

# **Design and Implementation of a framework for Multi-Cloud Service Broker**

By

**Md Ahsan Ullah**

**B.Sc. in Computer Networks with Honours  
University of East London in London, United Kingdom, 2008**

A thesis

Presented to Ryerson University

In partial fulfilment of the  
Requirements of the degree of

**Master of Applied Science**

In the program of  
Computer Networks

Toronto, Ontario, Canada, 2015

© Md Ahsan Ullah 2015

## Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis or dissertation to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis or dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.



---

Md Ahsan Ullah

## Acknowledgments

There are a number of people without whom this thesis might not have been completed and to whom I would like to give my sincere thanks.

First, I would like to express my deepest gratitude to my supervisor Prof. Muhammad Jaseemuddin who gave me the opportunity to pursue my MASC and for his constructive guidance throughout this work, which showed me how research can be.

I am particularly grateful to my research colleague Hager Ghouma for her patience and for her support towards my research work, which helped to improve the quality of this work. She has provided me with constructive feedback during this period and given me the support needed to pursue my research.

Furthermore, I would like to thank the Yates Graduates School and Programme Director Dr. Ngok Bobby Ma for funding my very productive research at the Ryerson University, which gave me the opportunity to collaborate with the nice people from the computer networks group members especially Dr. Lisa Li and Dr. Abdul Hafeez.

Finally, I would like to thank my parents and my wife for their continuous encouragement.

## Abstract

Cloud service broker (CSB) as an emerging technology intermediates heterogeneous multiple cloud services for both the providers and consumers. Recently, Cloud computing & mobile cloud computing applications (MCA) have gained an enormous popularity, which has led to an increasing need for the development of platform independent Middleware/CSB to support all types of cloud service consumer applications including x86\*x64 based standard OS & ARM based mobile applications, web browsers, etc. Developing Platform Independent Hybrid CSB, however, is not an easy task. Developers have to deal with difficulties inherent from the different cloud controllers, cloud service providers environments, clients' application types, network connection types (wired, wireless), GPS (Global Positioning Systems) information of cloud resources and clients' etc.

In this thesis, the proposed design of a middleware/CSB that abstracts the real-time resources of various clouds (private, public, home, Local) and stores the resources in its own Database. It will also store clients requests then analyzes the request to find the nearest available servers which is running the appropriate applications. Then the CSB will forward the destination servers information to the clients. Thesis goal is to achieve context awareness, location awareness, platform independence, portability, efficiency, and usability. Portability is achieved by following the J2ME platform specifications. The middleware has been implemented and tested on a real time Openstack cloud using by our newly designed Android Clients and platform independent Mozilla Firefox browser. The performance measurements of the middleware show that it achieves its efficiency requirements. Furthermore, the middleware's database can be used for resource algorithm, pattern analysis, and for future requirements.

# Table of Contents

<b>ABSTRACT</b>	<b>4</b>
<b>1. INTRODUCTION</b>	<b>9</b>
1.1. PROBLEM STATEMENT	10
1.1.1. Automatic Service-selection issues (decision-making)	11
1.1.2. Interoperability Issues (Vendor Lock-in)	11
1.1.3. Heterogeneity Issues (Inter-Cloud)	11
1.1.4. Cloud awareness issues (Context & Location)	12
1.2. RESEARCH QUESTIONS	12
1.3. THESIS ORGANIZATION	13
<b>2. BACKGROUND</b>	<b>15</b>
2.1. CLOUD SERVICE BROKER (CSB)	16
2.1.1. What is CSB?	16
2.1.2. Roles of CSB: Service Management	17
2.1.2.1. Cloud Service Intermediation/Integration	17
2.1.2.2. Cloud Service Aggregation	17
2.1.2.3. Cloud Service Arbitrage	18
2.1.3. The need of Cloud Service Broker	18
2.1.4. Related work (CSB)	19
2.1.4.1. Jamcracker CSB Platform	19
2.1.4.2. Forrester's CSB Business Model	20
2.2. CLOUD PROVIDER	21
2.2.1. Service Deployment	21
2.2.1.1. Private Cloud:	22
2.2.1.2. Public Cloud	22
2.2.1.3. Community Cloud	23
2.2.1.4. Hybrid Cloud	23
2.2.2. Service Orchestration	23
2.2.2.1. Infrastructure as a Service (IaaS)	23
2.2.2.2. Platform as a Service (PaaS)	24
2.2.2.3. Software as a Service (SaaS)	24
2.2.3. Cloud Service Management	25
<b>3. DESIGN ASSUMPTIONS OF CLOUD SERVICE BROKER</b>	<b>27</b>
3.1. CLOUD SERVICE BROKER WORKFLOW DESIGN	29
3.1.1. CSB – Client Controller	31
Spring Framework:	32
Why Spring Framework?	32
3.1.2. CSB – Cloud Connector	33
Openstack4j	35
Why Openstack4j?	37
3.1.3. CSB Message Diagram	37
3.1.4. CSB Algorithm (Context & Location Aware)	39
3.2. CLOUD PROVIDER PLATFORM	41

3.2.1. <i>Openstack</i>	41
<i>Why Openstack?</i>	43
3.3. CSB – CLIENT PLATFORM	43
3.3.1. <i>Android Client</i>	43
<b>4.1. PLATFORM INDEPENDENT CLOUD-SERVICE-BROKER</b>	<b>46</b>
4.1.1. <i>CSB Java-PROJECT</i>	46
Default package	47
Client	48
Controller	48
Resources	48
Service	49
4.1.2. <i>CSB Database</i>	51
4.2. CSB CLIENT IMPLEMENTATION	53
4.2.1. <i>Central module</i>	53
4.3.2. <i>Connection-Handler Module</i>	54
4.3.3. <i>App-WebView-Client Module</i>	55
5.1. CSB OPERATIONS	57
5.2. ANALYSIS OF CSB’S HANDLING CLIENT REQUEST	58
5.2.1. <i>x86-64 based Standard Browsers requests Results:</i>	58
5.2.1. <i>Mobile Device’s Browsers Request Results:</i>	60
5.2.2. <i>Androids Clients-APK Requests Results:</i>	60
5.2.3. <i>Evaluation of CSB Client handling results</i>	61
5.3. CSB PERFORMANCE ANALYSIS	62
5.4. CSB -CLOUD INTEGRATION PERFORMANCE ANALYSIS	63
<b>6. CONCLUSION</b>	<b>68</b>
6.1. SUMMARY	68
6.2. FUTURE WORKS	70
<b>APPENDIX – A</b>	<b>71</b>
1.1. OPENSTACK MULTI-NODE CLOUD CONFIGURATION	71
1.2.1. <i>Openstack Installation</i>	72
1.2.3. <i>OpenStack Configuration</i>	75
1.2.3.1. <i>Openstack Instance’s web-server Configuration</i>	78
<b>APPENDIX - B</b>	<b>80</b>
LIST OF TABLE	80
LIST OF ACRONYMS	80
<b>APPENDIX – C</b>	<b>82</b>
BIBLIOGRAPHY	82

# List of Figure

Figure 1.1: High Level System Architecture	9
Figure 2.1: NIST Cloud Computing Reference Architecture [19]	15
Figure 2.2: Relationship between Consumer, Broker & Provider	16
Figure 2.3: Possible CSB Platforms and its Services	18
Figure 2.4: Jamcracker CSB Platform [25]	20
Figure 2.5: Future CSB Business Model of Forrester [26]	21
Figure 2.6: Cloud Deployment type	22
Figure 3.1: Conceptual Design of Cloud Services Broker	28
Figure 3.2: Design of Hybrid CSB Workflow	29
Figure 3.3: CSB - Client Controller and related components	31
Figure 3.4: Spring Framework Architecture [37]	32
Figure 3.5: CSB-Cloud Connector and related component	34
Figure 3.6: CSB- Cloud Connectors Cloud-APIs'	36
Figure 3.7: CSB -Message Diagram	38
Figure 3.8: Openstack Multi-Node Architecture Design	42
Figure 3.9: Architectural Design of CSB Client Android Application	44
Figure 4.1: Default Package of CSB Java Project	47
Figure 4.2: CSB Client Information Storing Queries	48
Figure 4.3: CSB server-count to connect to multiple clouds	49
Figure 4.4: CSB Automation Services	49
Figure 4.5: CSB Cloud VMs' location acquiring process	50
Figure 4.6: CSB Algorithm to identify nearest cloud VMs'	50
Figure 4.7: CSB clients stored information (including IPs', location, etc)	51
Figure 4.8: CSB servers stored information (including IPs', location, etc)	52
Figure 4.9: CSB clients' location acquiring process from the host devices	54
Figure 4.10: CSB clients' request parameter building process	54
Figure 4.11: CSB clients' JSON REST & HTTP handling process	55
Figure 5.1: Mobile Browser requests results from CSB	60
Figure 5.2: Android application's request results from CSB	61
Figure 5.3: CSB Client's Request Results Comparison	61
Figure 5.4: CSB Enterprise Integration Tools Performances	63
Figure 5.5: CSB Cloud Integration Tools Performances	64
Figure 5.6: CSB Automated Cloud Resource Provisioning	66

# Chapter 1: Introduction



# 1. Introduction

With the evolution of cloud computing, Cloud providers typically use a "pay as you go" model, this can lead to unexpectedly high charges if cloud consumers and enterprise administrators do not adapt to the cloud pricing model [1]. The lack of common Cloud standards and "vendor-lock-in" issues obstruct the interoperability across Cloud providers [2]. For this reason, cloud users have to manually choose cloud provider to meet their functional and non-functional service requirements while keeping the payment low. It is hard for the cloud users to collect and maintain all the needed information from current commercial Clouds to make accurate decisions [3].

According to Forrester Research, "developers are bypassing IT and putting applications onto public clouds at a rate five times greater than IT thinks" [4]. Enterprises need to hook up [users, systems, databases, applications, and web services] with more than a dozen different cloud services providers. To meet the business demand, enterprises are looking to hook into a Cloud Service Brokers that would serve as mediators and also would provide customization, integration, security, and aggregation services [4].

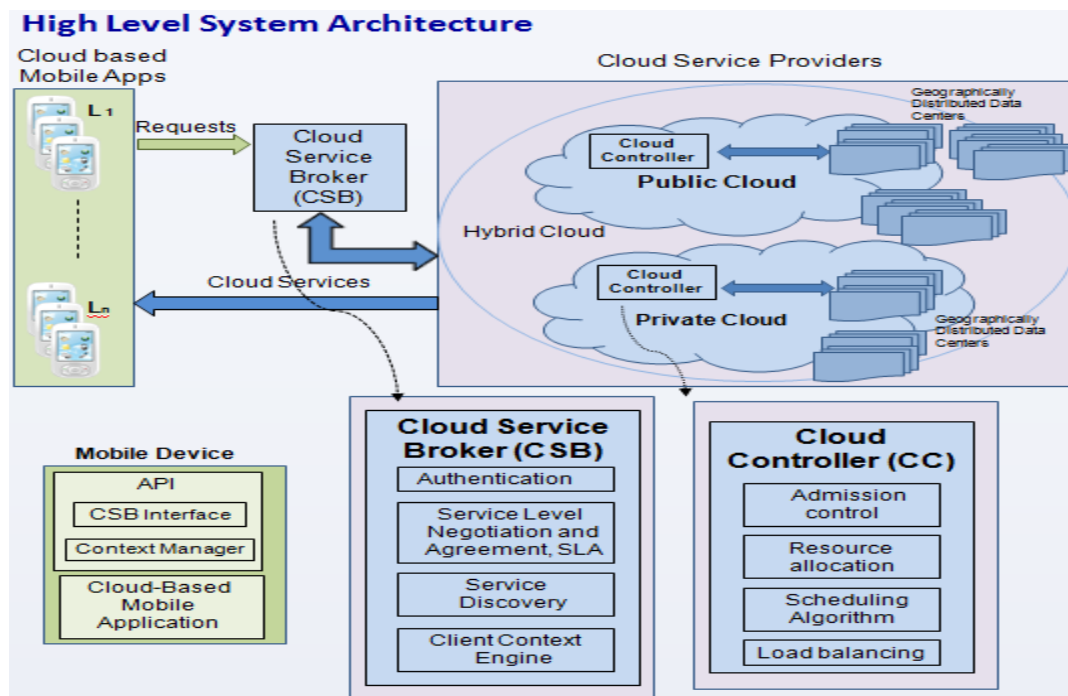


Figure 1.1: High Level System Architecture

The need of a universal service broker has become essential to facilitate cloud users' ability to find the most suitable cloud services by taking their functional and non-functional SLA requirements into account. What enterprise need is a multi-cloud service broker framework to act as a mediator between consumers and multiple cloud providers to automate the service selection and deployment. The framework requires components for decision-making, monitoring, SLA management and interoperability layer to interact with heterogeneous Clouds [4].

## **1.1. Problem Statement**

The Cloud customers are facing a difficult problem of finding the appropriate Cloud that fit business needs. As a result, uniform interfaces & intermediary services are needed to prevent monopolies of single Cloud providers. The potential use cases of the Inter-cloud vision are defined by the Global Inter-Cloud Technology Forum (GICTF) that the possibility of market transactions via brokers [5]. The independent broker entity needs to act as a mediator between the Cloud consumer and multiple Cloud providers to support consumers in selecting the provider that better meets consumer's requirements. By using uniform interface, broker service needs to easily deploy and manage user service regardless of the selected provider. Former research director at Gartner, Frank Kenney [6], explained the need for Cloud brokers as:

“The future of Cloud computing will be permeated with the notion of brokers negotiating relationships between providers of Cloud services and the service customers”

Currently, researchers trying to resolve many technical challenges in regards to the design and use of heterogeneous cloud brokers. In this thesis we identified the major problems that need to be addressed [6].

### **1.1.1. Automatic Service-selection issues (decision-making)**

Presently cloud consumers are bound to manually make decisions about which Cloud to choose to keep the payment low. For example of the Amazon web services (AWS), mobile cloud users of an enterprises are redirected to the amazons specific cloud servers (e.g. AWS data centre in Asia, Europe, America, etc) which are far away from the users mobile users, which result delay in services, mobile users end up paying more for mobile data, enterprise needs to pay more to Amazon for longer cloud service uses and lastly Amazon pays more for power consumption. Where as if the cloud broker holds real-time multi-cloud-providers resource repositories, then broker would be able to instantly analyse cloud consumer's request upon receipt and process requests based on both users and providers geographical location, required service type, application server type, etc [7].

### **1.1.2. Interoperability Issues (Vendor Lock-in)**

All the existing commercial brokerage solutions [8], support specific Cloud platforms and offer limited broker functionality. Most of the proposed architectures are still visionary [9], and have only prototypical or partial implementations [10]. From the Gartner research (Frank Kennedy) we realize that cloud service broker technology is the key to enabling the interoperability across Clouds provider and to permit their orchestration through centralized broker entities [6].

### **1.1.3. Heterogeneity Issues (Inter-Cloud)**

The most important feature of the inter-Cloud vision is that consumers can benefit services from multiple providers to run complex applications [11]. Unfortunately, the heterogeneity of the Cloud services differs in their QoS, cost and functionality which makes the manual composition of services a big challenge for users. Researchers [5], focusing on semantic description of a single cloud services which are not suitable enough for describing composite services provisioned by different Cloud providers [12].

Heterogeneous service broker can use the semantic description to automatically build provider & service repositories for the purpose of discovery and selection [13].

