon for her valuable guidance and support, and I would like to thank my committee members, Dr. Jim Martin and Dr. Hongxin Hu, for their valuable suggestions. Finally, I would like to thank my brother and family for their continuous support, encouragement and love during the long process in achieving this important goal.

.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTIOIN

1.1 Background and Motivation

The National Institute of Standards and Technology (NIST) defines Cloud Computing as; "A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction" [1]. The on-demand characteristic of cloud computing is what enables users to have elasticity, get access to resources over the network, and monitor the services. Some of the other important characteristics are storage, performance, networking, security, and maintenance. Two main models used to classify cloud computing are the Deployment Model and the Service Model [2].

The different ways in which cloud services can be deployed include public clouds, private clouds and hybrid clouds. The deployment model depends on the structure of an application, security of data and privacy of end users. The deployment model can also be considered as an "Access Model". The public cloud model is the most common form of deployment model and has the highest number of users with least constraints on accessibility, scalability and availability. The resources are offered to the client or users through a publically available platform using a pay-per-use basis. Amazon Web Services (AWS) and Google Compute Engine (GCE) are examples of public cloud offerings. The private cloud model is usually established exclusively for a single enterprise or an entity. The infrastructure is solely owned and maintained by an organization or third party vendor. Private cloud platforms offer a higher level of data security as compared to public or hybrid clouds. There are applications that require specialized protocol to interact with virtual machines and the infrastructure provided by

other deployment models is insufficient to cater to the needs of these specialized applications. The cost of utilizing the resources in a private cloud is higher. It is a responsibility of the utilizing organization to monitor and upgrade the infrastructure. There is a higher level of data security and more guarantee in the uptime of resources as they are not being shared. Some examples of the applications that help to maintain private clouds are OpenStack, CloudStack and VMware's vCloud. The hybrid cloud model provides a fallback platform for organizations when there is a spike in the traffic and the dedicated resources are not enough to handle the load. Additional traffic can be routed to a public cloud, which can interact with the private cloud through an API. The hybrid cloud model could also serve as a platform to run applications during disaster recovery. CliQr is an example of a hybrid cloud.

Cloud service models include Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Each service level provides a different set of manageability and customization of the underlying infrastructure. IaaS provides the highest level of customization and management of compute, storage and network resources. Cloud service providers offers customizable resources by installing any desired operating system along with supporting libraries and applications for use. Compute machines can be attached to storage and networking resources to create an end-to-end application. The user is responsible for maintaining and upgrading the software on the VMs. Amazon EC2, Google Compute Engine, Microsoft Azure are examples of platforms that offer IaaS. PaaS provides a greater level of management and control to the end users. It provides a set of pre-defined tools and software to develop and deploy the application. The cloud provider is responsible for maintaing and upgrading software on the VMs. User application can be scaled automatically depending on the traffic. It also ensures a certain level of data and application security. Google App Engine, Heroku, and

Amazon Elastic Beanstalk are examples of PaaS. SaaS provides fully managed on-demand delivery of applications that can be accessed over the internet. The applications can be utilized without IT support and maintenance. However, SaaS has offers a minimal amount of customization to the end users. Google Mail Engine and Salesforce.com are examples of SaaS.

Standards and semantic interoperability are a way of giving organizations a choice in terms of cloud vendor. A solution based on open standards is portable between multiple cloud environments. Currently, there is a high complexity and high cost associated with switching between different cloud environments.

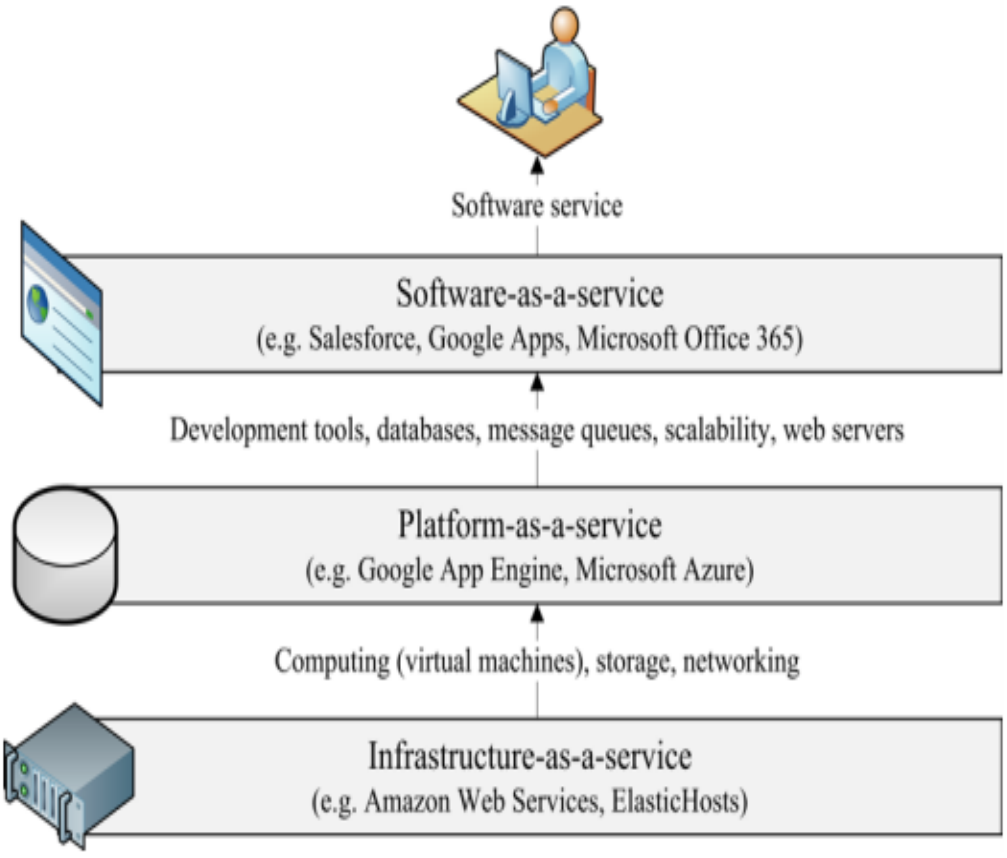

Figure 1.1: Layers of Cloud Infrastructure.

Cloud abstraction libraries help to reduce the complexity involved in leveraging multi-cloud environments. Two problems in a multi-cloud environment are portability and integration

with different management APIs. This is a place where cloud abstraction libraries like Dasein Cloud [21] and Apache jClouds [20] come into play. These libraries provide an abstraction layer to an end programmer, giving the ability to use single API to communicate between different cloud providers. The libraries hide the complexity related to each cloud provider by offering capabilities to start a virtual machine, stop a virtual machine or attach disk space, etc. The following points help to justify the need for the cloud abstraction layer. A cloud abstraction layer:

- Allows developers to take advantage of the cloud (Infrastructure as a Service) through regular object oriented code,
- Provides on demand resources and automation,
- Integrates with existing projects and can provision virtual machines,
- Hides complexity of the cloud provider API,
- Permits each cloud to be accessed via a different configuration file,
- Permits open source extensions to support any other non-common API, and
- Supports automation tools like Chef and puppet (Not all libraries support this).

1.2 Problem Definition

The abstraction layer provides an ability to run applications among different cloud platforms without having to rewrite them partly or fully [9]. An ideal cloud abstraction layer can move data, application or service components regardless of the Operating System, storage or compute requirements between the supported cloud providers. Some of the automation tools like Chef or Puppet supported by cloud abstraction layer are be used to automate service deployment. The focus of the thesis is on two major goals: The first goal is a qualitative comparison of Dasein cloud and jClouds which includes user experience, ease of setup and community support. The

second goal is to quantitatively evaluate the performance metrics such as cost involved for common operations (e.g. start a Virtual Machine with different configurations such as varying instance type and different regions in cloud provider) and the number of concurrent requests the abstraction layer can handle.

Dasein library is one of the candidates for comparison, since we are familiar with the code structure and have been using it for past two years. After doing research on the related and existing solutions we chose jClouds, jClouds has been around for more than five years and is probably the first open source library to support multi cloud deployment. The project has great community support and has an implementation for all major cloud providers. We evaluate the abstraction layer against two major cloud providers, which are Amazon Web Services (AWS) and Google Cloud Engine (GCE). AWS EC2 and GCE have constantly been in the list of top ten IaaS provider [10].

1.3 The Approach

The following approach is taken to do a complete investigation in identifying features provided by cloud abstraction libraries.

1) Study of existing solutions and literature review of work done in the field of cloud abstraction libraries.
2) Important features listed in table 3.1 and 3.2 in section III, are selected for qualitative comparison of the libraries.
3) Experiments on specific functionality of a selected feature. For example, cost to start a vitual machine is selected for calculating the cost of common operations.
4) Developers experience to report ease of use and community support.

5) Hands on experience with cloud abstractions libraries by making them available on the test environment.

Steps (1) to (5) are used to identify, compare and contrast features provided by different cloud abstraction libraries.

1.4 Outline of the Thesis

The rest of this thesis is organized as follows. Background and related work for cloud abstraction libraries are discussed in Chapter 2. Chapter 3 discusses the performance evaluation of the abstraction layer and discussion of tradeoffs. Chapter 4 explains the implementation details and how to set up an environment. Chapter 5 summarizes this thesis with conclusion and further work.