#### **1.1.4. Cloud awareness issues (Context & Location)**

The multi-Cloud research is still in its early stage [13, 14] and most of these works concentrate on running workflows on multiple Clouds without providing solutions to manage the Inter-cloud context & location aware data transfer at runtime, which can affect costs and performance [11]. Location awareness is important to support data-intensive workflow applications SLA constraints such as Cloud-to-Cloud latency, client-to-Cloud throughput, etc.

### **1.2. Research Questions**

In regards to design and implement the multi-Cloud service broker, the following research questions have arisen:

- How to design the framework of a generic architecture of a multi-Cloud service broker [16]?

The centralized multi-Cloud broker framework requires interacting on behalf of the cloud consumers with multiple, interoperable Clouds provider to perform different management and monitoring tasks [12].

- How to implement generic multi-Cloud broker architectures by uniform abstract layer (API) between the broker and the providers [5]?

The broker should be able to communicate cloud providers via its API to retrieve real-time update of various cloud provider resources. The purpose of these periodic updates is to add new service values like matchmaking and data management and automatic decision, service provisioning and execution status.

- How to achieve a composite multi-Cloud service semantically [13]?

To achieve this requirement, the broker should perform an efficient decision making process based on the user request parameter by its Cloud monitoring databases [17]. The collected Cloud providers' monitoring data and service descriptions need to be stored in an abstract manner within data repositories. Semantic techniques should be easily adaptable and extendable describe providers service offerings, QoS metrics, prices as well as the complex SLA requirements of composite services [18].

### **1.3 Thesis Organization**

We present in Chapter 3 the design of a Cloud Service Broker (CSB) framework to address the above need and provide the above requirements for multi-cloud environment, which are further discussed in the context of current research in CSB in Chapter 2. We further discuss the implementation of our CSB using Openstackj cloud API in Chapter 4. We demonstrate the proof of concept working of the CSB through our own Android application that we develop using Spring-framework. We chose the more difficult path of developing and implementing the CSB in a test-bed instead of simulating it within CloudSim because it allows our design to integrate the core CSB module with Open Stack cloud for multi-cloud solution, and with other components such as MySQL database and interworking with Android client. It also provides us vehicle to experiment with real life scenarios that can later be developed into a complete multi-cloud solution. The CSB includes components for enterprise integration tools, queue handler, query analyzer, decision maker, cloud integration tools to monitor and discover clouds and live repositories of cloud resources. We validate the functionalities of our CSB and evaluate its performance in Chapter 5. Finally, we present our conclusion and future work in Chapter 6.

# Chapter 2:

# Background

## 2. Background

As time passes, the computing technology advances immensely. The National Institute of Standards and Technology (NIST) [19] defines cloud computing as “Cloud computing is a pay-per-use model for enabling ubiquitous, convenient, On-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”

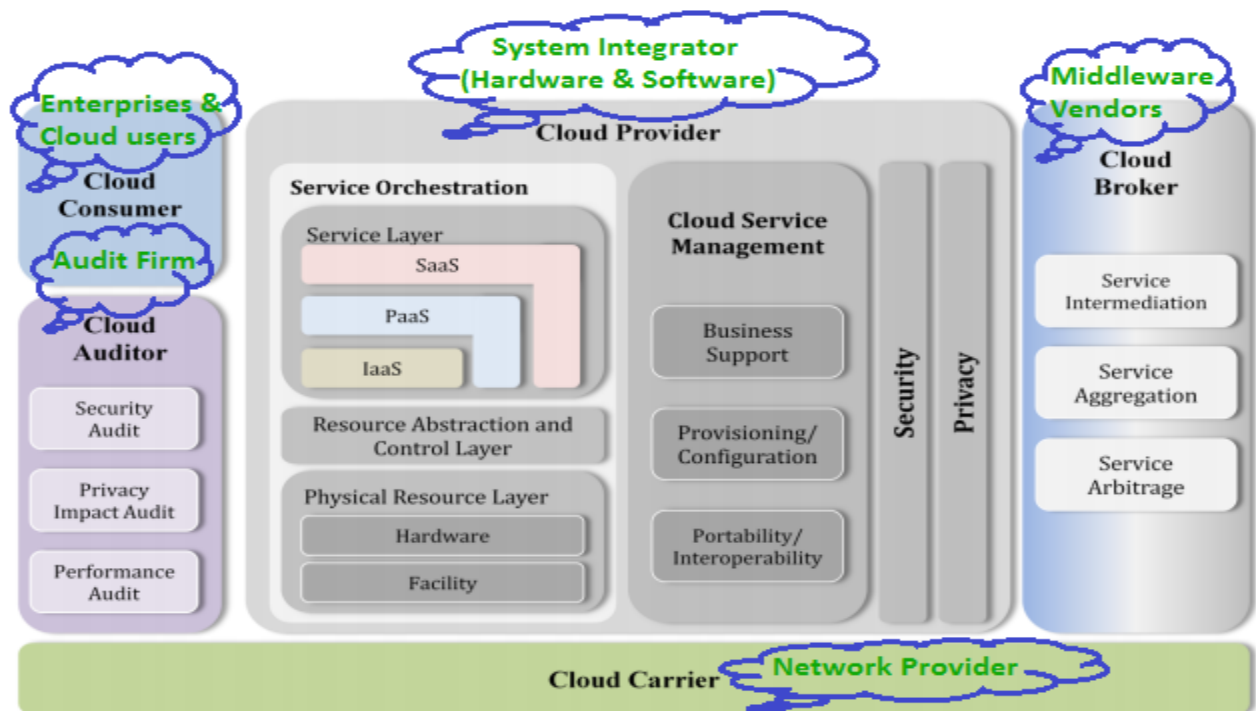


Figure 2.1: NIST Cloud Computing Reference Architecture [19]

NIST cloud reference diagram above categorizes the major cloud computing actors, their relationships, functions & activities. We modified the diagram to illustrate a high-level architecture to understand all the requirements, uses, characteristics of cloud computing. The actors involved in cloud ecosystem are cloud consumer, cloud provider, cloud auditor, cloud broker and cloud carrier [19]. Among them, cloud broker manages the delivery of cloud services and also maintains relationship between cloud providers & cloud consumers.

## 2.1. Cloud Service Broker (CSB)

### 2.1.1. What is CSB?

Gartner Group defines a CSB is “an IT role and business model in which a company or other entity adds value to one or more (public or private) cloud services on behalf of one or more consumers of that service [6]”.

Brokers might be middleware, platforms or suites of technologies that enhance the base services available through the cloud. It can manage access to inter-cloud services, provide greater security and even create completely new services [22].

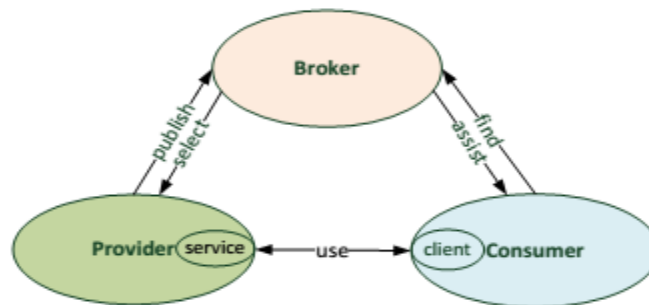


Figure 2.2: Relationship between Consumer, Broker & Provider

Major CSB characteristics are as follows:

- It enables Cloud Service Providers (CSPs) to register available their Infrastructure as a Service (IaaS) into the CSB's service register catalogue [6]
- Then it enables Cloud Service Consumers (CSCs) to find the needed infrastructure services from the CSB's catalogue, if a CSCs cannot search the service satisfying their utilization intent, then CSC can request to find candidate CSPs suitable to their needs by specifying service agreement to want to use
- It deals with CSC's service agreement that requires ability of CSPs by negotiating and it also offers and manages the most suitable IaaS product [6].



### **2.1.2. Roles of CSB: Service Management**

By using CSB, enterprises can add value to cloud services on behalf the users of that service via three primary roles including integration, aggregation and customization brokerage [23].

#### **2.1.2.1. Cloud Service Intermediation/Integration**

It directly enhances a given service by adding value to enhance some specific capability. It could also supervise pricing and billing, access management, etc. This type of brokerages will exist in three places [24].

- It may reside in the cloud at the provider's location and may allow let the provider to deliver a level of governance beyond the original service.
- It may reside at the enterprises location and may allow local management or administration of service levels.
- It may reside in the commercial CSB vendors' location as a service independent of the original service provider or the consumer location [24].

#### **2.1.2.2. Cloud Service Aggregation**

It combines multiple services into one or more new services. It will ensure that data is modelled across all component services and integrated as well as ensuring the movement and security of data between the service consumer and multiple providers. These aggregation brokers will exist primarily in the cloud as service providers in their own right, forming a layer of service provisions that approximates the application layer in traditional computing [23]. In aggregation-style brokerages, the services brokered are generally fixed and won't change frequently.

### 2.1.2.3. Cloud Service Arbitrage

It provides flexibility and opportunistic choices for the service aggregator and foster competition between Clouds e.g. providing a credit-scoring service that checks multiple scoring agencies and selects the best score [23].

### 2.1.3. The need of Cloud Service Broker

CSBs are one of the most needed and attainable opportunities for cloud service providers. What sits between service consumer & cloud provider has become a critical success factor as cloud services multiply and expand faster than the ability of cloud consumers to manage or govern them in use. CSBs will increase the ability of consumers to use cloud services in a trustworthy manner. Cloud providers must begin to partner with cloud brokerages to ensure that they can deliver the services they promote [23].

The diagram below shows how CSB operators can provide many services between multiple Cloud Service Providers (CSPs) and Cloud Service Consumers (CSCs) [20, 21]. The role of brokers to add value to services and to deliver new services built and delivered on top of old services.

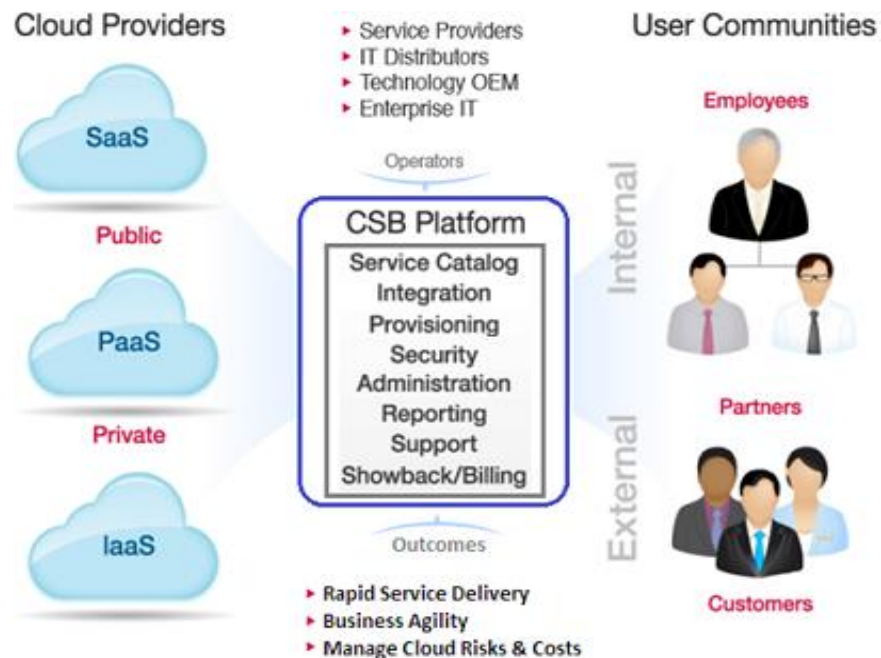


Figure 2.3: Possible CSB Platforms and its Services

CSBs not only offer intermediation, monitoring, transformation/portability, governance, provisioning and integration services but also negotiate relationships between various cloud providers and consumers. Cloud brokerage is needed for the following business demands [20, 21, 23, 24].

- It ensures efficient business operations through standardized lower business operation costs, an agile workforce and improving business productivity.
- It establishes an efficient IT operation that improves standardization and interoperability with easier system and network management
- It improves enterprises ability to address critical issues like security and enables transformation.
- It supports businesses to achieve maximum return on investment in technology whilst reducing the complexity of existing environments.
- Evolution of a capability aligned with the core principles of enterprise architecture.

#### **2.1.4. Related work (CSB)**

##### **2.1.4.1. Jamcracker CSB Platform**

Jamcracker CSB platform includes aggregation, cataloguing, provisioning, access control, security, auditing, monitoring, reporting, metering, billing, administration, and user support. It is based on the Java Enterprise Edition (Java EE) and it includes support for standard Web-services protocols such as SOAP/XML, SAML, WSDL, and DSML. This standards-based platform foundation ensures that cloud providers & enterprises have the freedom to integrate with any other services regardless of technology [25].

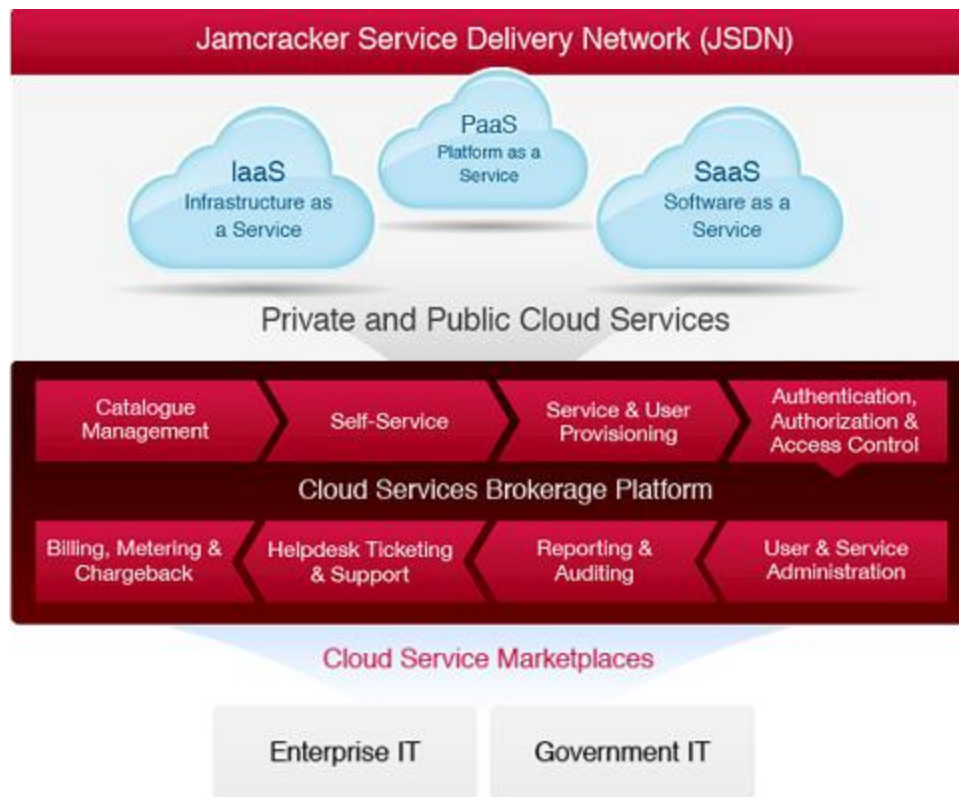


Figure 2.4: Jamcracker CSB Platform [25]

The platform is built on a standards-based architecture that is highly extensible through its data model and APIs. Major design principles include:

- Standards-based architecture
- Extensible data model
- Configurable workflow engine
- Published APIs for service and enterprise integration
- Federated user and security policy management

#### 2.1.4.2. Forrester's CSB Business Model

Forrester believes that by using existing simple cloud broker model enterprises can access to infrastructure as a service on demand. But in future, it expects a 'full broker' that will automatically span all clouds and acts as a "fixer" for companies seeking to deploy new apps or to handle busy times of year. The brokers do not just do this for

compute resources, but also will do it for consumer, e.g. automatic decision making, workload management, dynamic sourcing, elasticity management, etc [26].

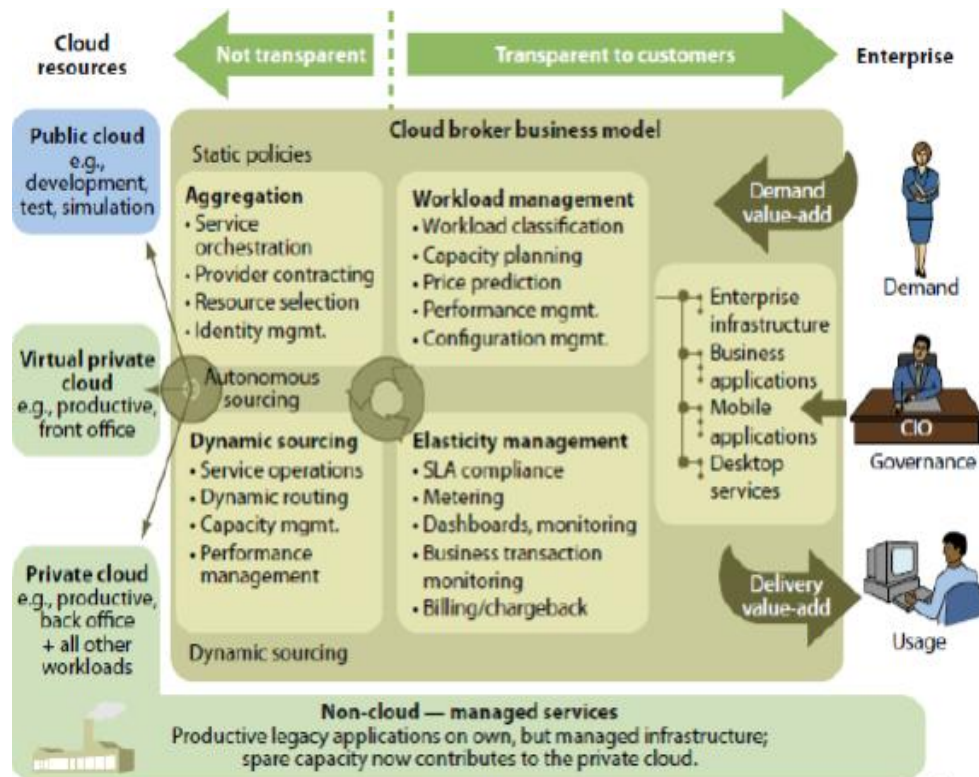


Figure 2.5: Future CSB Business Model of Forrester [26]

## 2.2 Cloud Provider

Cloud Provider runs the cloud software's to integrate their data centres under one cloud controller then deliver cloud services to Cloud Consumers through Internet. Generally, providers are equipped with the following main characteristics [27]:

### 2.2.1. Service Deployment

NIST categorizes four possible Cloud deployment models are public cloud, private cloud, community cloud and hybrid cloud. Cloud providers need to design their infrastructure to operate in one of the above deployment models [24].

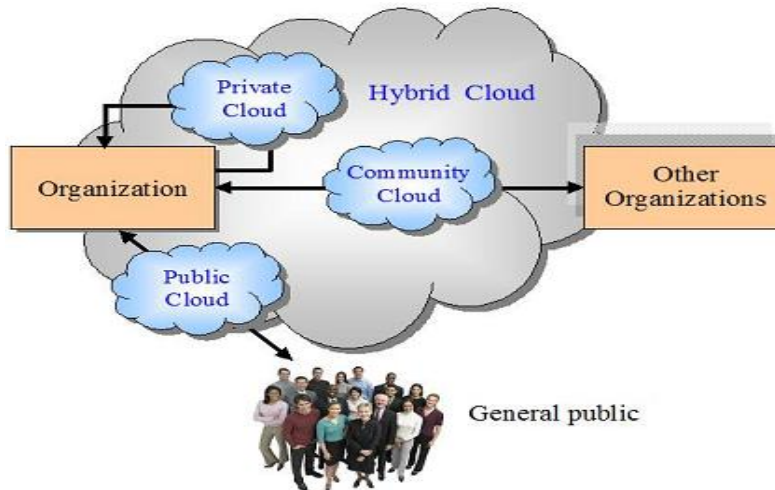


Figure 2.6: Cloud Deployment type

#### 2.2.1.1. Private Cloud:

Providers' services are restricted to the businesses that own and manage the infrastructure [24].

- Leverages existing CapEx (capital expenditure)
- Intended for a single Tenant
- Can help reduce OpEx (operational expenditure)

#### 2.2.1.2. Public Cloud

The provider's cloud resources should be available to the general public via internet.

- Provides on-demand services over public network to the general public.
- Supports multiple tenants [24]
- Offers pay-as-you-go (utility billing) model.
- Enable enterprises to Shifts from CapEx to OpEx
- Example of public cloud: In 2006, Amazon published its Elastic Compute Services (EC2) [28] & it's Simple Storage System (S3) [29] to allow users to rent a server and store data on Amazon's hosted computing & storage infrastructures.

### **2.2.1.3. Community Cloud**

Providers' infrastructure are shared by a group of organizations with common concerns

- Supports resource portability
- Reduce costs by allowing organizations to share CapEx & OpEx among each others [24].
- Bring together groups of organizations with common goal/interest.

### **2.2.1.4. Hybrid Cloud**

In this infrastructure, services are owned and managed by two or more clouds provider (private, public or community) [24].

- Bridges on or more private, public or community clouds
- Supports resources portability
- Allows manipulation of CapEx & OpEx to minimise costs

## **2.2.2. Service Orchestration**

It stitches cloud providers software & hardware components together to connect and automate workflows to deliver a defined Service. Three main service layer of Orchestration are explained below [24]:

### **2.2.2.1. Infrastructure as a Service (IaaS)**

It is a self-service model to access, monitor and manage cloud infrastructures such as compute (VM), storage, networking services (router, firewalls), etc. enterprises can install any required platform on top of cloud IAAS layer to establish their own computing environments [30].

Requirement of IAAS services:

- If enterprises need dynamic scaling, utility pricing model and variable cost
- For small businesses with no capital to invest in expensive hardware
- If organization is growing rapidly and scaling hardware would be difficult
- If business demand is very volatile and need temporary infrastructure [24]

#### **2.2.2.2. Platform as a Service (PaaS)**

It allows enterprises to develop, test and deploy their applications quickly, easily and without the complexity of buying and maintaining the software and infrastructure underneath it [24].

Requirement of PAAS services:

- To provide customers an entire hosting environment to develop their applications
- To provide Multi-tenant architecture for multiple concurrent users to utilize the same development application
- To provide built in scalability of deployed applications such as failover & load balancing [30]
- To Integrate with web services and databases via common standards

#### **2.2.2.3. Software as a Service (SaaS)**

It eliminates the need to install and run applications on individual computers. It enables enterprises to streamline their support & maintenance because everything can be managed remotely e.g. applications, runtime, data, virtualization, etc [31].

Requirement of SAAS services:

- to deal with applications from a central location
- to distribute software in a “one-to-many” model



- cloud consumers not required to handle software upgrades and patches
- Its API's allow for integration between different pieces of software [24]

### **2.2.3. Cloud Service Management**

It includes all types of service related functions that are crucial for the operations & management of those services required by cloud consumers such as:

- On-demand access- automatic ubiquitous access over the Internet to the resources [32].
- Resource elasticity- capability to scale resources up and out as required.
- Resource pooling- multi-tenant access to shared resources.
- Pay-per-use- requested resources are charged only when used [33].

# Chapter 3:

## Design Assumptions of Cloud-Service-Broker

In this chapter we present the design of our cloud service broker that maintains its own database of multiple cloud providers' resources and provides binding for location & context aware services. The following functions of the broker are explained in this chapter:

- Design of CSB
- CSB-Workflow
- CSB Components (Client Controller, Cloud Connector, Database Manager, Query Analyzer, Decision Engine)
- CSB Message Diagram
- CSB Algorithm

### 3. Design Assumptions of Cloud Service Broker

The lack of interoperability across cloud providers, their costs and SLAs hindered consumers' ability to deploy their services on multiple Clouds [34]. In this chapter, we address research questions as specified in Chapter-1 then present a broker-based framework to assist consumers selecting and deploying their services on multi-Cloud environments with respect to their needs in terms of SLA, location and context awareness.

Current frameworks for deploying multi-cloud services are still either in the implementation phase or do not provide all the desired functionalities needed by users [35]. In this section, we propose a generic multi-Cloud service broker framework developed to facilitate cloud services deployment on multi-Cloud environments. The three-tier architecture of the proposed framework, depicted in the following Figure 3.1, is composed of the users, Cloud service broker and the Cloud providers.

The CSB as shown below in the middle tier serves as a mediator between Cloud users and the providers. The CSB forms the core of our designed framework by offering new, value-added services to consumers' applications. Its main task is to find a suitable, nearest target provider cloud to run user requested applications on multi-Cloud. Furthermore, its design allows the deployment and monitoring of services on top of heterogeneous Cloud providers. The internal components of every tier are discussed in detail in the following subsections [36].

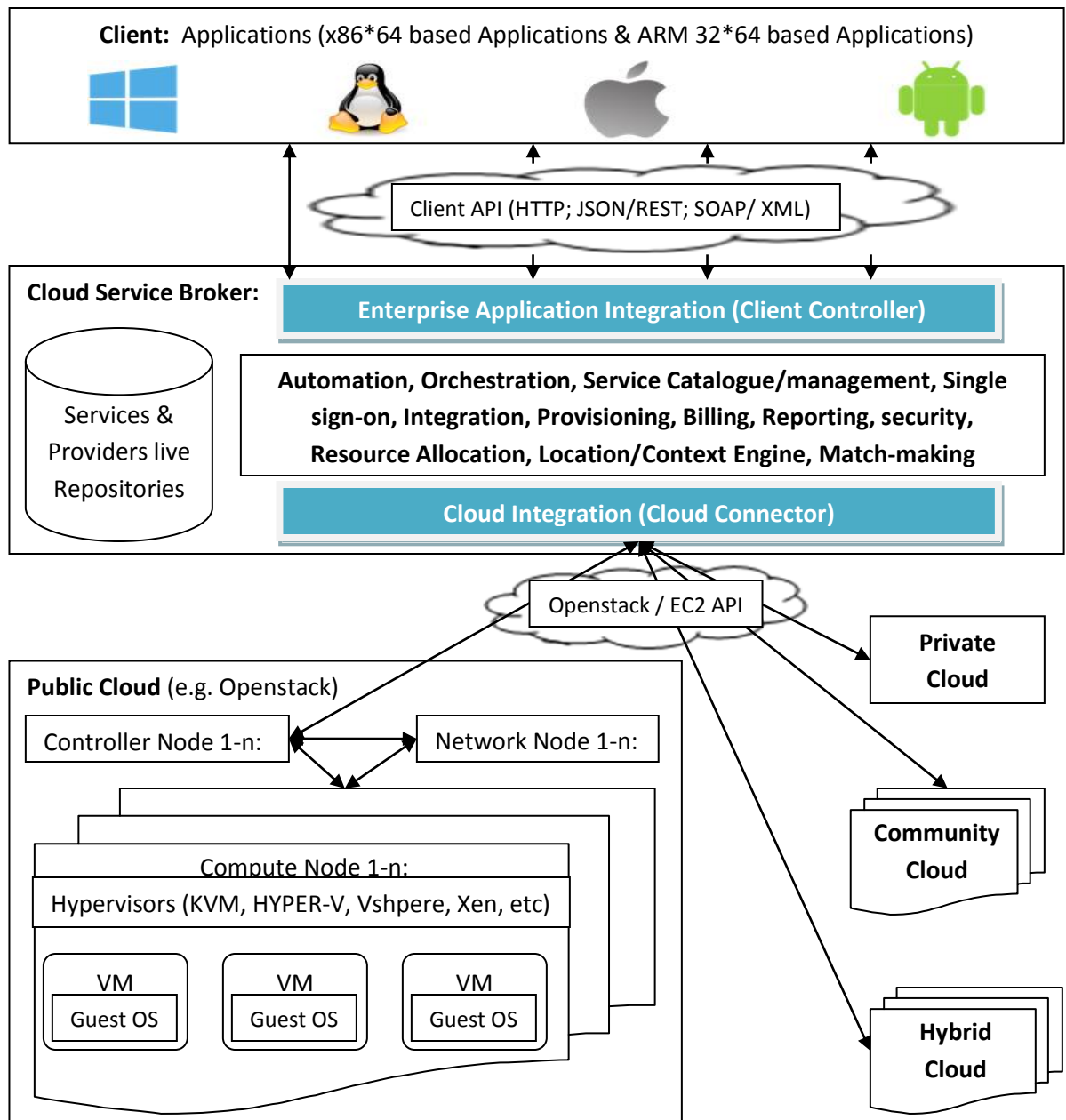


Figure 3.1: Conceptual Design of Cloud Services Broker

The 'Cloud service broker', 'Cloud providers' and 'Users' roles are briefly described in the subsection 3.1, 3.2 and 3.3.

### 3.1. Cloud Service Broker Workflow Design

We describe the CSB components through cloud workflow shown in Figure 3.2.

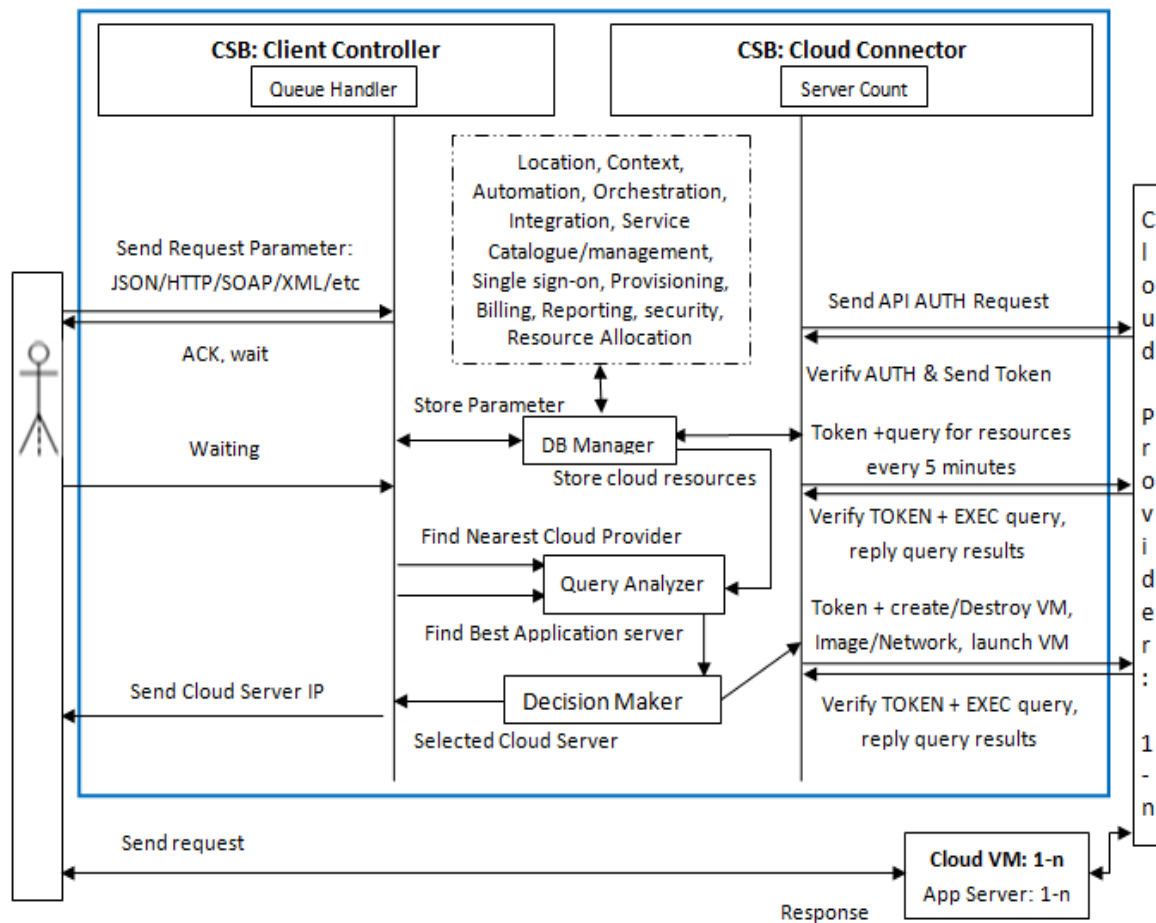


Figure 3.2: Design of Hybrid CSB Workflow

The main components of the broker are explained below:

- **CSB-Client-Controller:**

It is one of the main components that initially accepts all clients' requests then asks other CSB components to process the request in the order of their calls, and then finally responds the clients with results.