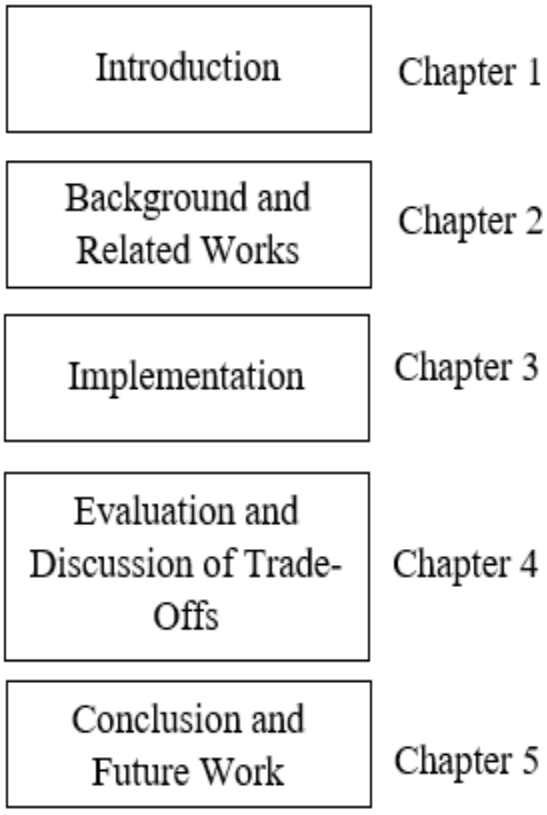

Figure 1.2: Outline of the thesis

CHAPTER TWO

BACKGROUND AND RELATED WORKS

In section 2.1 and 2.3, background of cloud portability concept and cloud abstraction libraries are discussed.

2.1 Cloud Portability

Cloud portability enables multiple cloud platform providing diverse services to dynamically provision hardware or software environments for the same application. From a cloud perspective, application, service or data should have the flexibility to move between multiple cloud providers as the decision might be dictated by economic, geographical, and legal reasons. Cloud portability allows infrastructure to be treated as pure commodity and prevents vendor lock-in. Currently, the services and features provided by the cloud providers are heterogeneous and incompatible between different service providers [3].

2.1.1 Economical reasons

Two economic reasons to promote cloud portability are: 1) Maximizing the Rate of Interest for investment in commodity hardware 2) Creation of a shared cloud platform. Reducing operational cost is one of the main reasons why in the first places users switch to cloud platform and cloud portability will help to achieve that goal [4]. Currently, porting applications between different cloud platforms require change in design of the application and services to make it work with a new environment. Some of the reasons why organizations and users switch cloud providers are; Enhancement in cloud services, services unique to a cloud provider, and cloud provider offering similar service at a lower price.

If cloud portability is ensured, third parties libraries like Dasein cloud and jClouds can act as an intermediary between users and cloud providers. This can be envisioned as a many-to-

many relationship between users and cloud providers, paving a way for multi cloud deployment as per the user requirements.

2.1.2 Technical reasons

Technically, Cloud portability is needed for at least two reasons: 1) To adapt to workload changes and 2) Disaster Recovery Management. During an increase in the workload, the application might be deployed to a hybrid cloud to handle peaks in service and address additional resources required by the private cloud. Portability between cloud helps to ensure users, that the application is up and running during any un-scheduled activity or disaster. Porting process is dependent on an agreement between the application owner and cloud provider.

2.1.3 Legal reasons

Legally, Cloud portability is needed from for at least two reasons: 1) The Service Level Agreements may vary according to the state and country of the cloud provider and end users and 2) Avoiding dependence on only one external provider. Changes in a country legislation or changes in the customer location can trigger the need to port the software assets from a cloud environment to another.

2.2 Types of Solutions

The following work has discovered [36] three types of solution suggested by Gonidis et al [5] and Silva et al [6] for the cloud portability problem: 1) supporting state of the art standards and protocols like Topology and Orchestration Specification for Cloud Applications (TOSCA), Cloud Data Management Interface (CDMI), Open Cloud computing Interface Working Group (OCCI), Open Virtualization Format (OVF) [34] and [35]; 2) cloud abstraction APIs (like jClouds or Dasein) and 3) publishing and adopting models based on semantics . These solutions are closely related to each other rather than being independent. Moreover, regardless of the

solution type, it makes more sense to develop a cloud abstraction layer that will resolve the dissimilarities between different cloud providers. The abstraction layer makes it possible for the developers to not be bound to a specific programming language and hide the complexity of cloud provider's API. There are also several libraries and tools that support interaction between an abstraction layer and the target service.

2.3 Cloud Abstraction API

With rise in cloud computing all the cloud providers are offering their own API, with limited set of functionality to extend it to support multi-cloud deployment. The end users are willing to commit to host and run their applications on a certain cloud provider for a fixed period of time [7]. This means if they decide to move to another cloud provider, they would have to spend huge amount of money to re-write the software to support the new cloud provider. Cloud abstraction API libraries like Dasein cloud [20], Apache Jclouds [19], delta cloud [22], Apache Libcloud [23], Cloudloop [24] and Simple Cloud offers a standardized API definition for IaaS clouds with access to the different kinds of services implemented by the abstraction layer. Dasein clouds and jClouds provide a REST API to communicate with the cloud providers supported in the project. The projects are extensible to support new cloud providers and services.

Delta cloud provides a REST API to access resources offered by the cloud provider. Fundamentally, it provides a number of extensions or libraries corresponding to each cloud provider. Each library implements some of the features provided by a specific cloud provider and may not cover all the features supported otherwise. On the contrary, there are also some present language dependent open-source projects. Libcloud, for example, is an open source library written in Python and supports many popular cloud providers. The library provides a shell to implement some of the compute, storage and networking resources. Jclouds and Dasein, are

examples of open source projects written in java. Both of these projects support a large number of cloud providers. Similarly, to Libcloud, the following libraries are extensible to support various IaaS's compute, platform, database, storage, etc. services.

Dasein library is one of the candidates for comparison, as we are familiar with the code structure and have been using it since past two years. After doing research on the related and existing solutions we chose jClouds. jClouds has been around for more than five years and is probably the first open source library to support multi cloud deployment. The project has great community support and has an implementation for all major cloud providers. Some of the other libraries like; Delta cloud run on a server typically one port per provider, Libcloud is a standard Python library and Cloudloop has very less documentation. Keeping in mind the environment used to setup the library, language in which the library is written and documentation, we decided to go with Dasein and jClouds. A more comprehensive qualitative comparison of the two libraries is discussed further in the thesis.

Most of the providers of proprietary technologies, like to offer their own integrated stack built on best of breed technology and ways on how cloud applications within a cloud should be accessed, operated, and managed [8]. Vendor lock-in is a major issue for business customers and the end user which wants to eliminate the need to be dependent on a cloud provider.

2.4 Cloud Architectures Compared

2.4.1 Amazon Elastic Compute Cloud (Amazon EC2)

Amazon EC2 is an acronym for Amazon Elastic Compute Cloud, and provides a platform to start virtual servers in a cloud environment. The EC2 dashboard allows any user with an active Amazon Web Services (AWS) account to start, access, stop and delete a virtual machine. Amazon EC2 provides a flexible way to start a virtual machine, either from an existing

Amazon Machine Image (AMI) or by specifying compute, storage and network requirements. Amazon Machine Image is a snapshot of previously launched EC2 instance and helps to start a new virtual machine based on pre-existing operating system and any installed applications.

Amazon EC2 supports a variety of Linux operating systems such as Red Hat, Debian, Ubuntu and Fedora. Recently, the EC2 service has also started supporting Windows server operating system. The power of the VM depends on factors like memory capacity, storage, CPU cores, and GPU. The flexibility to configure such components allow user to run applications with varying needs on a single platform. Amazon EC2 defines the minimum processing unit, referred to as EC2 Compute Unit (ECU), which is equivalent to CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. Amazon also limits the maximum number of instances a user can run in a region to 20.

The pricing and availability model for Amazon EC2 are divided into three types.

- On Demand: pay fixed rate depending on the instance type per hour.
- Spot: pay a fixed amount initially and get discounted rate for subsequent use
- Reserved: pay a fixed amount and reserve the instance type

Apart from EC2 instances, AWS also offers a variety of storage and network services that can be plugged into the EC2 virtual machine. Amazon's Simple Storage Service (S3) and networking service like Virtual Private Cloud (VPC), helps to develop end-to-end application with all the pieces in cloud and accessible to the end user. Amazon cloud watch provides a real-time metric collection and tracking service that can be used to monitor application performance. Within cloud watch, alarms may be defined.

2.4.2 Google Compute Engine (GCE)

Google Compute Engine (GCE), is one of the most widely used IaaS along with Amazon Web Services. Similar to EC2 platform, GCE provides a dashboard with identical features to manage virtual machines in cloud. It provides a restful API for managing compute, storage, and networking resources with an enhanced security protocol requiring an access token provided by OAuth 2.0. It provides a functionality to create VMs from custom image or a snapshot, similar to the concept of AMIs in EC2. The minimum processing unit defined by GCE has a CPU capacity of a 1.0-1.2 GHz 2007 Opteron processor. GCE uses 2.75 GCEU's to represent the minimum processing power of one logical core. The default configuration for each instance type are n1-standard-2 and n1-standard-2-d. The following instance type only differ in available scratch disk capacity while CPU capacity, memory capacity and persistent disk capacity remains the same. The maximum number of instances per user varies according to the project setup and are not limited to a region. GCE supports Linux and Windows operating system for virtual machine management. Similar to EC2, the compute, storage, and networking of the VMs are configurable.

Pricing and availability model for GCE is different as compared to AWS:

- Predefined machine types: pay fixed rate depending on the instance type per hour
- Sustained use discounts: discounted rate if instance is used for more than 25% of the month
- Committed use discount: pay according to the contract which includes usage on discounted

The present research uses High-CPU instances and choice of the zone is the nearest zone available, which avoids network and latency issues.