- **CSB-Cloud-Connector:**

The other main component of the broker is 'CSB-Cloud-Connector' that not only maintains the connection with multiple cloud providers and updates broker database every 5 minutes with cloud resource information but also able to execute resource provisioning tasks in the remote clouds as per broker needs.

- **Queue handler:**

It adds each client's request in its request-pool and also asks database manager to store the request's parameter into the client table of the broker database named '*csb-db*'.

- **Query Analyser:**

It deals with client's request parameter such as context (application type, priority, connection type, etc) and location (both clients and servers latitude and longitude). It accesses '*csb-db*' through DB-manager and compares cloud providers resources to find all the candidate servers.

- **Decision Engine:**

It performs a matching process where the functional and non-functional SLA requirements of users are compared to the monitored SLA metrics as well as the capacity load of the Clouds. It is able to use resource allocation algorithms to make a trade-off between the cost and SLA characteristics of the selected Clouds. It controls negotiation and provisioning such as high-level management (e.g. create, start, stop, suspend VM, image, volume, network) to meet peak hours service demands.

- **Database Manager:**

The DB manager and the database can reside in different machine as the 'DB manager' acts as a SQL client and connects to the database server through TCP/IP connection. It manages (stores, extracts and updates) client request parameter and Inter-cloud resources at runtime [36].

In the following subsections we discuss each CSB component in detail.

### 3.1.1. CSB – Client Controller

The client controller is the interface to clients that accepts their requests. It then forwards each request to Queue handler to put them in the ‘request pool’. It retrieves requests from Queue handler in FIFO order and dispatches them to the query analyzer for further processing. The Query analyzer asks db manager to list all cloud servers that match with the service type of client required service type (e.g. App1-HTTP, App2-FTP, App3-SMTP, etc). It reviews the server-list to form the list of active servers list. Then query analyzer further analyzes the active-server-list to find the nearest cloud server. Finally Query analyzer sends the analyzed data to the Decision Engine, then Decision engine decides which cloud server address to select for each clients request and sends the results to the client controller. Finally, the client controllers sends the cloud server IP address to the client.

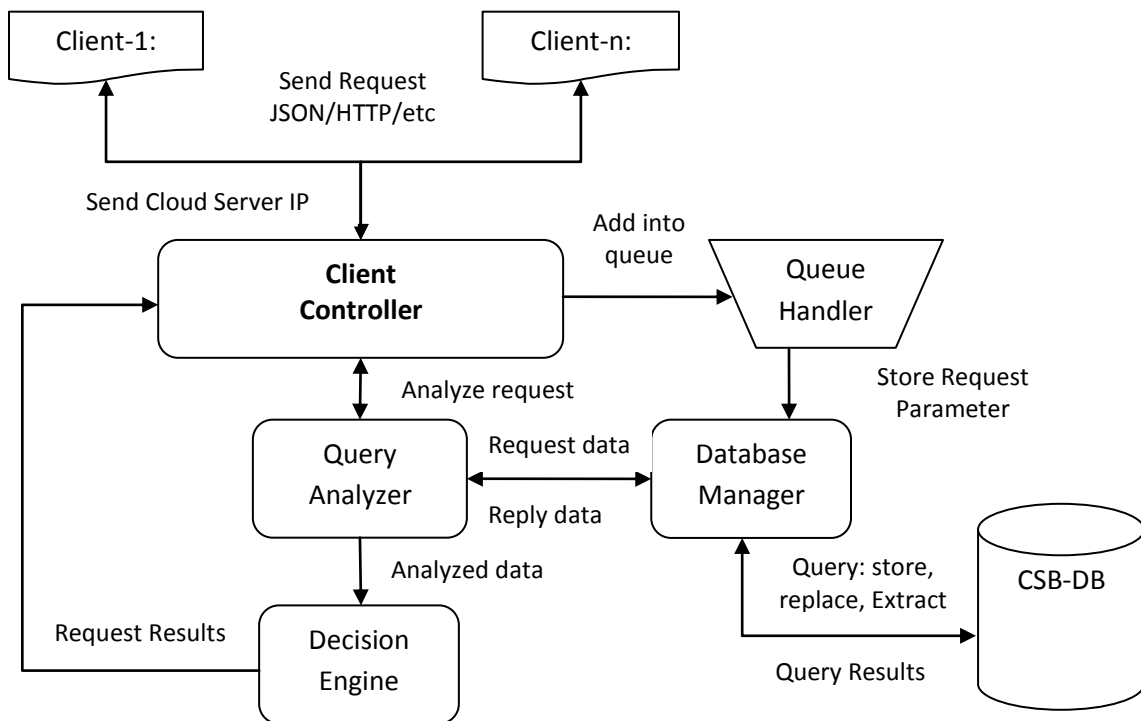


Figure 3.3: CSB - Client Controller and related components

We have chosen Spring Framework as Enterprise Application Integrator to handle requests from many commonly used types of client requests by using JSON REST, SOAP XML and HTTP API. The Spring-Framework modules are integrated with our client controller to process different types of client applications and request parameters.

### Spring Framework:

It is a popular open source Java platform to facilitate infrastructure support to develop high performing, easily testable, reusable Java applications. It handles the infrastructure so developers can focus on application. As shown in the figure, its features organized into about 20 modules. These Modules are grouped into Data Access/Integration, Web, Aspect Oriented Programming (AOP), Core Container, Instrumentation, Messaging, and Test [37].

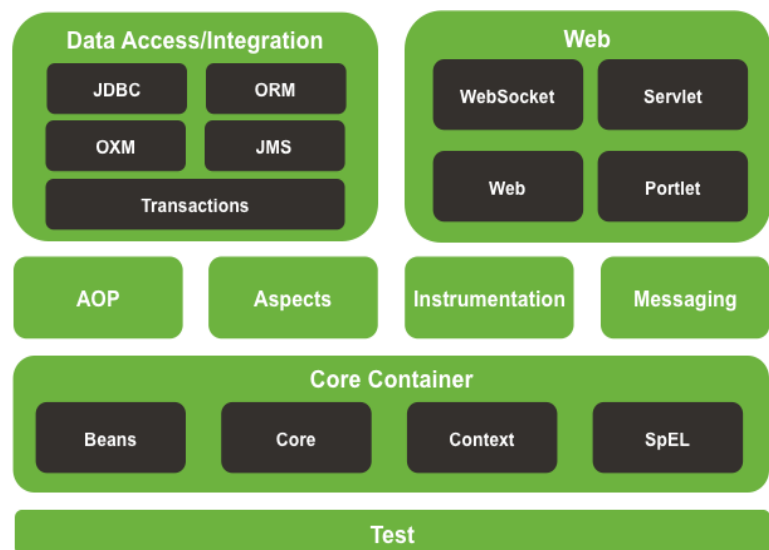


Figure 3.4: Spring Framework Architecture [37]

### Why Spring Framework?

It is lightweight when it comes to size and transparency. The basic version of spring framework is around 2MB. Following is the list of few important reasons of using Spring Framework [37]:



- It enables developers to create applications from plain old Java objects (POJOs). The benefit of using POJOs is that we do not need an EJB container (e.g. application server) but we have the option of using servlet container (e.g. Tomcat or others) [37].
- It is organized in a modular fashion so we have to worry only about packages/classes we need and ignore the rest.
- It truly makes use of the existing technologies such as several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, other view technologies.
- It offers a handy API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO) into reliable, unchecked exceptions.
- Its dependency injection helps in gluing classes together and same time keeping them independent.

### **3.1.2. CSB – Cloud Connector**

The main purpose of the Cloud Connector is to be able to execute resource provisioning tasks on behalf of cloud controller that eliminates the need of manual resource provisioning on multiple cloud. The cloud connector is capable of to remotely creating/launching/stopping/removing remotely any cloud server. Then cloud services will be easily accessible and affordable for cloud service consumers.

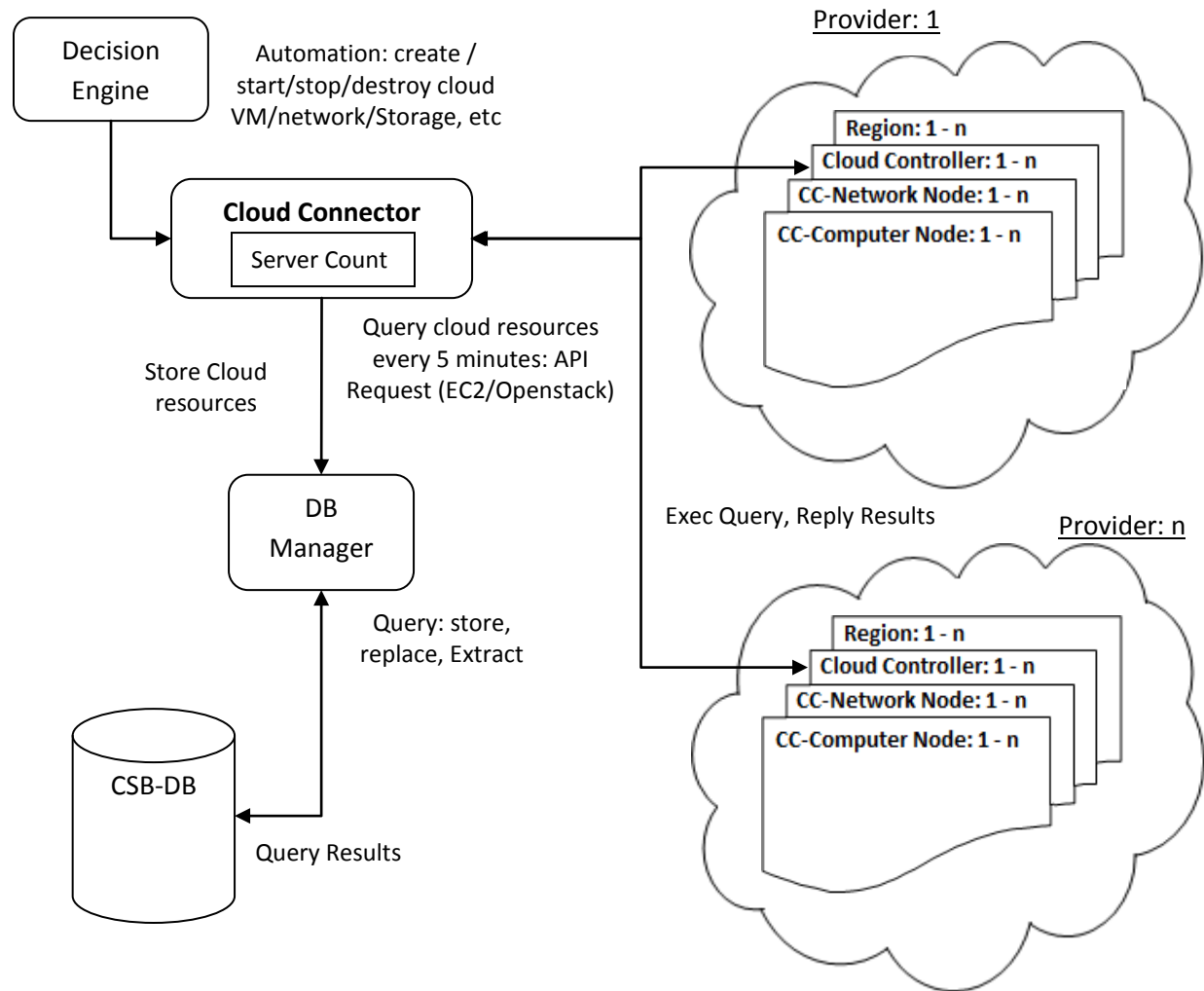


Figure 3.5: CSB-Cloud Connector and related component

- Our Cloud connector runs daemon services every 5 minutes to build and maintain live repositories from multiple cloud controllers. To connect to multiple cloud controllers it maintains server count feature for each cloud's API authentication calls. It asks Database manager to store cloud-resources-information after each periodic update it receives via API calls.
- The Connector is able to execute tasks in the cloud controller via API calls whenever required by the decision engine. For example, if the decision engine decides to use cloud servers that are paused/off then then the connector can

- immediately start the cloud server by API calls without any authentication delay or human interactions.
- The step-by-step API calls between our connector and cloud controllers are shown in the message diagram in Figure 3.7

We have chosen Openstack4j and AWS java to implement our cloud connector to connect, extract cloud resources and store in database. The Openstack4j supports most of the cloud API such as EC2 and Openstack. It even has separate API for cloud compute, Identity, image, Network, Volume, Object store, telemetry and Orchestration, etc. The reason of choosing Openstack4j is described below [34].

## **Openstack4j**

It is an open source library that helps manage most popular cloud deployment (e.g. OpenStack and AWS EC2). It allows provisioning and full control of OpenStack system. Its library offers following API as shown in Figure 3.6 [38]:

- Identity (Keystone): This API provides the central directory of users, tenants, and service endpoints and also responsible for authenticating and providing access to all the other OpenStack services. It enables administrators to configure centralized policies, users and tenants [38].
- Compute (Nova): It provides management to Servers (running virtual machines), VM Management, Flavors and diagnostics, etc.

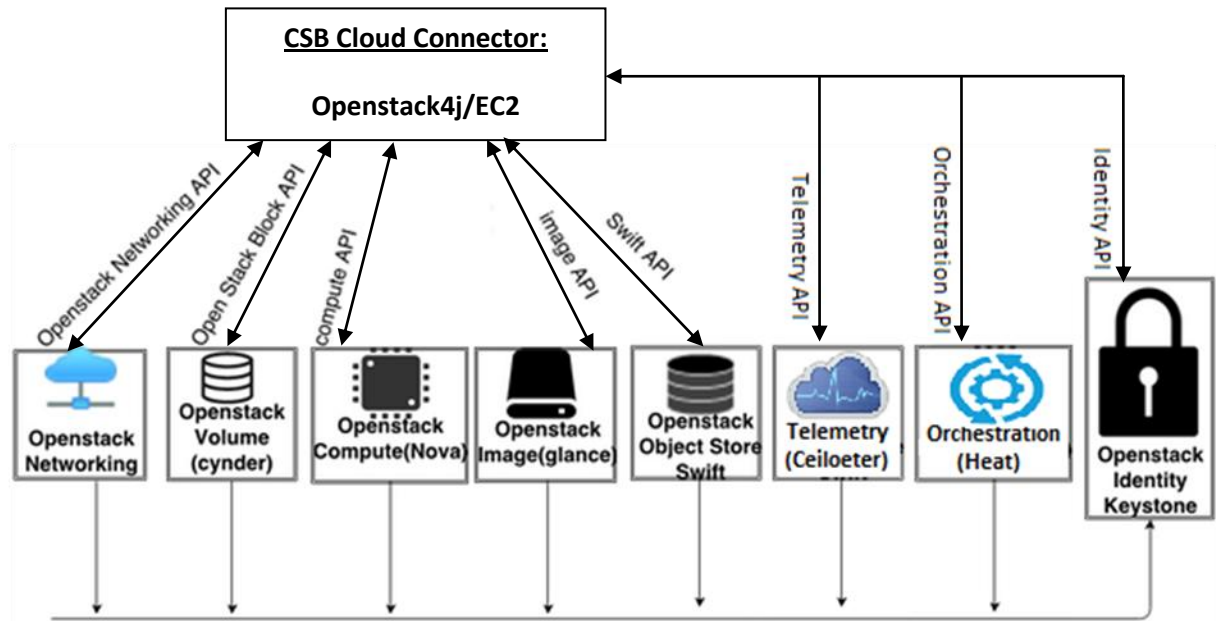


Figure 3.6: CSB- Cloud Connectors Cloud-APIs'

- Image (Glance): It provides discovery, registration and delivery services for cloud images.
- Network (Neutron): It provides network connectivity between interface devices managed by OpenStack service such as Nova.
- Block Storage (Cinder): It mounts drives to scale storage. Object Storage
- Object Storage (Swift): It provides object storage for files and media which can be shared globally or kept private for adhoc storage.
- Telemetry (Ceilometers): It delivers metering and statistic measurements against OpenStack core components. It is very useful for customer billing, account and reporting of resources [38].
- Orchestration (Heat): It can control Stacks, Templates, Resources and Events.

### *Why Openstack4j?*

OpenStack4j has made it easy to manage large cloud system by providing a simplistic API and intelligent error handling. The reasons of choosing openstack4j are explained below [38]:

- Fluent Interface: All API calls are fluent by nature as they are object oriented, easy to use and easy to read.
- Concrete API: APIs are interface defined includes corresponding models and builders. API Implementations are defined within an 'internal' package.
- Deployment Tested: APIs have been tested and are used in various cloud environments.
- Exception Handling: Exceptions shows exact reason for failure allowing cloud application to report appropriately [38].

### **3.1.3. CSB Message Diagram**

In this subsection we have constructed a detailed Message diagram of the overall process of the cloud service broker. The diagram below shows that initially cloud connector acts as a client to cloud controller, then authenticates via either EC2 or Openstack API calls. . After the initial authentication it sends request to each cloud controller to build its database. It continuously monitors the changes of the resource information based on the periodic updates it receives from provider clouds and it then updates its database through db manager. The cloud connector is incarnated as daemon process so that it performs its functions independent to the controllers that process client requests simultaneously. The client controllers receive simultaneous requests from any client application then process each request in the order they receive via queue handler. The Query analyzer analyzes requests then sends analyzed data to the decision engine. The Decision engine sends results to client controller then

controller replies the result immediately. It also requests the cloud connector if it is needed to execute tasks in remote cloud controllers in provider clouds. .

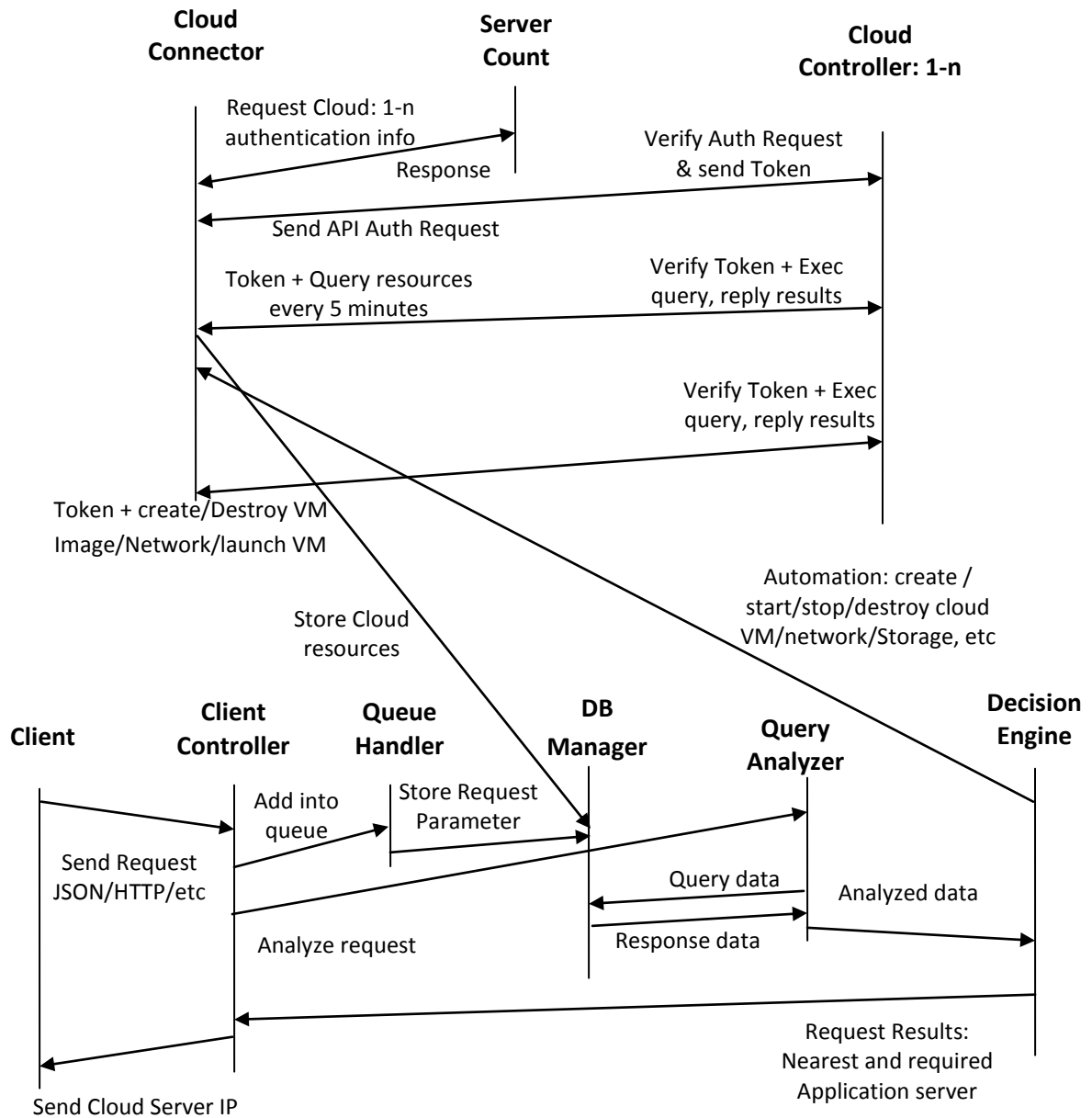


Figure 3.7: CSB -Message Diagram

### 3.1.4. CSB Algorithm (Context & Location Aware)

In this subsection we present a simple context and location aware cloud selection algorithm that is executed in the decision engine. More sophisticated algorithms can be designed and implemented using our proposed CSB framework. The idea of presenting simple algorithm to show the proof of concept of our CSB. The main focus of our thesis to design the CSB, but to test the CSB we need to design our own context & location aware algorithm.

```
// CSB Algorithm (Context & Location aware)
Input: (Request      aReq) //Context of Clients request parameter
{
aReq.Loc ();    //location          loc (latitude, Longitude)
aReq.Prt ();    //Priority           Prt (0-normal, 1-high)
aReq.ST ();     //Service Type      ST (App1-HTTP, App2-Ftp, App3-SMTP)
}
Output: Nearest Cloud Server = calc loc[active server{query DB(cloud-server, Client)}]
// Initialize All CSB-Server and CSB-Client request variable
// calculate ExecTime [|T| * | aReq |] //Queue Handlers calculation time of each request
in pool
// for each clients request list all the server matches the service type from 'csb-db'
SS-List = QueryDB.GetServerList {Cloud.ST() == aReq.ST ()}
//find all the active server from the server-list (SS-List)
AS-List = SS-List.GetActiveServerList (vm-stat == Active)
//find the nearest server from the Active-server-list (AS-List)
NS = AS-List.GetNearestServer {Cloud.SLoc(), aReq.Loc()}
//find all the active server from the server-list (SS-List)
//AS-List = SS-List.GetActiveServerList (vm-stat == Active)
```

<pre> A<sup>S-List</sup>=Null; S= S<sup>S-List</sup>; For (int i=0; i&lt;n; i++) {     If (isActive (s[i]))         A<sup>S-List</sup>.add (s[i]); }  Return A<sup>S-List</sup>; </pre>
<pre> //find the nearest server from the Active-server-list (A<sup>S-List</sup>) //N<sup>S</sup> = A<sup>S-List</sup>.GetNearestServer {Cloud.S<sup>Loc</sup>(), a<sup>Req</sup>.Loc()} N<sup>S</sup> = Null; Distance = ∞; S =A<sup>S-List</sup>.Loc; C =a<sup>Req</sup>.Loc; For (int i=0; i&lt;n; i++) {     newDistance = Distance (S[i], C);     If (Distance &gt; newDistance)     {         Distance = newDistance;         N<sup>S</sup>. add (S[i]);     } }  Return N<sup>S</sup>; </pre>

**Table 3.1: CSB Algorithm (Context & Location Aware)**



The above algorithm takes few inputs from client request parameter, then analyze those inputs by comparing with cloud resources information from our CSB live database named csb-db, and finally makes the cloud (server) selection.

- for each clients request list all the server matches the service type from 'csb-db'
- find all the active server from the server-list ( $S^{S-List_t}$ )
- find the nearest server from the Active-server-list ( $A^{S-List_t}$ )

## 3.2. Cloud Provider Platform

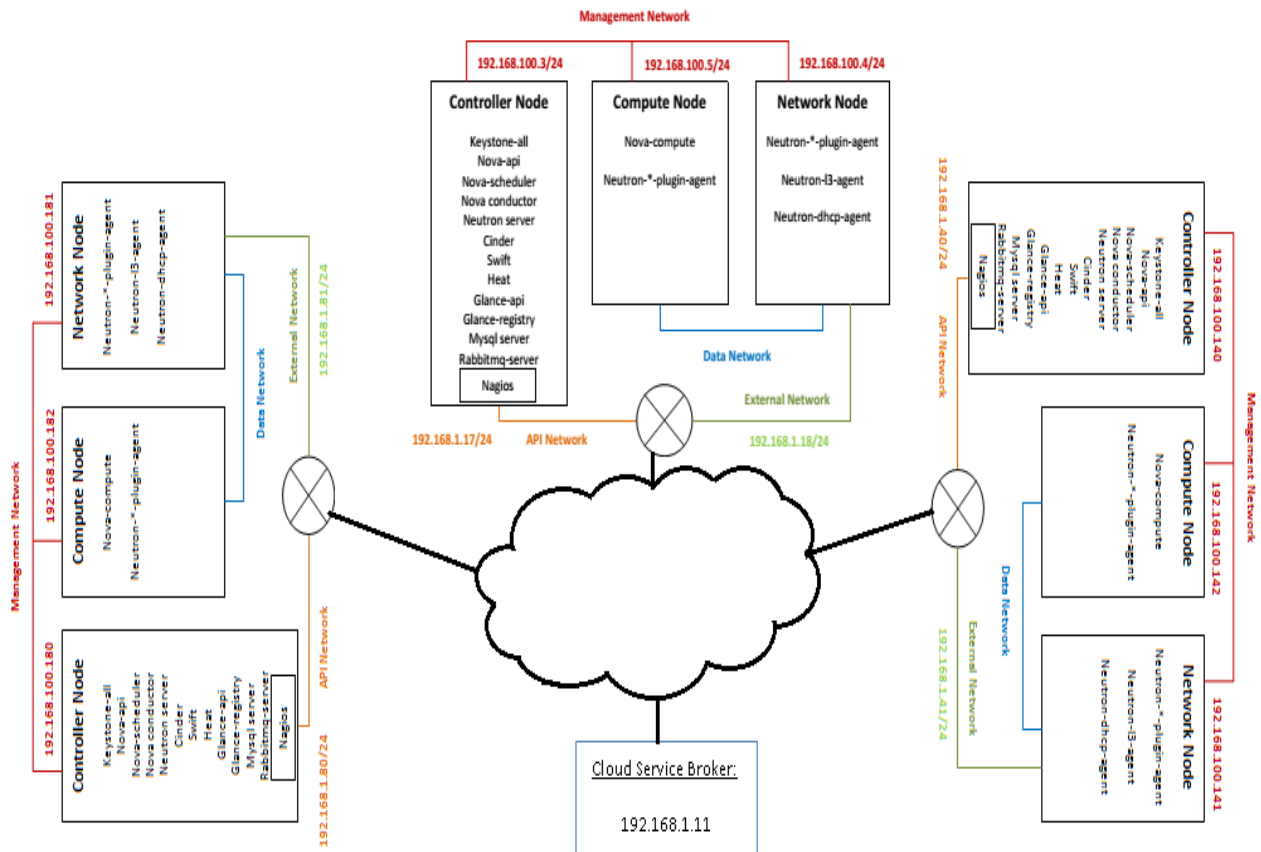
### 3.2.1. Openstack

To test our CSB in real cloud systems we have chosen OpenStack cloud. OpenStack is the most popular open source complete cloud computing platform that is currently supported by more than 150 tech companies worldwide. It allows users to manage cloud resources through web-based dashboard, command-line tools, and RESTful API [40].

There are some basic requirements we need to meet to deploy OpenStack. Here are the requisites, drawn from the OpenStack manual [39].

- Hardware: we are using IBM rack server x3650 with 16GB RAM. We will create 4 VM and will name them as Controller Node, Compute Note and Network node and CSB Node. All nodes have 2 CPUs, 4GB of RAM and a 60GB root volume, with the exception of the compute nodes that have 6GB of RAM so they can better run virtual guests [39].

- Operating system (OS): Although OpenStack supports many operating systems, we have chosen open-source enterprise class OS named CentOS for our multimode cloud setup.



**Figure 3.8: Openstack Multi-Node Architecture Design**

The figure above shows our Openstack multi-node cloud design with 1 controller node, We created 1 network node and 1 compute node for each provider cloud. We have created management network for all three nodes and external network for controller and network node, and the compute node and network node with data network. In chapter 4 we will explain the functions of each node and network [40].

## Why Openstack?

OpenStack supports a variety of virtualization technologies and hypervisors (e.g. KVM, Xen, VMware, Hyper-V, QEMU, LXC, etc). In the following table we showed the similarities between free Openstack and paid AWS (Amazon Web Services) cloud services [40].