2.5 Related Works

Over the past few years' organizations have been moving towards Service Oriented Architecture (SOA) as cloud platform offerings has evolved into sophisticated services, with an agenda to innovate and meet the needs of the current applications. Some of the simple storage and network service offerings are now being extended to provide complex services compatible with other services being offered by the cloud provider. Amazon Web Services is an example of such a provider. We have seen that more cloud providers are moving towards higher-level services with more features and semantics [18].

Sahoo et al [11] and Graham et al [12] provides a detailed explanation of the evolution of Infrastructure-as-a-Service in area of cloud computing. Cloud benchmarks to assess the end-to-end performance over the internet and best practices of good benchmark to evaluate different cloud management software are discussed in [15], [16] and [17]. The obstacles to the adoption of cloud computing environment can be seen as an opportunity to further explore and research in this area. There is a tradeoff when adopting cloud services between vendor migration and cost savings. The danger of vendor lock-in [27] is inevitable without the standardization of the APIs and their interfaces [9]. This current ideology to focus on applications and services is being obstructed by vendor lock-in.

Open APIs like jClouds [20] and Dasein cloud [21], can be seen as standardization efforts across cloud providers. The top six obstacles mentioned in [13] for growth in cloud abstraction libraries are data lock-in, performance issues, storage, large-scale bug fixing, scalability and reputation of cloud provider. The paper aids in selecting features for comparison. Some of the other work related to feature comparison of cloud abstraction libraries is discussed in [13] and [14].

Cloud computing stems from the concept of virtualization and the improvement in the security over time are discussed in [19]. The following work does not discuss in detail some of the security features like secure HTTPS protocol to interact with the cloud provider. There is no standard architecture to implement a cloud abstraction library and all the libraries have varying implementation and varying level of support with the cloud provider. The need of cloud independent applications highlighted by Harmer et al. [29] can be achieved by developing the application as collection of services offered by different cloud providers and matching them to the requirements of the users. Authentication and data management are two of the biggest concerns that needs to be tackled.

Keahey et al. [30] envision a sky computing platform which is an aggregation of several cloud providers to create a large scale Infrastructure as a Service platform. They mainly focus on single network connectivity, security issues between different cloud providers and performance of varying hardware.

The design of the cloud abstraction libraries, such as Dasein cloud and jClouds are discussed in [21] and [20] respectively. Major focus is given to the implementation details, such as cloud provider support, how users can extend the library, project management and examples. In addition to these, this thesis discusses other qualitative and quantitative features, such as ease of use, and cost of common operations on AWS and GCE. Other open-source cloud management software, such as Delta Cloud, Apache Libcloud, Cloudloop and SimpleCloud are not included in the comparison. The main difference between the paper [26] and this thesis is that the paper analyzes the abstraction layer against the storage capabilities, whereas this thesis analyzes the compute and network capabilities of the abstraction layer.

CHAPTER THREE

IMPLEMENTATION

3.1 Feature Comparison

The abstraction layer is primarily used to help an application implement a solution in a provider-independent manner. The aim of this section is to critically evaluate features of jClouds and Dasein cloud as Open APIs against a set of attributes that affect the experience of a cloud user. The mentioned APIs are distinguished on the following features:

- Project start year
- Cloud Service Model
- Main philosophy behind the project
- Programming Language used
- Type of Web Service, For example: REST
- Compatibility with cloud providers
- Storage type supported by these projects
- Deployment model used
- License

FEATURES	jclouds
Year Started	2010
Service	Infrastructure as a Service (IaaS)

FEATURES	jclouds
Philosophy	“ Open source multi-cloud toolkit to create applications that are portable across clouds while giving you full control to use cloud-specific
Written In	Java
API	REST based API
Supported Cloud Providers	Compatible with almost all major cloud service
Storage type supported	Buckets and Blobs
Set up public, private or	Public and Private
License	Apache 2.0

Table 3.1 Features in jClouds

FEATURES	Dasein
Year Started	2009
Service	Infrastructure as a Service (IaaS)
Philosophy	Goal of the project is allow developers to “write-once, run against any cloud”
Written In	Java
API	REST based API
Supported Cloud Providers	Compatible with few major cloud service
Storage type supported	Blob
Set up public, private or	Public and Private

FEATURES	Dasein
License	Apache 2.0

Table 3.2 Features in Dasein

Dasein and jClouds were developed nearly at the same time and are written in Java. They both use REST architectural protocol to exchange information with the cloud provider. Some of the cloud providers supported by jClouds and are not supported by Dasein are CloudSigma, SoftLayer, Rackspace, ProfitBricks and Go2Cloud. Dasein supports blob storage only, whereas jClouds supports both bucket and blob storage. We can think of blob storage as a container and bucket storage as a folder or directory. Both of the abstraction APIs can be used to set up public or private clouds. There is a concern of security and further investigation is required.

3.2 Implementation Details

3.2.1 Setting up Multiuse Environments in the AWS cloud

This section provides an overview of how to set up and manage multiuser environments in the Amazon Web Services and provides information regarding configuration for the account. Separate accounts for each user is mostly common in graduate labs and among researchers who needs separate account environment with an option of consolidated billing.

Following are the scenarios to take advantage of AWS Multiuse environment:

- Least privilege: Access to resources based on credentials provided by account owner
- Limited privilege: Access to resources by using AWS management console
- Account ownership: Access to all resources and can have separate invoicing account

Multiuse environment is ideal for a scenario where the user is not responsible for setting up an account or paying for the resources being utilized on AWS. The permissions to access and use resources are limited by the admin user.

The first step in setting up a multiuse account is to create an admin user for AWS. Next step, is to create account for each individual user and setup Identity Access Management (IAM) to control access to the resources. IAM allows user to access and utilize compute, storage, and networking resources according to the access policy set the by admin user. It provides the flexibility to own an account by changing the default credentials, although the user is still restricted by the access policy set by admin user. Having a separate user for each member in a group helps to monitor resources on an individual basis and is beneficial for long term or shot term projects. For short term projects, the IAM user accounts can be deactivated after a certain time frame and for long term projects the ownership of the account can be transferred to the IAM user if the user wishes to continue working on the project and is not subjected to any prior intellectual property. The IAM policy is a set of document that explicitly states the set of permissions granted to a user, group or organization in AWS. The IAM policy can be explained in the following ways:

- Actions: What you can do? For example: start a virtual machine, start is the action
- Resources: What you can access? EC2 VMs are resource here
- Effect: Allow or Deny?

A) Account Setup

The administrator need to create a user group in AWS and have associated users as part of that group. Group will have a specific policy that allows access to compute, storage, and network resources. The administrator can also set up consolidated billing for the user.

The following information is required to create an IAM based access account for a group:

- AWS administrator account for the group. The account can be owned by an individual or an organization.
- User name and email address to be associated with AWS
- AWS account credentials for users who have existing AWS accounts can use their existing account for the setup. The security credentials are used to authenticate and authorize calls that we make to AWS. All AWS accounts have root account credentials. These credentials allow full access to all the resources in the account. We cannot control the privileges of the root account credential; so it is preferable to store them in a safe place and instead use AWS Identity and Access Management (IAM) user credentials for day-to-day interaction with AWS
- Following are the steps to create an Amazon account:
    - Enter “http://aws.amazon.com/ “in the URL of a browser and navigate to > Create an AWS Account
    - Enter login credentials
    - Provide all the required contact information, such as Name, Phone number, and Credit Card information
    - Authenticate and verify the newly created account via the code received on the provided phone number
    - Account will be activated
    - AWS provides public and private key pair to access products like EC2 or S3 buckets
    - Save Security Credentials key pairs and account keys

- AWS resources can be monitored using shell commands, Elastic Fox or Amazon dashboard

- List of the resources, services, applications and the action applicabe on them based on each user within the group.
- The administrator can also set up billing alerts preferences such as who will receive warning emails and usage reports

B) Cost Tracking and Monitoring Resources

AWS offers multiple ways to monitor and control costs. Administrator can choose the option of consolidated billing to track costs in a better way. The consolidated billing feature provides a detailed report on the resources being utilized by each IAM account. The IAM policies can be setup to grant access to limited users in terms of cost management. CloudWatch, along with a number of third party extensions like Cloudcheckr, can be used to setup billing alerts. Cloud billing management tools help to enforce monetary policies and make sure the IAM accounts created do not go over the threshold. The tools can be used with application-specific custom metrics to track performance and price. Cloud watch metrics are only aviavalble in the region in which they were posted.

C) Runtime Environment

The IAM users created are given a unique set of credentials to login for the first time on AWS. The IAM user can access resources based on the set of permissions assigned by the administrator. For example, if the user has access to EC2 resources, they can perform all the actions like starting a VM, stopping a VM, saving the state of a VM, etc. The IAM users do not have access to view resources of other users within the same group as they are independent accounts. The administrator can also setup the account expiry date for the IAM user account.

3.2.2 Setting up Multiuse Environments in Google Cloud Engine (GCE)

Google Cloud Engine is Google's IaaS offering in an increasingly crowded cloud computing market. The way that we manage and set up a multiuse account in GCE is different from that of AWS. GCE does not have any concept of Identity Access Management (IAM) or access policy for managing different levels of access control. We can give other users access to the project by adding them to our project and the level of access depends on the role the user has been added with. When we add a user to our Google Compute Engine project, it gives the user the same amount of access to the Google Compute Engine resources in that project, determined by the roles such as viewer, editor or owner. Adding, deleting or changing user's permission can be done through the permissions page in the console.

A) Account Setup

The administrator or owner of the project can add new members to the project by entering the email address of the new user associated with Google Cloud Engine. In case of AWS, it recommended to give least privileges to a new user, while in GCE it is recommended to give project ownership to all the members of the project. It is advised to use Google account for the all the members of the project, as when a user leaves, the admin user can mark the account suspended. GCE also allows the admin user to reassign project and give ownership of documents and apps to other users within the project.

Creating user accounts for the Linux virtual machine instances so that we can connect to instances using a unique username and a personal home directory to perform necessary tasks is different from AWS. A user account can be assigned to anyone who has read access to your project. In particular, user accounts are useful for granting project viewers SSH access to the instances, without giving them additional permissions as an editor or owner. If there are project

viewers who need to connect to virtual machine instances using SSH, the best way to do this is by creating a user account for them.

The following information is required for creating a new user account through the cloud platform console:

- To create a new user account, choose an account username and assign the account owner. The account username must be unique within the project and is used to log into instances. The account owner is a Google account that is responsible for the user account. An account owner can upload a public key to the account and start using it.
- By default, most virtual machine instances are created with a default service account. A service account authenticates applications running on the instances to other Google Cloud Platform services. Using service account credentials, the application can authenticate seamlessly to other APIs without embedding any secret keys or credentials in the instance, image, or application code. Because service accounts can be used to authenticate to other services, users who can connect to an instance can effectively act as the service account and gain access to other Google APIs. As such, a project owner must explicitly grant the account owner permission to an instance's service account using Google's Identity and Access Management tools or the account owner will not be able to connect to the instance. There are two exceptions to this requirement:
    - The account owner is also a project editor or owner, in this case they already have read-write access to all service accounts in the project.
    - The instance does not have any service accounts enabled and therefore, has no ability to use service account credentials

- After an account is created, the account owner can add a public key and connect to an instance. If the account owner uses gcloud, the tool automatically generates a public/private key pair for the account and connects user to the instance

B) Cost Tracking and Monitoring resources

We can set project wide billing alerts to track costs and monitor bills. Unlike AWS, we cannot restrict individual users on the basis of cost to use a particular resource. The Google Cloud Monitoring dashboard can be used to alert cloud powered applications. Two unique features of Google Cloud Monitoring are resources and groups. A resource is an abstract object provided by certain products, platforms, or services within Google Cloud Platform. For example, Google Compute Engine provides a resource called VM instance, Cloud SQL provides a database instance, Cloud Publisher/Subscriber provides topics, and so forth. A group is a collection of resources, such as "all VM instances in my project that are running Cassandra". Cloud Monitoring automatically detects related resources and groups them together.

C) Runtime Environment

GCE has a functionality where we can make objects available to specific users and can also work at a granularity where we can specify permissions for individual objects in the resources. All resources are set up by the project owner's group.

Project owners are granted `OWNER` permissions automatically to all buckets inside their project. When we create a project, we are automatically added as a project owner.

3.3 Downloading and Configuring jClouds

The experimental environment used to set up a jClouds library consists of a windows machine with Windows 10 Home installed, a system with 6.00 GB of installed memory (RAM), and a 64-bit operating system with x64-based processor. The configuration management system

used for jClouds is Maven. Maven is a software project management and build tool. In case of jClouds, Maven handles project's build, reporting and documentation from a central piece of information. Maven 3.02 or higher version is required in order to download the dependencies and the required jars for the project to run.

3.3.1 Pre-requisites

- Java 7 Oracle JDK edition or OpenJDK
- Git
- Eclipse 3.4 or higher
- m2eclipse (m2eclipse is included in the *Eclipse IDE for Java Developers* package for Eclipse 3.7 (Indigo) and higher)
- TestNG plugin for Eclipse
- A local clone of jClouds/jClouds
- If working on jClouds-labs projects, a local clone of jClouds/jClouds-labs

3.3.2 Adding jClouds to Eclipse

- Import jClouds projects as Maven projects. We can optimize our workspace by excluding the unneeded projects (such as those without a src folder)
- Accept any warnings about Incomplete Maven Goal Execution related to the maven-remote-resource-plugin: bundle goal for the jClouds-resources project. We can either:
    - Ignore the error, which will not affect our builds
    - Remove the jClouds-resources project from our workspace, unless developing against it
    - Right-click on the build error and use the `Quick Fix` option to ignore the `bundle` goal

- Add this configuration snippet to our jClouds-resources POM file and refresh the project in Eclipse

- If working on jClouds-labs, import the jClouds-labs projects as Maven projects

3.3.3 Building from command line

To build the project we need to run “mvn package”, which takes the compiled code, packages it in its distributed format such as a JAR, and puts it in the target directory. In particular, the traditional way to run the code is through the command line interface, which involves invocation of the jar file along with the arguments specific for the cloud. In case of AWS, we need to specify public access key, secret key and security group name. The access keys are the ones get downloaded during the setting up of AWS account and a *security group* controls the incoming and outgoing traffic in a Virtual Private Cloud (VPC). The security group has a set of inbound and outbound rules and acts as a virtual firewall that controls the traffic for one or more instances. All the new instances that are started within a same VPC are applied with the same set of inbound/outbound rules based the security group policies.

3.4 Downloading and Configuring Dasein

The environment used to set up Dasein library is identical to that of used in jClouds. Following are the steps to get started with Dasein.

3.4.1 Cloning the skeleton

The best way to do this is to create a copy of dasein-cloud-skeleton in a Github account and work on it from there. We do not really want to fork dasein-cloud-skeleton since we are not actually forking it, but instead we are creating a new project. The other way would be to download the skeleton to a local machine.

3.4.2 Alter the project information

Dasein uses Maven for configuration management and build tool. The first step is to do a global search for 'skeleton' and replace it with the name of the cloud provider. We also want to edit the contributor list so it has our name in it. If we do not intend to contribute the code back to Dasein Cloud, we need to edit the copyright information in the pom.xml and src/etc/header.txt class as well as remove the LICENSE-APACHE.txt file and insert our own.

3.4.3 Rename the packages and class

The skeleton project includes packages in src/main and src/test for org.dasein.cloud.skeleton. The following directories should be renamed and Java classes under those directories should start with the package name. All package references within the sample Java classes should also be changed. We may optionally rename the classes (like MyCloud.java) to something more descriptive.

3.4.4 Abstract Cloud Interaction

The Dasein Cloud implementation does API tracing and wire logging of all API interaction. If we plan to contribute back to the Dasein Cloud, it is a requirement for any implementations contributed back to Dasein Cloud and is therefore best to abstract out all cloud interaction into a single call that does API tracing and wire logging.

3.4.5 Create the Context Test

The most basic function in a Dasein Cloud implementation is testContext() method in the provider object (MyCloud.java in the skeleton project). It does two things:

- Verifies the configured API access credentials
- Returns a canonical account number for the connection

3.4.6 Implementing Datacenter Services

The one set of services that all Dasein Cloud implementation requires is DataCenterServices. MyCloud.java already contains a getDataCenterServices() method that returns an instance of Geography.java. We should therefore implement Geography.java as our first programming task.This class is generally the easiest and the hardest to implement. It is the easiest because it has fewest methods you absolutely must edit. It is the hardest because the model mapping is generally not straightforward.

In DataCenterServices, the task is to map the Dasein Cloud concepts of "region" and "data center" to whatever geography exists with the underlying cloud provider. In AWS, it is a simple task because AWS region = Dasein Cloud region and AWS availability zone = Dasein Cloud data center. Very few other clouds have this kind of division. In most cases, we end up with one or more regions that each maps to a single data center. Or we have a single region that maps to multiple data centers. Here is the rule for on how to map concepts:

- A region must fit within a single jurisdiction
- No resources are shared across regions
- Some resources are shared across data centers
- Data centers represent physical proximity
- Block volumes may not be attached to virtual machines in another data center (NFS volumes are not so constrained)

3.4.7 Implementing other services

All other services are optional, within reason. Typically, the process works like this:

- Create a package for the service (like org.dasein.cloud.aws.compute for Compute Services)

- Create a class that extends the abstract version of the service in that package (like AWSCompute extends AbstractComputeServices)
- Implement a constructor that takes your cloud provider as an argument
- Create a getWhateverServices() method in your cloud provider class (i.e. MyCloud.java in the skeleton). The method needs to match the signature in CloudProvider since we are overriding a default method that returns null. In the case of compute services, it is getComputeServices().

Implementing support classes is where the real work takes place. A service object is simply an enumerator of what resources that services support. For example, compute services can optionally support images, virtual machines, volumes, snapshots, and more. To implement support for virtual machines, we might create a VM sub-package and then create a MyCloudVMSupport class that implements VirtualMachineSupport. We then override getVirtualMachineSupport() method in our compute services class and implement all of the VirtualMachineSupport methods in our MyCloudVMSupport.java class to do everything necessary to support VM interaction.

3.4.8 Testing the implementation

Dasein Cloud includes a test suite (already referenced in the skeleton pom.xml) which can be used to verify that the implementation of Dasein Cloud is working the way we expect. To run the full test suite use the following command:

Mvn –Pmyprofile clean test

Note on Debugging:

It is easy to debug code through integration tests, since they are in essence just the JUnit tests. Individual tests can be run with Maven with `-Ddasein.inclusions` parameter, and for debugging it is important to instruct Maven not to fork additional process.

CHAPTER FOUR

EVALUATION AND DISCUSSION OF TRADEOFFS

An open source project needs to be evaluated on a number of factors like code activity, bug tracking, user support, documentation, release cycle, etc. The selected features are qualitative and quantitative methods that target small scale or large scale deployment. The criteria are grouped into two main categories: performance and developer experience. Performance related criteria focus on load testing, cost of common operations such as starting an instance on AWS and GCE under varying configurations. Developer experience criteria include user's experience while using the cloud abstraction libraries, ease of setup, accessibility and community support.