Openstack	Used for	AWS
Horizon	Dashboard for end users & administrators to access other backend services	AWS Management Web Console
Nova Compute	manages virtualization and takes requests from end user through dashboard or API to form virtual Instances	AWS Elastic Compute
Cinder	virtualizes pools of block storage devices and also make them directly attachable to any virtual instance	EBS(Elastic Block Store)
Glance	Maintains catalogue for images and is kind of a repository for images.	AMI (Amazon Machine Images)
Swift	Allows applications or instances to store & retrieve all types of data that can grow without bound	AWS S3
Keystone	It is responsible for managing authentication services for all components. It also enables administrators to configure centralized policies, users and tenants.	AWS Identity And Access Management(IAM)

**Table 3.2: Comparison of Openstack & Amazon Cloud Services**

## 3.3. CSB – Client Platform

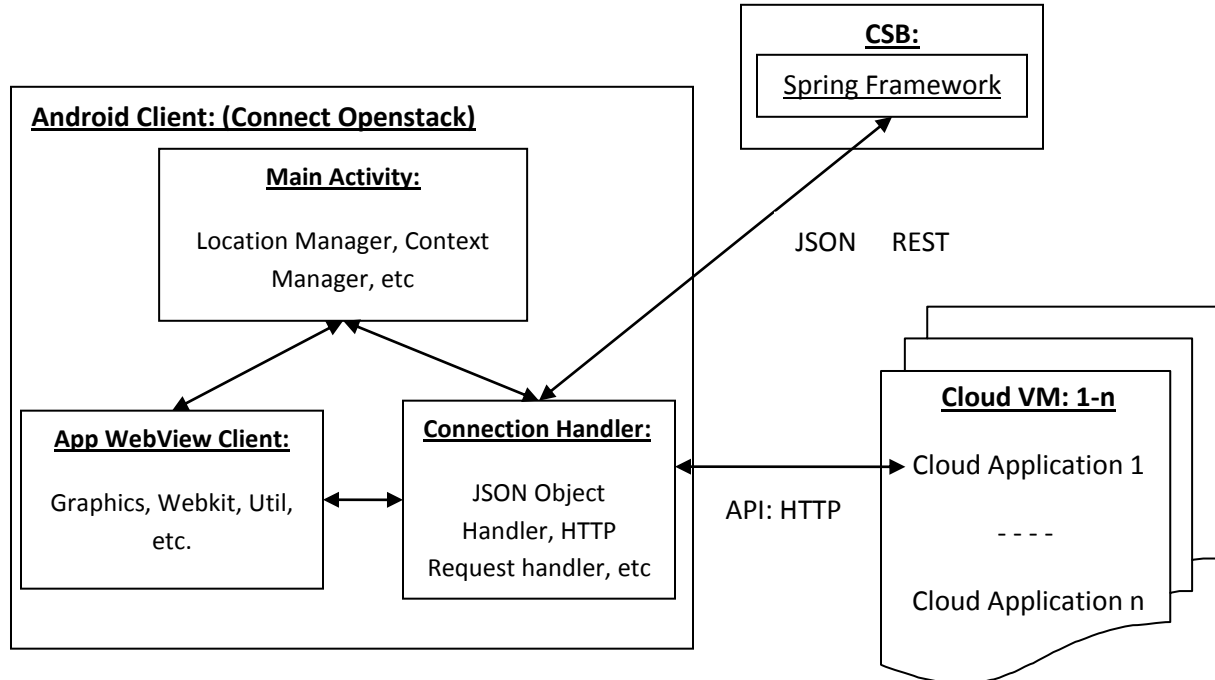
To test the capability of our designed CSB we have designed a simple android client. We chose android as client platform because it is widely supported by tablet, cell phones, and netbook manufacturers. To develop the android client, we have used Android Studio which is the official IDE for Android application development [41].

### 3.3.1. Android Client

The goal of our design is to demonstrate that the client application is able to run from any devices that run Android OS, and the application is lightweight that consumes fewer

resources of mobile devices. The components of android clients are shown in the figure 3.10 and are explained below [41]:

- Central Module: It is the core of the client application as it communicates with built-in GPS module and data connection module (WiFi, 3G/4G). It also builds the request parameter to send to CSB by using connection handler.
- Connection Handler: It has two main functions to meet the demand of the main-activity module. First, it sends request parameter to CSB by using JSON-REST API. When it receives CSB's reply with the destination cloud server IP address, then it forwards the reply to the central module. Second, it sends central module's http request to cloud VM and wait for the reply. Third, it forwards the HTTP reply to App-webview-client module.
- App-webview-client: It parses HTTP responses from the connection handler and represents the information inside the client without any browser.



**Figure 3.9: Architectural Design of CSB Client Android Application**

# Chapter 4:

## Implementation of Cloud-Service-Broker and Android Client

In this chapter we discuss the implementation of our java based CSB that connects to cloud remotely, the deployment of Openstack-based provider cloud and implementation of our Android client to request web services from our CSB.

## 4.1. Platform Independent Cloud-Service-Broker

We chose to implement our CSB in Java to make it platform independent. We used eclipse IDE because it contains a base workspace and an extensible plug-in system to develop Java applications [42]. Our CSB supports mobile http client applications with both custom interface and mobile-browser based interface. We implemented the location and context module using enterprise integration tool (Spring Framework) and cloud integration tool (Openstack4j).

### 4.1.1. CSB Java-PROJECT

First we created a java project in eclipse and modified pom.xml to add Spring MVC framework, and many dependencies such as maven, openstack4j, mysql, JSON, etc. Table 4.1 shows major components of our pom.xml that downloads all the libraries from internet required by the dependencies shown below.

<pre>&lt;groupId&gt;org.springframework&lt;/groupId&gt; &lt;artifactId&gt;gs-rest-service&lt;/artifactId&gt; &lt;version&gt;0.1.0&lt;/version&gt;</pre>	<pre>&lt;dependencies&gt;   &lt;dependency&gt;</pre>
<pre>&lt;parent&gt;  &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;   &lt;artifactId&gt;spring-boot-starter- parent&lt;/artifactId&gt;   &lt;version&gt;1.1.10.RELEASE&lt;/version&gt; &lt;/parent&gt;</pre>	<pre>&lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;   &lt;artifactId&gt;spring-boot-starter- web&lt;/artifactId&gt; &lt;/dependency&gt; &lt;dependency&gt;   &lt;groupId&gt;org.pacesys&lt;/groupId&gt;   &lt;artifactId&gt;openstack4j&lt;/artifactId&gt;   &lt;version&gt;2.0.1&lt;/version&gt;   &lt;classifier&gt;withdeps&lt;/classifier&gt; &lt;/dependency&gt; &lt;dependency&gt;   &lt;groupId&gt;mysql&lt;/groupId&gt;   &lt;artifactId&gt;mysql-connector- java&lt;/artifactId&gt;   &lt;version&gt;5.1.6&lt;/version&gt; &lt;/dependency&gt; &lt;dependency&gt;</pre>
<pre>&lt;build&gt;   &lt;plugins&gt;   &lt;plugin&gt;  &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;   &lt;artifactId&gt;spring-boot-maven- plugin&lt;/artifactId&gt;   &lt;/plugin&gt; &lt;/plugins&gt; &lt;/build&gt;</pre>	<pre>  &lt;groupId&gt;com.googlecode.json-simple &lt;/groupId&gt;   &lt;artifactId&gt;json-simple&lt;/artifactId&gt;   &lt;version&gt;1.1&lt;/version&gt; &lt;/dependency&gt; &lt;dependency&gt;   &lt;groupId&gt;org.apache.commons&lt;/groupId&gt;</pre>
<pre>&lt;repositories&gt;   &lt;repository&gt;     &lt;id&gt;spring-releases&lt;/id&gt;     &lt;url&gt;https://repo.spring.io/libs- release&lt;/url&gt;   &lt;/repository&gt; &lt;/repositories&gt;</pre>	

<pre> &lt;pluginRepositories&gt;   &lt;pluginRepository&gt;     &lt;id&gt;spring-releases&lt;/id&gt;     &lt;url&gt;https://repo.spring.io/libs-release&lt;/url&gt;   &lt;/pluginRepository&gt; &lt;/pluginRepositories&gt; </pre>	<pre> &lt;artifactId&gt;commons-io&lt;/artifactId&gt; &lt;version&gt;1.3.2&lt;/version&gt; &lt;/dependency&gt; &lt;/dependencies&gt; </pre>
--	---

Table 4.1: POM.XML of CSB Java Project

We organised all the java class in a conventional manner to modify and update for the future purposes. The following figure shows how the java class files are organised.

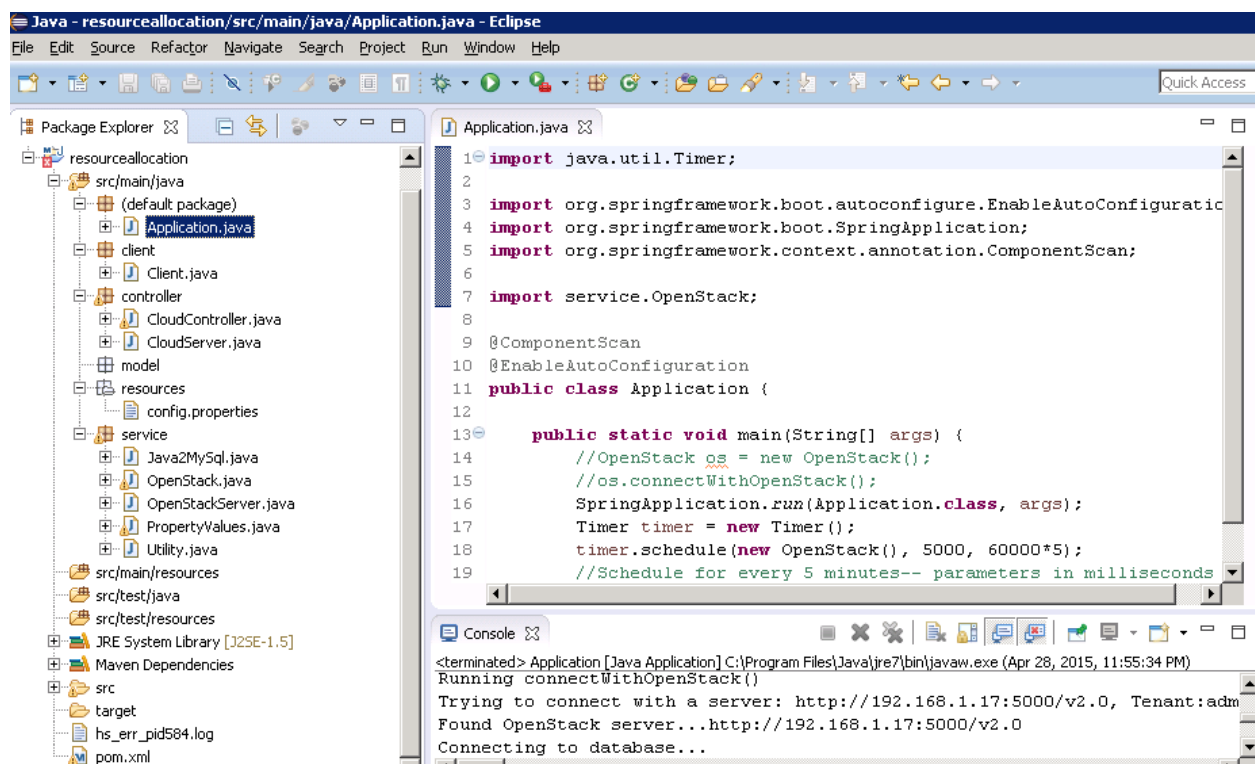


Figure 4.1: Default Package of CSB Java Project

## Default package

- **Application.java:** as shown in figure 4.1, it is the starting point of our CSB project named resource allocation. It starts the spring MVC framework when we run the CSB, which handles all the clients' request. It also runs the ScheduledExecutor every five minutes to extract live cloud resources.

## Client

- **Client.java:** As part of Spring MVC, it stores clients request parameter to csb\_client table in the MySql database. It also communicates with Utility.java to get the analysed results of both clients and cloud providers GPS location. It also determines the appropriate cloud VM that can be assigned to a client request by quering Java2MySql.java. The selection of VM is based on its availability (that is if its status is up) and if it fulfils other client's request parameter (such as application type, priority, etc).

```
String strSql = "REPLACE INTO `csb_client` SET ip_addr = \""+ipAddress+  
    "\", longitude = \""+longitude+ "\", latitude = \""+latitude+  
    "\", priority = \""+priority+ "\", request_type = \""+requestType+  
    "\", connection tvpe = \""+connectionType+ "\", request_time = \""+requestTime+\""";  
db.InsertMySql(strSql);
```

Figure 4.2: CSB Client Information Storing Queries

## Controller

- **CloudController.java:** It controls all kinds of clients API request (JSON-REST, SOAP-XML, HTTP, etc) those looking for cloud services based on their request parameter. It also instructs Client.java how to process each request parameter, deals with concurrent request, queue management, etc.
- **CloudServer.java:** It extracts selected cloud VM's IP by querying Java2MySql.java, as requested by CloudController.java,

## Resources

- **Config.properties:**

It is not a java file, but is used to configure properties for the project. It defines server counts to keep track of the servers that can be connected to multiple cloud providers by using Openstack4j API (EC2, Openstack, etc). It also provides the DNS names of the cloud VMs if needed. In the following snapshot it shows one cloud setup so the server count is set to 1, but we can add more cloud anytime by changing the server count.



```

openstack_servercount = 1
#toronto0 server configuration
toronto0_server=http://192.168.1.17:5000/v2.0
toronto0_user=admin
toronto0_password=PasswOrd
toronto0_tenant=admin

#toronto1 server configuration
#toronto1_server=http://23.91.157.86:5000/v2.0
#toronto1_user=admin
#toronto1_password=PasswOrd
#toronto1_tenant=admin

10.0.0.35=http://ahsan.sytes.net/
192.168.1.202=http://ahsan.sytes.net/

```

Figure 4.3: CSB server-count to connect to multiple clouds

## Service

- **OpenStack.java:**

It authenticates with all types of cloud provider by using OpenstackServer.java, PropertyValues.java, config.properties and openstack4j libraries. It stores all the cloud resources by using Java2MySQL.java. It also executes cloud admin task via Openstack4j API calls. It is capable of provisioning cloud services such as create, destroy, launch, stop, suspend cloud VM's, image, network, volume, flavour, etc by using openstack4j API calls. It also provides automation services by starting the required cloud VM's if found stopped.

```

OSClient os = OSFactory.builder()
    .endpoint(serverName)
    .credentials(user,password)
    .tenantName(tenant)
    .authenticate();
if (os!=null) {
    System.out.println("Found OpenStack server..." +serverName);
    // Find all running Servers
    List<? extends Server> servers = os.compute().servers().list();
    for (Server server : servers) {
        try{
            if(server.getName().equals("ubuntu14") && server.getVmState().equals("stopped"))
                os.compute().servers().action(server.getId(), Action.START);
            storeServerInfo(server);
        }
    }
}

```

Figure 4.4: CSB Automation Services

- **Java2MySQL.java:** It is used by several java file as it provides access to the MySQL database by registering JDBC driver, opening connection, executing query, cleaning up environment then closing the connection. It also runs DB queries on behalf of other components and modules that requested the service and returns the results. The snapshot of query in figure 4.4 shows one such query of Client.java

```
stmt = conn.createStatement();
String strSql = "SELECT sip, sip_dns, sname, sstatus, longitude, latitude FROM `csb-db`.cloud_servers where sstatus=\"ACTIVE\"";
ResultSet rs = stmt.executeQuery(strSql);
double prvDist = 1000000;
while(rs.next())
{
    double currDist = Utility.distFrom(latitude, longitude, rs.getFloat("latitude"), rs.getFloat("longitude"));
    if(prvDist > currDist)
    {
        prvDist = currDist;
        //serverDns = rs.getString("sip_dns");
        serverIP = rs.getString("sip_dns");
        serverName = rs.getString("sname");
        cloudServer = new CloudServer(serverIP, serverName);
    }
}
```