4.1 Load Testing

The most common question regarding an APIs performance is not how fast the translation is or how it scales, but something more fundamental, i.e. the performance goal in terms of concurrent requests. Should it be one hundred concurrent requests? A thousand? Ten thousand? Does it require a single resource or a hundred to handle the load? The performance of an application depends on the interaction between the frontend client making web requests to the backend API. If the response time or error rate is high for backend applications then the load time of the application will be higher. This also increases delay in page view; Hence making the application slower to respond. The performance metric is motivated by the Internet of Things (IoT) use case. The IoT environment is typically used to deploy and monitor thousands of physical entities. An IoT system is a part of a larger system encompassing several other functions. Performance is one of the key factors in an IoT environment. In an IoT environment, all the data collected by the devices are typically sent to a central server. The central server then

needs to share the data with the services and business logic server to make it available to the end user. It would be really extensive and out of the scope to test all the end points of jClouds and Dasein cloud web service. In this case we selected to test two main endpoints for the API:

- /create – creates a virtual machine and returns a virtual machine id.
- /deletes –deletes a virtual machine and returns NULL.

4.2 Load Testing Set up

The tests performed for the concurrent requests are useful as the workload being tested is as similar as possible to the IoT use case. We envision that the edge devices will be placed in a public or private cloud with an orchestration layer at the top level of the architecture. The orchestration layer will orchestrate and assign work to the edge devices through a cloud abstraction layer such as jClouds or Dasein cloud. The edge devices can be one of the following:

- Virtual machines in private or public cloud
- Containers in a private or public cloud

In our experiments, we consider the case of virtual machines. We will start virtual machines in AWS and GCE through jClouds and Dasein libraries. The traffic that we are going to simulate during the test is also an important point to consider. Following are the three types of traffic that can be generated:

- Generating a load of millions of requests at fixed interval of time
- Mocking real traffic
- Directing real user traffic

Our approach for the current work is to load test the end points using repetitive load generation and make sure the environment is stable for functional tests. Load testing the

endpoints using this approach will also help us to find out a threshold and identify bottleneck in the application. An ideal scenario to do regression load testing is to monitor the number of requests and response of the endpoint over the network. Since, in this case the client code and the API endpoint are on the same machine, such a test would be redundant.

Inspecting the code for Dasein and jClouds and performing some simple experiments like using a multi-threaded program to start virtual machines on AWS EC2 or GCE, we concluded that the translation of various cloud APIs takes place in the cloud itself rather than taking place on the endpoint. We believe that this increases latency and reliability as the translation is not done by the open source software.

4.3 Virtual Machine Startup Time

The startup time of a VM [31] is defined as the time taken for a scheduler to allocate resources and time taken by the VM to boot the OS and startup applications. The following experiments do not run any custom applications during startup. To measure the startup time correctly we do not rely on the different phases/states for the VM instances as a “running” instance on AWS may not necessarily mean the same as “running” state of an instance on GCE.

A virtual machine is provisioned by an EC2 scheduler in case of AWS. The scheduler provisions the requested resources from the user such as RAM, memory, number of cores, RAM, type of OS. Figure 4.1 depicts the VM startup process after a request is made by scheduler to start a VM.

The status tag provided by the cloud provider is not a reliable way to see whether the instance is in running state or not. For our experiment, we use the time difference when a request is made and the time we are able to make first successful SSH login. There are a variety of systems that employ software and hardware enhancements to reduce the VM start up time. Some

of the popular approaches are discussed in [32], [33] and [34]. In our thesis the startup time is measured without any optimizations and enhancements

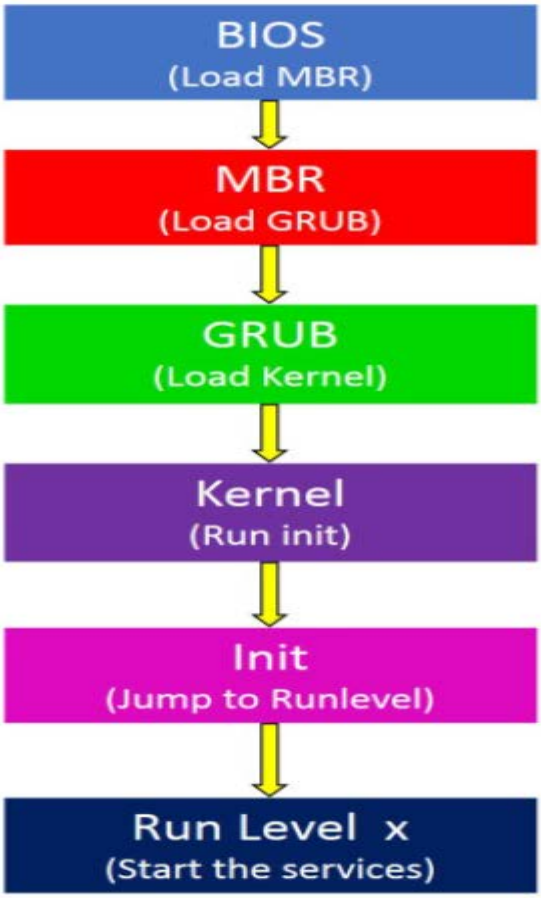

Figure 4.1 Virtual Machine start up process.

.

4.3.1 Instance Lifecycle on AWS

The following illustration (Figure 4.1) represents the transitions between instance states.

1) Pending: It is the first state that the user sees on the dashboard when an EC2 instance is started. The matching of our required specifications/configurations to the available hardware happens during this stage.
2) Running: From pending state the instance enters a running state. In this state the instance is ready to connect to and we can use ssh or command prompt to connect to the VM.

3) Stopping: Stopping an instance halts the execution of the applications running on the VM. This state cannot be triggered directly by a user and is seen when an instance changes from running state to stopped state.

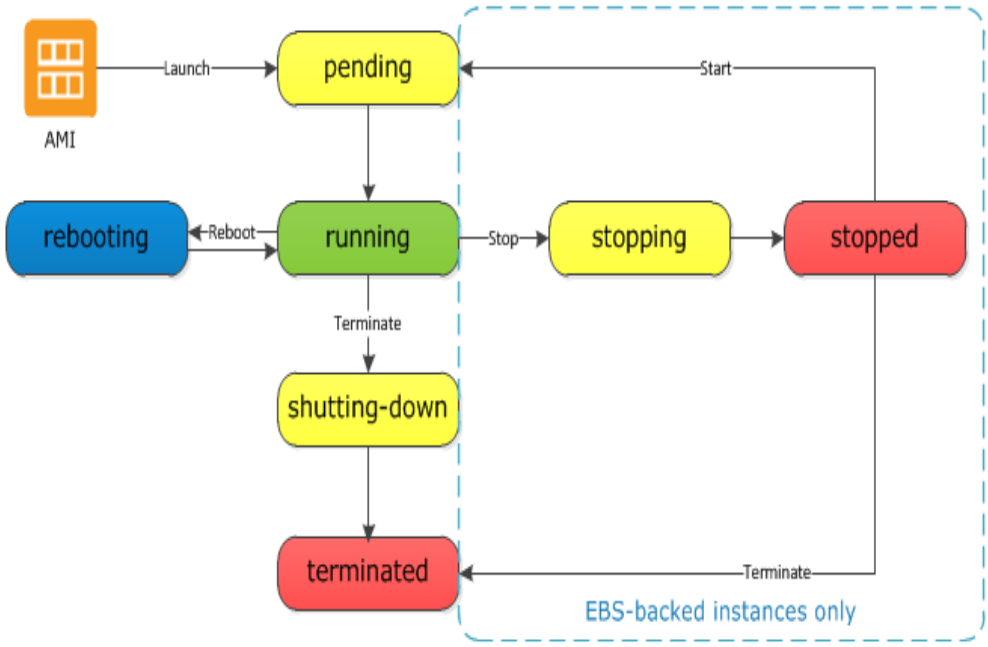

Figure 4.2: Instance Lifecycle on AWS

4) Stopped: In stopped state all the applications and services installed are stopped. Users are not being charged when an instance is in stopped state. Any other resources attached to the instance, for example Amazon EBS volume may continue to charge. The data stored in VM and any volume attached to the instance is not lost in this state.
5) Rebooting: When the user selects reboot option for an instance, the VM loads the boot loader which in turn restarts the operating system. Any temporary or unsaved data is lost after rebooting. The networking and storage resources attached to the VM are not affected. When the VM starts restarts, it is in the same Virtual Private Cloud or in the

same set of IP range. We can also configure the “on reboot” option, which can perform any user specified action after rebooting.

6) Terminated: When an instance is terminated the connection to the VM is lost and the attached networking and storage resources can no longer interact with the VM. Once the VM enters terminated state, no charges are incurred to the user.

4.3.2 Instance Lifecycle on GCE

Instances running on GCE can be in the following states:

1) Provisioning: Resources are being reserved for the instance. The instance isn't running yet.
2) Staging: Resources have been acquired and the instance is being prepared for launch.
3) Running: The instance is booting up or running. We should be able to SSH into the instance soon, though not immediately, after it enters this state.
4) Stopping: The instance is being stopped either due to a failure, or the instance being shut down. This is a temporary status and the instance will move to terminated.
5) Terminated: The instance was shut down or encountered a failure, either through the API or from inside the guest. You can choose to restart the instance or delete it.

4.3.3 Measuring Startup Time of Java Virtual Machine

The startup time of an instance provisioned by the cloud abstraction API is affected by, the time it takes to boot up the Java Virtual Machine (JVM) along with libraries of the API. In this section we will measure the startup time for JVM, Dasein cloud and jClouds library. The approach used is to loop over the program multiple times and calculate the average execution time. There are a number of factors like CPU caching, OS caching, and java garage collection that could skew the execution time.