Figure 4.5: CSB Cloud VMs' location acquiring process

- **OpenStackServer.java:** It provides cloud controllers authentication information (IP, username, password, tenant) whenever needed by OpenStack.java.
- **PropertyValues.java:** It assists OpenStack.java to authenticate and to handle resources with multiple cloud providers
- **Utility.java:** It helps CloudController.java and Client.java by comparing the location information of both clients and all candidate servers to identify the nearest cloud VM for every client.

```
public static double distFrom(double lat1, double lng1, double lat2, double lng2) {
    double earthRadius = 3958.75;
    double dLat = Math.toRadians(lat2-lat1);
    double dLng = Math.toRadians(lng2-lng1);
    double sindLat = Math.sin(dLat / 2);
    double sindLng = Math.sin(dLng / 2);
    double a = Math.pow(sindLat, 2) + Math.pow(sindLng, 2)
        * Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2));
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    double dist = earthRadius * c;

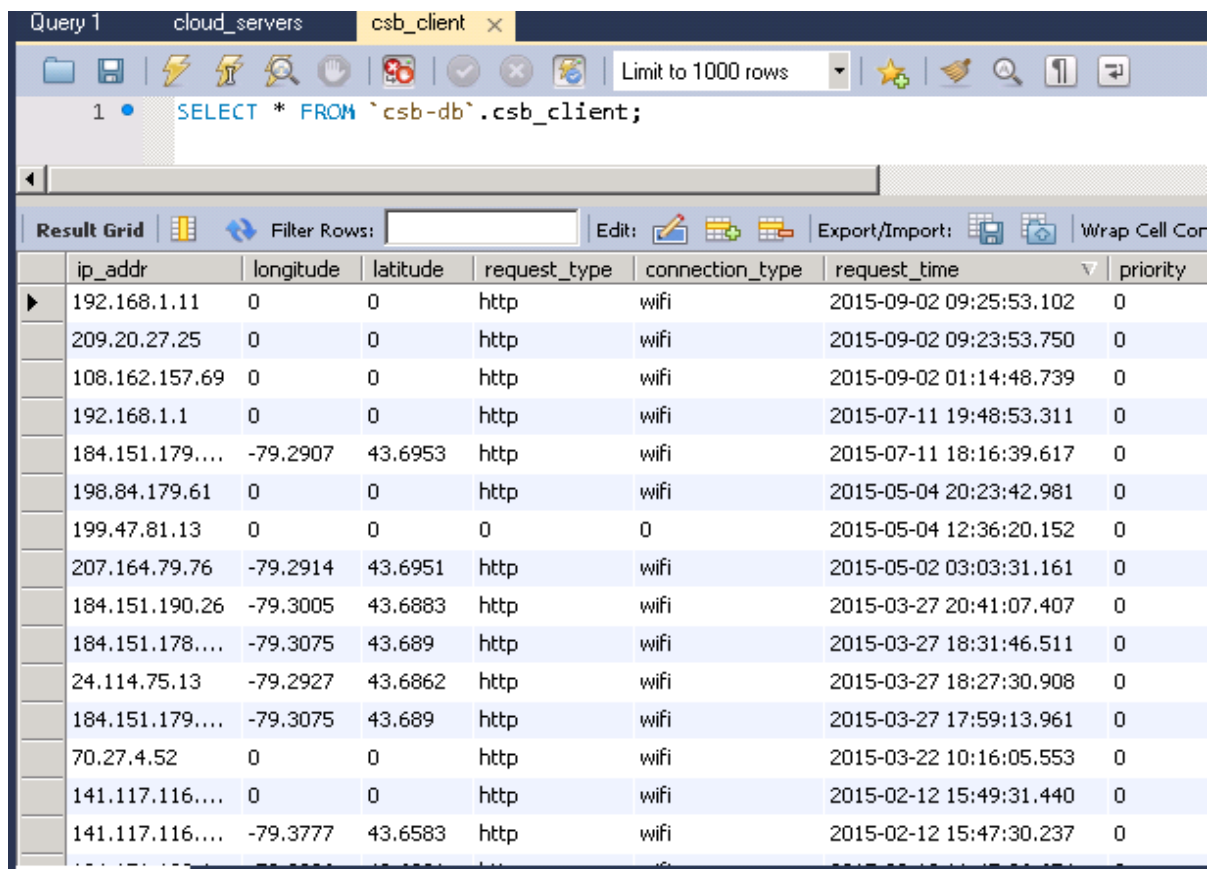
    return dist;
}
```

Figure 4.6: CSB Algorithm to identify nearest cloud VMs'

### 4.1.2. CSB Database

We decided to use MySql version 5.6 and workbench version 6.2 as both can be installed on any platform [43]. We created one database named csb-db that includes following eight tables:

- Csb\_client: It is used to store consumer applications request parameter such as ip\_addr, longitude, latitude, request\_type, connection type, request\_time, priority, etc. This information can be used for resource allocation, pattern analysis and is never deleted from the table. We maintain a single client record that is never duplicated. The CSB checks the clients unique device ID, and If user's location and IP are changed or slew of connection requests arrive at the CSB, then the CSB updates the client record each time it processes a request.

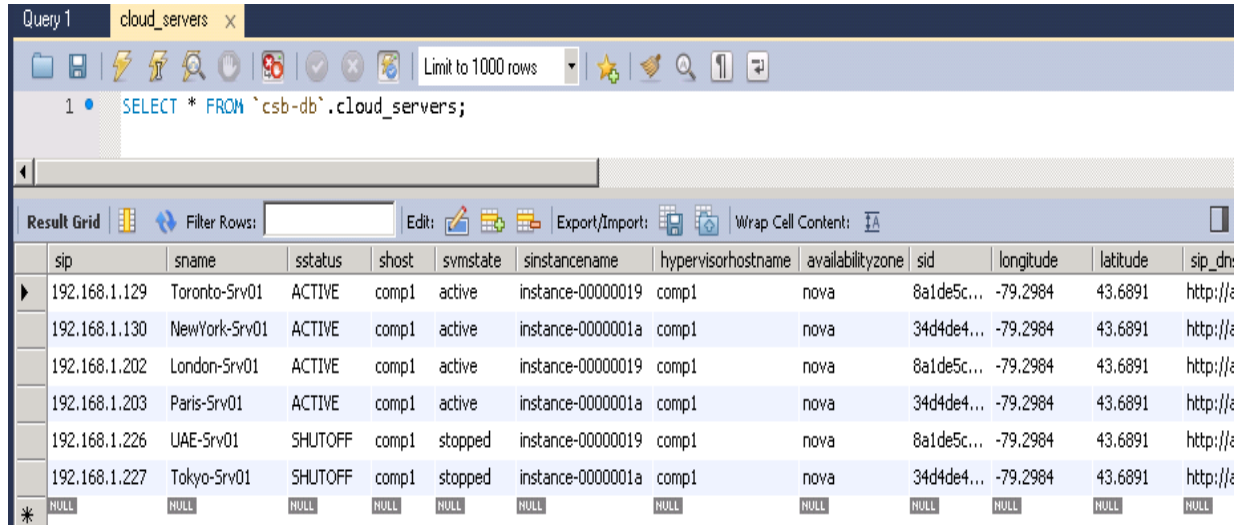


ip_addr	longitude	latitude	request_type	connection_type	request_time	priority
192.168.1.11	0	0	http	wifi	2015-09-02 09:25:53.102	0
209.20.27.25	0	0	http	wifi	2015-09-02 09:23:53.750	0
108.162.157.69	0	0	http	wifi	2015-09-02 01:14:48.739	0
192.168.1.1	0	0	http	wifi	2015-07-11 19:48:53.311	0
184.151.179....	-79.2907	43.6953	http	wifi	2015-07-11 18:16:39.617	0
198.84.179.61	0	0	http	wifi	2015-05-04 20:23:42.981	0
199.47.81.13	0	0	0	0	2015-05-04 12:36:20.152	0
207.164.79.76	-79.2914	43.6951	http	wifi	2015-05-02 03:03:31.161	0
184.151.190.26	-79.3005	43.6883	http	wifi	2015-03-27 20:41:07.407	0
184.151.178....	-79.3075	43.689	http	wifi	2015-03-27 18:31:46.511	0
24.114.75.13	-79.2927	43.6862	http	wifi	2015-03-27 18:27:30.908	0
184.151.179....	-79.3075	43.689	http	wifi	2015-03-27 17:59:13.961	0
70.27.4.52	0	0	http	wifi	2015-03-22 10:16:05.553	0
141.117.116....	0	0	http	wifi	2015-02-12 15:49:31.440	0
141.117.116....	-79.3777	43.6583	http	wifi	2015-02-12 15:47:30.237	0

Figure 4.7: CSB clients stored information (including IPs', location, etc)

The following table's data are updated every 5 minutes by CSB's ScheduledExecutor to find most suitable resources when needed

- Cloud\_servers: to store cloud VM's IP, name, status, host, vm state, instance name, hypervisor host name, availability zone, VM ID, longitude, latitude, DNS name, etc.



The screenshot shows a database query result for the 'cloud\_servers' table. The query is 'SELECT \* FROM `csb-db`.cloud\_servers;'. The result is displayed in a grid with 13 columns: sip, sname, sstatus, shost, svmstate, sinstancename, hypervisorhostname, availabilityzone, sid, longitude, latitude, and sip\_dns. There are 7 rows of data, including a row with NULL values.

sip	sname	sstatus	shost	svmstate	sinstancename	hypervisorhostname	availabilityzone	sid	longitude	latitude	sip_dns
192.168.1.129	Toronto-Srv01	ACTIVE	comp1	active	instance-00000019	comp1	nova	8a1de5c...	-79.2984	43.6891	http://e
192.168.1.130	NewYork-Srv01	ACTIVE	comp1	active	instance-0000001a	comp1	nova	34d4de4...	-79.2984	43.6891	http://e
192.168.1.202	London-Srv01	ACTIVE	comp1	active	instance-00000019	comp1	nova	8a1de5c...	-79.2984	43.6891	http://e
192.168.1.203	Paris-Srv01	ACTIVE	comp1	active	instance-0000001a	comp1	nova	34d4de4...	-79.2984	43.6891	http://e
192.168.1.226	UAE-Srv01	SHUTOFF	comp1	stopped	instance-00000019	comp1	nova	8a1de5c...	-79.2984	43.6891	http://e
192.168.1.227	Tokyo-Srv01	SHUTOFF	comp1	stopped	instance-0000001a	comp1	nova	34d4de4...	-79.2984	43.6891	http://e
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

**Figure 4.8: CSB servers stored information (including IPs', location, etc)**

- Cloud tenants: to store cloud tenants ID, name, descriptions, etc.
- Cloud\_flavors: to store cloud flavors ID, name, vcpu, ram, disk, rxtx factors, etc. This data is used to create new flavors and also to launch new VM's by CSB API calls.
- Cloud\_images: to store cloud images ID, name, status, size, minimum RAM, minimum disk, etc. This data is used to create new image and also to launch new VM's by CSB API calls.
- Cloud telemetry: to store cloud metering data of CPU, memory and network costs, etc.
- Cloud\_key pair: to store cloud key pair name, public key, fingerprint as these are first step to launch an instance for the first time.

- Cloud\_secgrouprule: to store cloud security group ID, name, tenant ID, rules, etc. These rules allow administrators and tenants to specify the type of traffic that is allowed to pass through a port.

## **4.2. CSB client implementation**

We built our CSB client application by using 'Android Studio', as it provides the necessary tools to develop and execute APK files for android devices. We have created java based project in android studio named 'ConnectOpenstack' and divided clients task under the following three modules.

### **4.2.1. Central module**

This module is the core of our client's application. It communicates with android OS modules such as GPS, WiFi, IP address, etc. First, it collects devices location information, followed by unique device ID and IPV4 address, then it builds the request parameter with its own request parameter (e.g. request types, priority, etc). We designed our client for HTTP request, so in the request parameter the request type is http, which is added in the request parameter by this module without asking the user. This module contacts the connection handler to send the request to our CSB which is implemented as a separate VM outside our Openstack cloud.

```

public Location getLocation()
{
    LocationManager lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    Location location = lm.getLastKnownLocation(LocationManager.GPS_PROVIDER);
    if(location == null)
        location = lm.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
    return location;
}
public String getDeviceId()
{
    return Secure.getString(this.getContentResolver(),
        Secure.ANDROID_ID);
}
public String getDeviceIpAddress() {
    try {
        for (Enumeration<NetworkInterface> en = NetworkInterface
            .getNetworkInterfaces(); en.hasMoreElements();) {
            NetworkInterface intf = en.nextElement();
            for (Enumeration<InetAddress> enumIpAddr = intf
                .getInetAddresses(); enumIpAddr.hasMoreElements();) {
                InetAddress inetAddress = enumIpAddr.nextElement();
                // for getting IPV4 format
                String ipv4;
                if (!inetAddress.isLoopbackAddress() && InetAddressUtils.isIPv4Address(ipv4 = inetAddress.getHostAddress()))
                {
                    return ipv4;
                }
            }
        }
    }
}

```

**Figure 4.9: CSB clients' location acquiring process from the host devices**

The following figure shows how the module builds and sends the request parameter to our CSB application.

```

Location location = getLocation();
if(location != null)
{
    longitude = location.getLongitude();
    latitude = location.getLatitude();
}

String openStackServerAddress = getResources().getString(R.string.openStack_url);
String id = getDeviceId();
String reqType = "http";
String connType = "wifi";
String requestParam = "?id="+id +
    "&lon="+longitude+
    "&lat="+latitude+
    "&priority=0"+
    "&reqType="+reqType+
    "&connType="+connType;

Log.e(TAG, openStackServerAddress+requestParam);
serverAddress.execute(new URL(openStackServerAddress+requestParam));

```

**Figure 4.10: CSB clients' request parameter building process**

### 4.3.2. Connection-Handler Module

This module has two main functions to meet the demand of the central module. First, it sends the request parameter to CSB by using JSON-REST API. When it receives CSB's reply with the destination cloud server IP address then it forwards the reply to the

central module. Second, it sends the request from central module to cloud VM and waits for the reply. Third, it forwards the HTTP reply to App-webview-client module.

```
public String getcontent(String url) {
    StringBuilder builder = new StringBuilder();
    HttpClient client = new DefaultHttpClient();
    HttpGet httpGet = new HttpGet(url);
    //httpGet.setHeader("Accept", "application/json");
    try {
        HttpResponse response = client.execute(httpGet);
        StatusLine statusLine = response.getStatusLine();
        int statusCode = statusLine.getStatusCode();
        if (statusCode == 200) {
            HttpEntity entity = response.getEntity();
            InputStream content = entity.getContent();
            BufferedReader reader = new BufferedReader(new InputStreamReader(content));
            String line;
            while ((line = reader.readLine()) != null) {
                builder.append(line);
            }
        }
        String serverAddress = "";
        JSONObject jsonResponse;
        try {
            jsonResponse = new JSONObject(content);
            serverAddress = (String) jsonResponse.get("serverAddress");
        }
        catch (JSONException e) {
            e.printStackTrace();
        }
        MainActivity.loadURL(serverAddress);
    }
}
```

Figure 4.11: CSB clients' JSON REST & HTTP handling process

### 4.3.3. App-WebView-Client Module

This module parses connection-handler HTTP responses and represents the information inside the client without any browser.