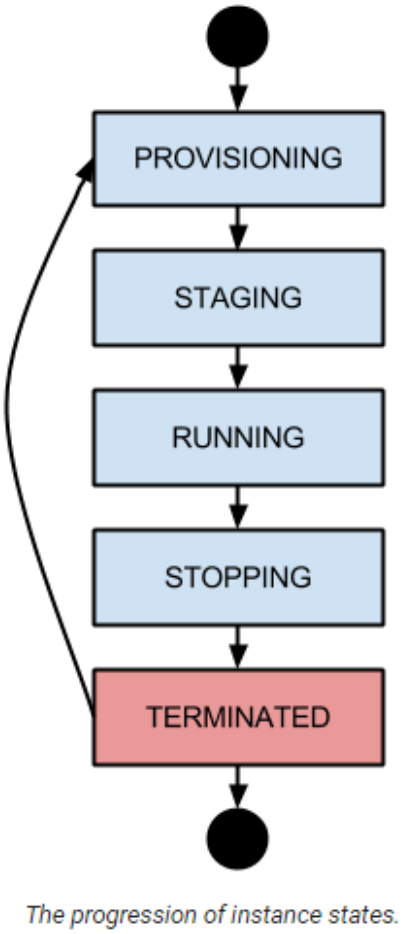

Figure 4.3 Instance Lifecycle on GCE

The results can also vary depending on the target environment where the JVM and libraries are being run. These are some of the disadvantages of using such a naïve method.

We measure the startup time of each run as it is happening. To accomplish this, we do not launch the user's Java class directly. Instead, we run a PowerShell script that does the following:

- Uses "Measure-Command" to measure the execution time of arguments passed. The argument is a command to run a jar file
- Get results for each iteration in an array
- Average the values of the results array

The results from the experiment conducted to calculate the startup time for jClouds and Dasein are shown in Figure 4.4. The two different scenarios are: A) The PowerShell script measures the time for the JVM, Dasein library and jClouds library to load up and not print anything to the console. B) The PowerShell scripts measures the time for the JVM, Dasein library and jClouds library to load up and print “Hello World” to the console. Each experiment is run 100 times and we represent mean and standard deviation on the bars.

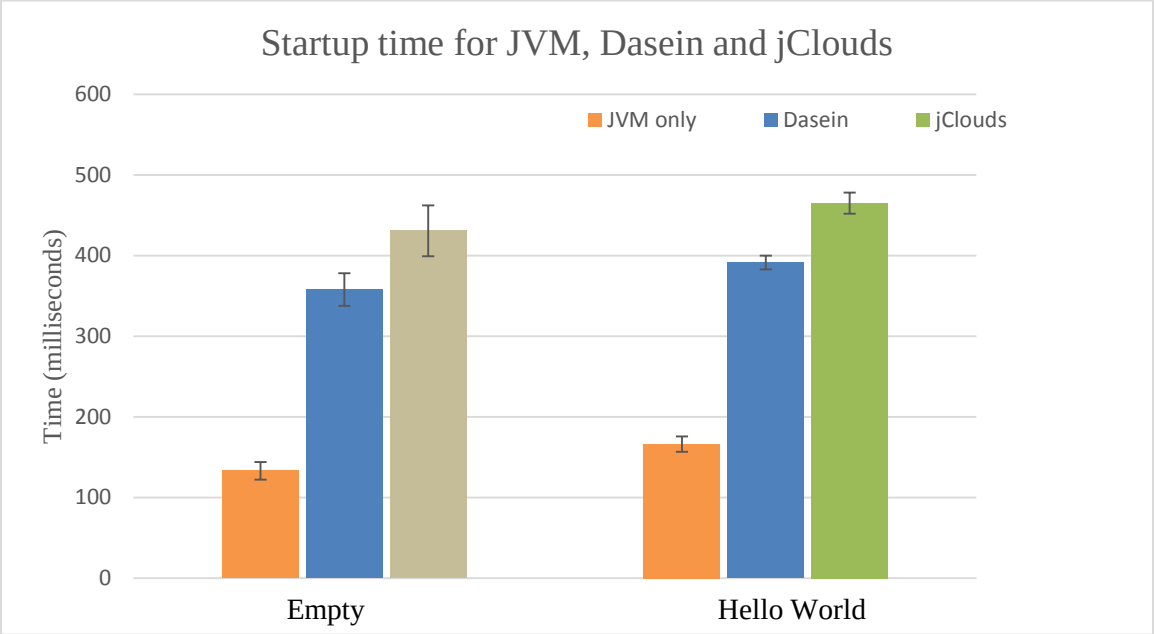

Figure 4.4 Startup time of JVM, Dasein and jClouds

The startup time for Dasein library in Table 4.1 is calculated by the difference between the Dasein startup time and JVM startup time. We observe that, it takes 1.7 times greater to load up the Dasein library as compared to just loading the JVM. For jClouds it takes 2.2 times greater to load up the library, as compared to loading just the JVM. jClouds take a longer time to load up because of the way the project is being setup. The classes are nested and there is hierarchical function call from the main class to the core class.

The solution time is the time taken to print “Hello World” on the console. The results show that it is consistently between 30 to 35 ms. This also helps to validate the JVM startup time.

	JVM	Dasein	Jclouds	Startup time for Dasein (ms)	Startup time for jClouds (ms)
No output to the console (ms)	133.03	358.06	430.79	225.03	297.76
Hello World (ms)	166.08	391.53	465.08	225.45	299.00
Solution time (ms)	33.05	33.47	34.39		

Table 4.1 Startup time for JVM, Dasein and jClouds

4.3.4 Measuring Startup Time on AWS and GCE

There are a number of factors like geographical location of the datacenter, type of instance requested, number of instances per request, and request of additional resources like networking and storage along with the compute instances that could affect the startup time of a VM on a cloud provider. For this experiment we measure the startup time on AWS EC2 and GCE. We will examine the cost of common operations such as VM start up time based on different factors.

The experiments are conducted in the availability zone 1 of the US East region for EC2 and us-east 1 zone B in case of GCE. We use PowerShell script to measure the startup time. The argument passed to the “Measure-Command” is a jar file for the abstraction layer library along with different configurations we want the VM to start with.

4.3.5 Startup time by VM Instance Type

The startup time of an instance is dependent on the instance type. The following experiment quantitatively measures the relationship between the VM startup time and the VM instance type. For example, EC2 splits up instances into six families with a total of 29 instance types, whereas GCE splits its instances into four families with a total of 15 instance types.

The instances listed in table 4.2 comprise varying combinations of CPU, memory, storage and networking capacity. From a broad selection of instance types, we have chosen most commonly used instances such as standard, high memory and high CPU types. This allows us to look at the scaling aspect of the use case.

AWS Instance Type	CPU Cores	RAM	GCE Instance Type	CPU Cores	RAM
m3.medium	1	3.75	N1-standard-1	1	3.75
m3.2xlarge	8	30	N1-standard-8	8	30
r3.4xlarge	16	122	N1-highmem-16	16	104
c3.8xlarge	32	60	N1-highcpu-32	32	28.8

Table 4.2 AWS and GCE instance type used to measure startup time

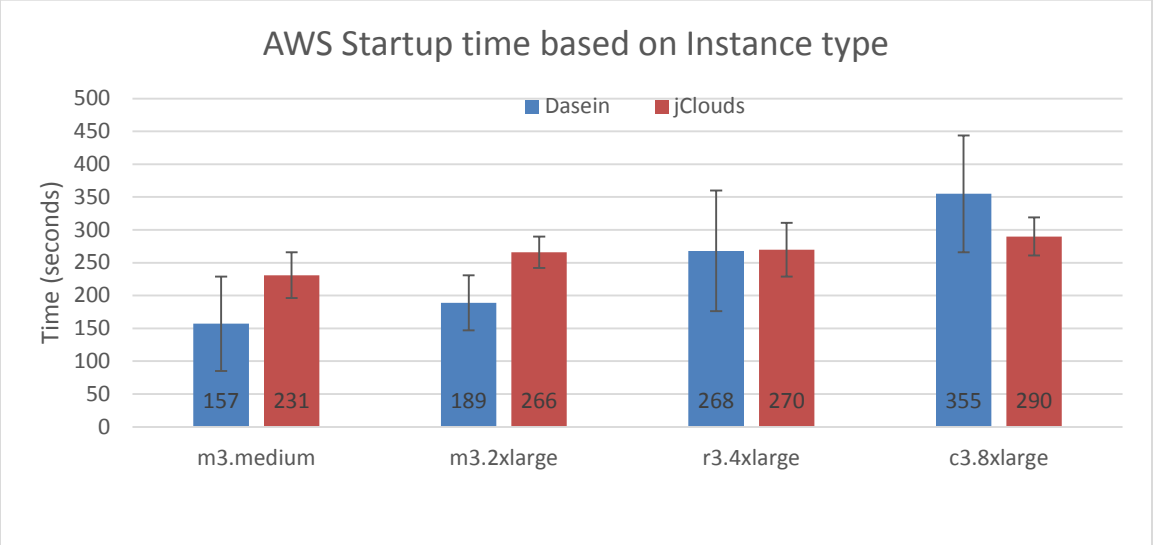

Figure 4.5 AWS startup time based on instance type

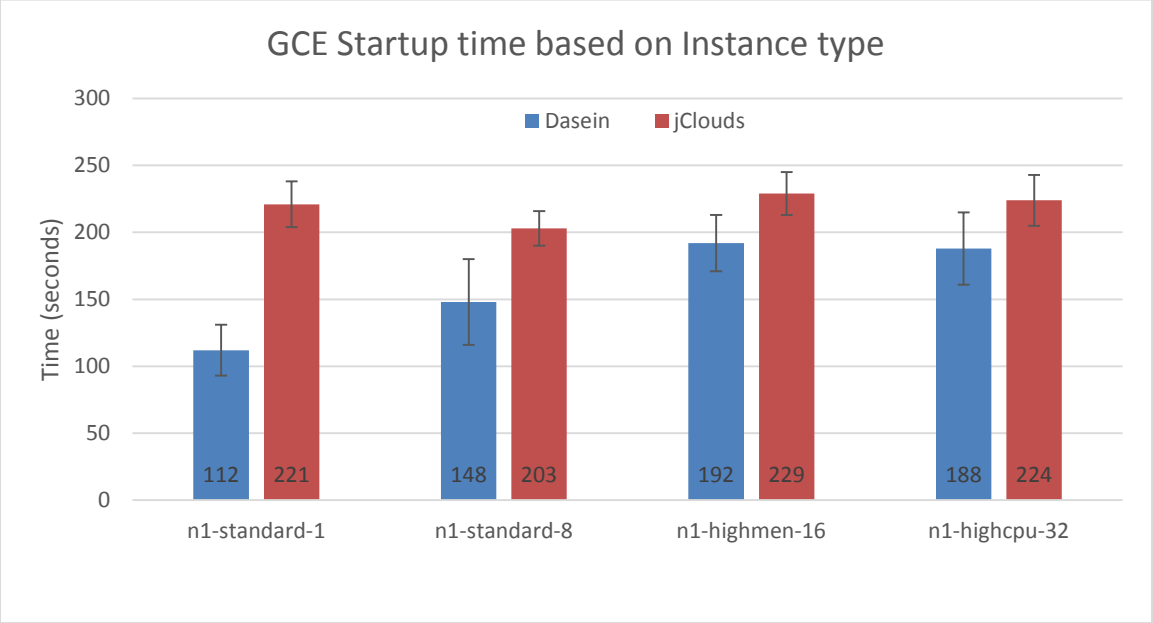

Figure 4.6 GCE startup time based on instance type

Figure 4.5 and 4.6 shows AWS and GCE startup time based on instance type respectively. We can interpret the following points from the experiment conducted:

- Is Dasein cloud faster than jClouds?
- Which abstraction layer is more consistent?
- Startup time on AWS vs GCE for similar instance type

In AWS and GCE cloud provider, Dasein cloud performs better as compared to jClouds, except for the case in AWS cloud for c3. xlarge instance type where jClouds outperforms Dasein cloud. The experiments conducted show that, jClouds is relatively more consistent than Dasein cloud in case of AWS and GCE. The startup time may be more than Dasein cloud, but it is consistently same for various instance types in AWS and GCE. The startup time for similar instance type is less in GCE as compared to AWS.

4.3.6 Startup time by location

The geographical location of the datacenter affects the communication latency between the client application and the VM. In this experiment, we focus on the cloud VM startup time by different locations. The region and location of the data center locations of AWS and GCE are summarized in Table 4.3. We will measure the startup time of m3. medium and n1-standard-1 across different data centers in AWS and GCE respectively.

AWS		*GCE*	
Region	***Location***	***Region***	***Location***
us-east-1	Northern Virginia	Eastern US	Berkley County, South Carolina
eu-west-1	Europe (Ireland)	Western Europe	St. Ghislain, Belgium
ap-northeast-1	Asia Pacific (Tokyo)	East Asia	Changhua County, Taiwan

Table 4.3: Region and location in AWS and GCE

Figure 4.7 and 4.8 shows the AWS and GCE startup time based on the regions respectively. The experiments conducted validated our statement that the startup time in jClouds is consistent and does not vary a lot as compared to Dasein cloud. The startup time is highest in Eastern region in AWS showing that most of the AWS traffic is run on Eastern region. The startup time is highest in Western Europe, in case of GCE. The high startup time in Western Europe is probably due to Geographical distance and latency in the network.

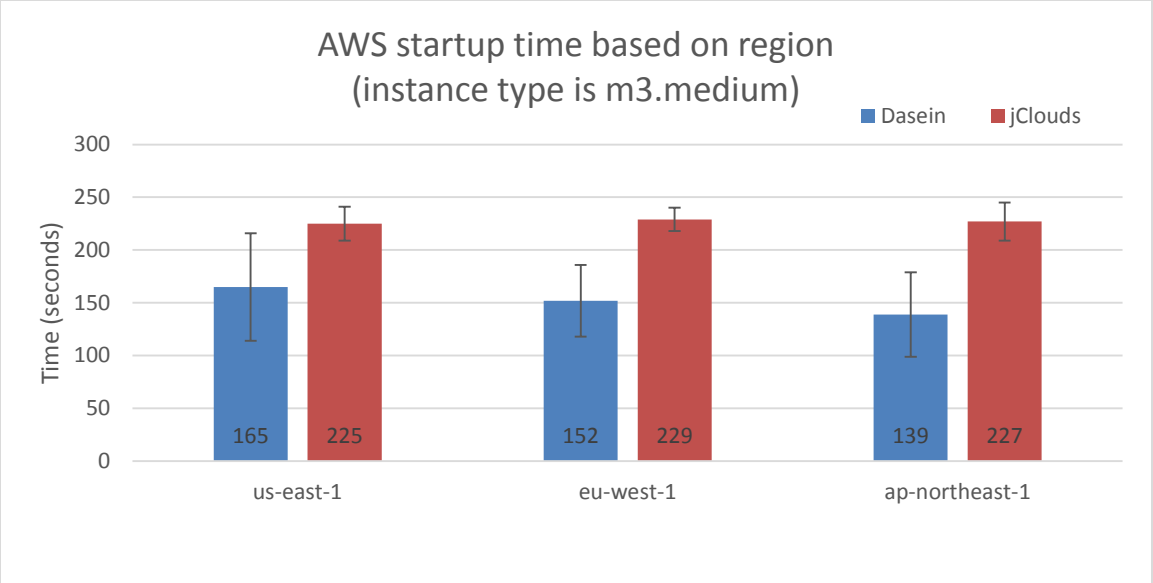

Figure 4.7 AWS startup time based on region.

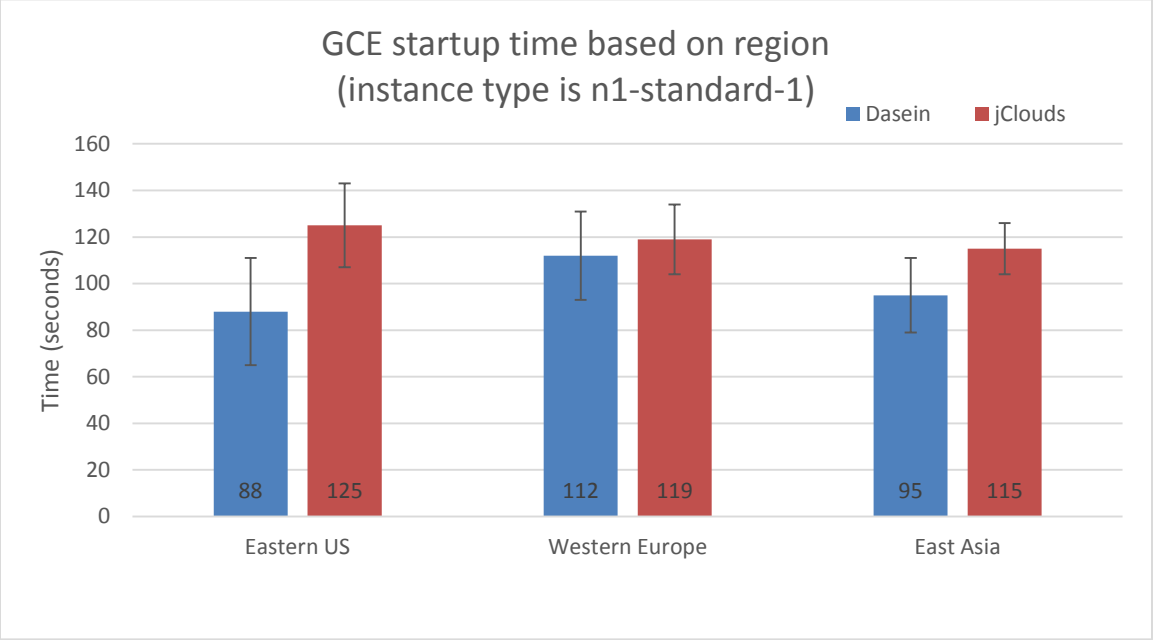

Figure 4.8 GCE startup time based on region.

4.4 Developer Experience

Cloud abstraction API can be used to deploy and manage services across multiple cloud provider. It is difficult for organizations and developers to keep up with the development of various cloud orchestrators to manage a single cloud provider. A translation layer like Dasein or

jclouds could reduce the effort to develop application for a multi cloud environment. The open standards are not yet widely accepted by major cloud providers and right now developing a wrapper application for different cloud platforms seems to be a potential solution. We learned this when implementing our platform. The abstraction layer can perform some of the basic capabilities like compute actions such as start, stop, etc; data transfer actions such as upload and download; networking actions such as adding or deleting a network interface. Although, some of the functionalities are supported by Dasein clouds and jClouds, there are several inconsistencies and pitfalls. Some of them became easy to identify while, others were hard to spot due to missing or insufficient information. The “available locations” action is not consistent between Dasein and jclouds. Dasein shows different behavior in listing availability zones with the same cloud provider. jclouds has incomplete implementation for some of the network related functionalities like Content Distribution Network (CDN).

The jClouds project is fairly easy for new users. The webpage is well documented and the build and install commands in Windows environment work as they are supposed to work. While, on the other hand, setting up Dasein project is not an easy task. The pom.xml file used to build the project uses some of the dependencies that have either been removed from the Maven repository or are outdated. Dasein has one basic example to connect to a cloud provider that does not cover all situations. For example, Google cloud uses a different authentication and authorization mechanism as compared to Amazon EC2, and there are no examples for Dasein. In contrast the jClouds library is well documented to guide through a different implementation for GCE. Dasein clouds do not have enough information for understanding how to implement the GCE cloud provider just by reading the documentation.

The jClouds library is an active project and has very good community support. There is a chat forum that we can join and the members are active enough that we can get a reply within 24 hours. Dasein cloud have a Google group for user queries. With our experience of using it for more than two years, we will say there is no fixed time for the answer.

Both jClouds and Dasein can be used in command line interface or GUI (Eclipse IDE) to build and run classes. The dependencies are more complex in Dasein cloud and we recommend using the Eclipse IDE to build and run the project.

CHAPTER FIVE

CONCLUSION AND FUTURE WORK

A service that operates entirely in cloud is responsible for service operational maturity, DevOps enablement, incident and problem management, and capacity management. Many organizations are moving towards micro-services and cloud services architecture. It is very important for an infrastructure platform to offer a high-quality cloud computing environment consistently across multiple cloud platforms. To enable this, a collaborative yet an independent cloud abstraction service is required. In this thesis, we conducted experiments using Dasein cloud and jClouds as the cloud abstraction layer under test. It supports automatic configurations and deployment of virtual machines to pre-defined services. The cloud abstraction layer helps to enhance functionality around data center, infrastructure and operations.

The following is a discussion of the results presented in chapter 4. Dasein cloud and jClouds were tested against Amazon EC2 and GCE cloud services to calculate the cost of VM startup under varying conditions. The expected startup time of Java Virtual Machine (JVM) is not significant and highly variable (see Table 4.1 for results). Our technique would be valuable for measuring startup time for libraries. The measured startup time for JVM was around 100 ms, Dasein 225 ms and jClouds 300 ms. The startup time was consistent across different programs and different runs. We observed the variation between startup time due to the run time consumed by Dasein or jClouds.

The second experiment focused on the relationship between the VM startup time and VM instance type. The different instance types used on AWS EC2 and GCE are summarized in Table 4.2. The increase in the startup time for various instance type does not correspond to the configuration. For example, the startup time for next larger configuration may not be twice as of

the previous configuration. In case of Dasein with AWS EC2 and GCE, the startup time increases almost linearly as the configuration are doubled, while in case of jClouds the startup is consistent even when VM configurations are changed.

The third experiment collected the VM startup time by different locations. Table 4.3 summarizes the data center locations of EC2 and GCE. The experiments were conducted with m3. medium on AWS EC2 and n1-standrad-1 on GCE. Apart from the geographical distance that can increase the latency, we think the number of users on a particular data center will also affect the startup time. The startup time is highest in the Eastern region in AWS and Western region in GCE.

This thesis also presented lessons learned while implementing Dasein cloud and jClouds. We have seen inconsistencies around the implementation of cloud abstraction libraries to support features offered by the cloud providers. Also, there is lack of development around the tool to monitor and identify bottleneck in the abstraction libraries. We are also carrying out more research with respect to an orchestrator that would be able to do multi-cloud deployment, discovery of resources and monitoring of the VMs running in the cloud provider. We envision our orchestration layer, to deal with failures and share the collected information from all cloud providers to make a unified decision.

REFERENCES

1) Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." (2011): 20-23.

2) Armbrust, Michael, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee et al. "A view of cloud computing."*Communications of the ACM* 53, no. 4 (2010): 50-58.

3) Petcu, Dana. "Portability and interoperability between clouds: challenges and case study." In *Towards a Service-Based Internet*, pp. 62-74. Springer Berlin Heidelberg, 2011.

4) Ortiz Jr, Sixto. "The problem with cloud-computing standardization."*Computer* 7 (2011): 13-16.

5) Gonidis, Fotis, Anthony JH Simons, Iraklis Paraskakis, and Dimitrios Kourtesis. "Cloud application portability: an initial view." In *Proceedings of the 6th Balkan Conference in Informatics*, pp. 275-282. ACM, 2013.

6) Silva, Gabriel Costa, Louis M. Rose, and Radu Calinescu. "Towards a model-driven solution to the vendor lock-in problem in cloud computing." In*Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 1, pp. 711-716. IEEE, 2013.

7) Satzger, Benjamin, Waldemar Hummer, Christian Inzinger, Philipp Leitner, and Schahram Dustdar. "Winds of change: From vendor lock-in to the meta cloud." *IEEE internet computing* 1 (2013): 69-73.

8) Sellami, Mohamed, Sami Yangui, Mohamed Mohamed, and Samir Tata. "PaaS-independent Provisioning and Management of Applications in the Cloud." In *2013 IEEE Sixth International Conference on Cloud Computing*, pp. 693-700. IEEE, 2013.

9) Petcu, Dana, Georgiana Macariu, Silviu Panica, and Ciprian Crăciun. "Portable cloud applications—from theory to practice." *Future Generation Computer Systems* 29, no. 6 (2013): 1417-1430.

10) HostMonk, All Rankings of Cloud Providers. http://www.hostmonk.com/ranks/cloud
Armbrust, Michael, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee et al. "A view of cloud computing."*Communications of the ACM* 53, no. 4 (2010): 50-58.

11) Sahoo, Jyotiprakash, Subasish Mohapatra, and Radha Lath. "Virtualization: A survey on concepts, taxonomy and associated security issues." In*Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pp. 222-226. IEEE, 2010.

12) Graham, Steven Thomas, and Xiaodong Liu. "Critical evaluation on jclouds and cloudify abstract APIs against EC2, azure and HP-cloud." In *Computer Software and Applications*

Conference Workshops (COMPSACW), 2014 IEEE 38th International, pp. 510-515. IEEE, 2014.

13) Bubak, Marian, Marek Kasztelnik, Maciej Malawski, Jan Meizner, Piotr Nowakowski, and Sumir Varma. "Evaluation of cloud providers for VPH applications." In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pp. 200-201. IEEE, 2013.

14) Verma, Akshat, Gautam Kumar, and Ricardo Koller. "The cost of reconfiguration in a cloud." In *Proceedings of the 11th International Middleware Conference Industrial track*, pp. 11-16. ACM, 2010.

15) Ueda, Yohei, and Toshio Nakatani. "Performance variations of two open-source cloud platforms." In *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, pp. 1-10. IEEE, 2010.

16) Koeppe, Falk, and Joerg Schneider. "Do you get what you pay for? using proof-of-work functions to verify performance assertions in the cloud." In*Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pp. 687-692. IEEE, 2010.

17) Cooper, Brian. "The prickly side of building clouds." *Internet Computing, IEEE* 14, no. 6 (2010): 64-67.

18) Sahoo, Jyotiprakash, Subasish Mohapatra, and Radha Lath. "Virtualization: A survey on concepts, taxonomy and associated security issues." In*Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pp. 222-226. IEEE, 2010.

19) https://jclouds.apache.org/start/concepts/

20) https://github.com/greese/dasein-cloud/wiki#project-information

21) http://deltacloud.apache.org/

22) http://libcloud.apache.org/

23) https://java.net/projects/cloudloop

24) http://www.ibm.com/developerworks/library/os-simplecloud/

25) Hill, Zach, and Marty Humphrey. "CSAL: A cloud storage abstraction layer to enable portable cloud applications." In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pp. 504-511. IEEE, 2010.

26) Ananthanarayanan, Rajagopal, Karan Gupta, Prashant Pandey, Himabindu Pucha, Prasenjit Sarkar, Mansi Shah, and Renu Tewari. "Cloud analytics: Do we really need to reinvent the storage stack?." In *HotCloud*. 2009.

27) Petcu, Dana, Georgiana Macariu, Silviu Panica, and Ciprian Crăciun. "Portable cloud applications—from theory to practice." *Future Generation Computer Systems* 29, no. 6 (2013): 1417-1430.

28) Harmer, Terence, Peter Wright, Christina Cunningham, and Ron Perrott. "Provider-independent use of the cloud." In *Euro-Par 2009 Parallel Processing*, pp. 454-465. Springer Berlin Heidelberg, 2009.

29) Keahey, Katarzyna, Mauricio Tsugawa, Andrea Matsunaga, and José AB Fortes. "Sky computing." *Internet Computing, IEEE* 13, no. 5 (2009): 43-51.

30) Mao, Ming, and Marty Humphrey. "A performance study on the vm startup time in the cloud." *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012.

31) Hu, Shiliang, and James E. Smith. "Reducing startup time in co-designed virtual machines." In *ACM SIGARCH Computer Architecture News*, vol. 34, no. 2, pp. 277-288. IEEE Computer Society, 2006.

32) Razavi, Kaveh, Liviu Mihai Razorea, and Thilo Kielmann. "Reducing vm startup time and storage costs by vm image content consolidation." In*Euro-Par 2013: Parallel Processing Workshops*, pp. 75-84. Springer Berlin Heidelberg, 2013.

33) Al-Kiswany, Samer, Dinesh Subhraveti, Prasenjit Sarkar, and Matei Ripeanu. "VMFlock: virtual machine co-migration for the cloud." In*Proceedings of the 20th international symposium on High performance distributed computing*, pp. 159-170. ACM, 2011.

34) Open Grid Forum. Open Cloud Computing Interface - Core, 2011.

35) DMTF. Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol, 2013.

36) Petcu, Dana, and Athanasios V. Vasilakos. "Portability in clouds: approaches and research opportunities." *Scalable Computing: Practice and Experience* 15.3 (2014): 251-270.