# Chapter 5:

## CSB Operations and Performance Analysis



The goal of this thesis is to design and implement a platform independent CSB along with an android client. We also tested our CSB with universal browser Firefox and the result shows that CSB is able to handle any types of clients request simultaneously and response promptly with no observable delay. The CSB implementation can perform its tasks regardless of its location over the public network. It can be implemented inside the cloud systems in enterprises location and even in home network.

In this chapter we discuss the test runs to validate the operation and performance of CSB in the following subsections.

## 5.1. CSB Operations

The CSB is designed such that its modules are independent and execute concurrently as background processes.. The CSB starts with some modules and then continues until it is explicitly stopped. During its lifetime it periodically updates its repository with resource information from multiple cloud providers. The CSB operations in the order of their executions are explained below:

- The CSB's application.java calls the spring framework that runs spring MVC processes to handle clients' requests.
- The application.java starts the timer 'ScheduledExecutor' to run periodic updates every five minutes in a background process.
- The timer calls the Openstack.java to run all the queries.
- The Openstack.java calls the propertyvalues.java and config.properties to run the servercount to authenticate and then send queries to multiple cloud providers.
- To build each queries Openstack.java uses Openstack4j's API, library and connectors as most of the cloud responses with either EC2 or Openstack API calls

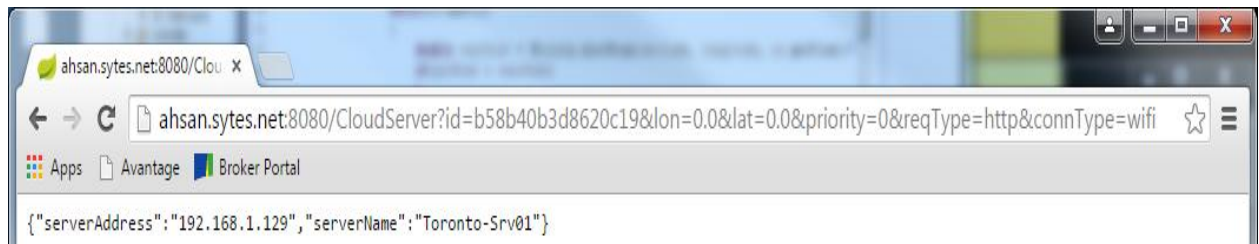
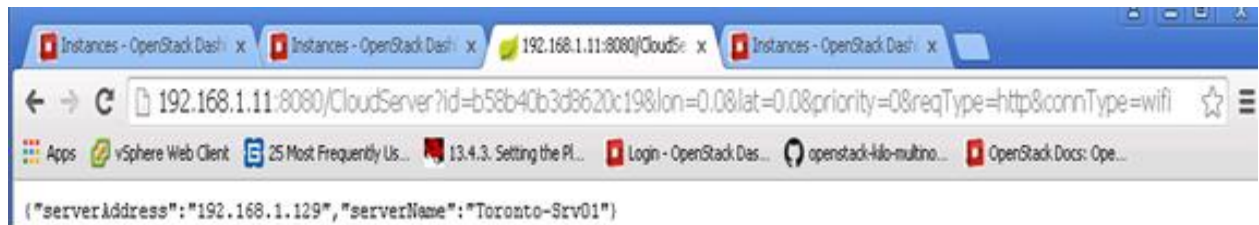
- Cloud systems authenticate each API requests then execute and send the results to CSB.
- CSB calls Java2MySQL to store every queries results to its MySQL database.
- If anytime CSB receives any clients request for cloud service. It uses cloudcontroller.java to store request parameter by cloudserver.java and Java2Mysql.
- Then it calls client.java to analyse the request to find the nearest cloud server by using utility.java, then selects which server runs users required application, then checks if the server is up or down.
- Lastly cloudserver.java decides and sends the results to clients by using Spring MVC framework.
- CSB is also able to create, launch and stop VM (and other services e.g. image, network, etc) in cloud from remote location over the internet by using openstack4j API. We only implemented the VM automated starting service, if not running any VM at all.

## 5.2. Analysis of CSB's Handling Client Request

In this subsection we analyse how CSB handles its client's requests. We discuss below the analysis of test runs of the browser and android applications to send requests, the CSB's handling of the requests and the results.

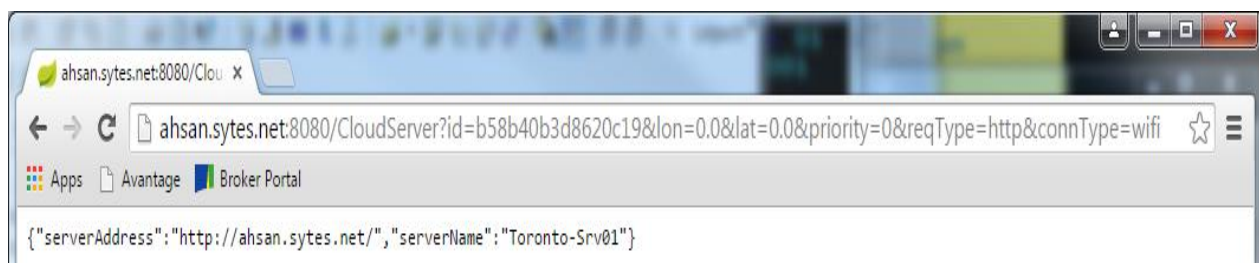
### 5.2.1. x86-64 based Standard Browsers requests Results:

We have tested CSB by sending request from standard x86\*64 based browser applications, which shows that the CSB responses within seconds as the internet connection is faster by using LAN/WiFi connections. **Firstly** we designed CSB to reply with Cloud-VM's IP address and the results are shown below:



In the above figures we tested CSB by sending request from the lab locally and from the remote location via internet. In scenario' we received the results instantly, without any delay at all.

**Secondly**, we designed CSB to reply with Cloud-VM's DNS address and the results are shown below:



In the above figures we tested CSB by sending request from the lab locally and from the remote location via internet. In scenario' we received the results instantly, without any delay at all.

### 5.2.1. Mobile Device's Browsers Request Results:

In this section we tested the CSB by using mobile devices browser by installing bluestack (android emulation application) and using its web browser. To test CSB we developed the following HTTP request parameter with demo longitude, latitude and priority.

- `http://ahsan.sytes.net:8080/CloudServer?id=b58b40b3d8620c19&lon=0.0&lat=0.0&priority=0&reqType=http&connType=wifi`



Figure 5.1: Mobile Browser requests results from CSB

From the figure we can see the mobile browser shows the result sent by CSB programme immediately. We sent the request at 3:23pm and the reply came back within 1.1 second. Again the performance seems excellent not only because of the WiFi connection but also for CSB's quick request handling capacity.

### 5.2.2. Androids Clients-APK Requests Results:

To further analyse the CSB's clients handling capacity, in this subsection we send cloud service request by our Android application. The android application was run on the HTC desire cell phone with 3G data connection.

We installed our android apk in the cell phone named 'ConnectOpenStack'. Before opening the apk we turned on the location services and checked the location by opening google-map apk. Then we opened the ConnecOpenStack.apk. Initially the apk shows blank screen for few second and after 3 seconds shows the cloud-vm's web server page as shown below.



Figure 5.2: Android application's request results from CSB

### 5.2.3. Evaluation of CSB Client handling results

Although we observed CSB's client handling performances by using local Ethernet, WiFi, 3G connection from standard x86-64 computer based browser (google Chrome, Firefox, Internet), and Mobile ARM based Android APK and Android Browser.

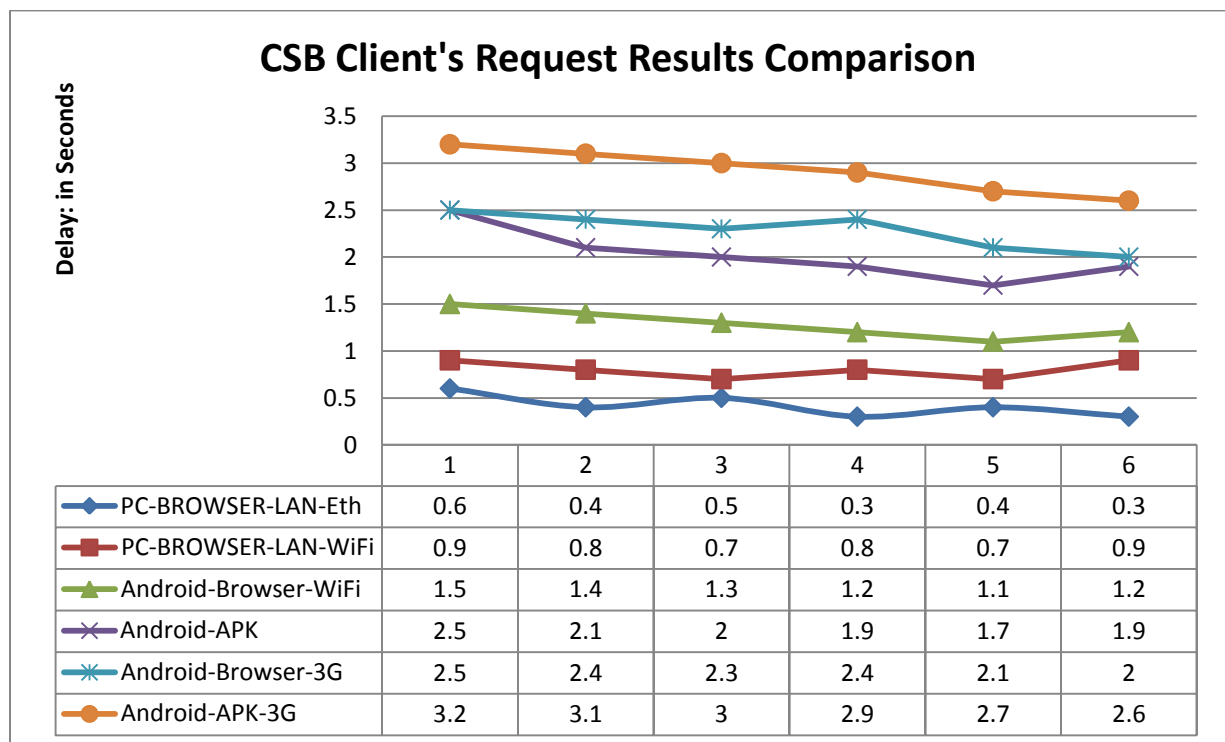


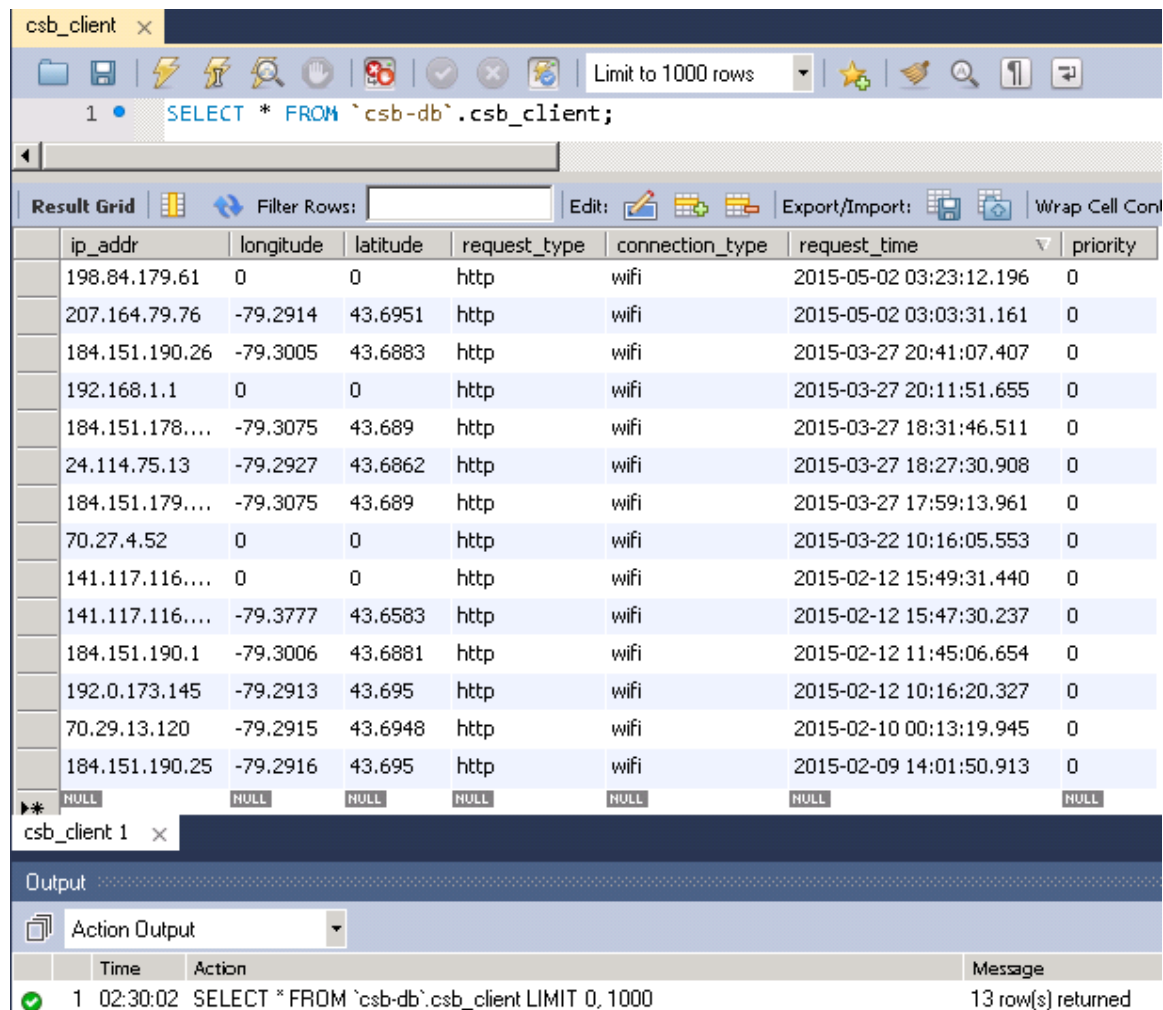
Figure 5.3: CSB Client's Request Results Comparison

The figure above shows that PC based browser has better processing power/Connection that is why PC browser gets the results within 1 second. On the other hand mobile devices has less processing power, and slow connection that is why the first request results take around 3 seconds to get results. but slowly mobile devices gets results quickly as the CSB can reply to client quickly to known clients. This extra 1.1 seconds compared to Wi-Fi connection is nominal as apk need to parse the HTTP results and display in mobile device which take times, and also cell phones CPU and memory are weaker than workstations as well. Based on the results we realised that the CSB can handle both web browser and client's applications with little delay.

### 5.3. CSB Performance analysis

In this subsection we analysed the performance of CSB's enterprise integration tools by reviewing the client's request response time, and measuring the organization and storage of client database by the CSB. The figure below shows how CSB stores and organises clients request parameter.

- CSB clients table named `csb_client` shows the `ip_addr` column is the primary key of the client record that stores location information, connection types, request received time, priority, etc. The client record is used to compare the locations of both clients and destination servers to select the nearest server that supports the client application type and runs the clients requested applications. There are unused information in the client record that can be used in future works for more sophisticated server selection algorithm that performs pattern analysis and/or deal with high priority applications, etc. From our test we realized that the table replaces the duplicate information of the same devices if its IP or other information are changed by using devices unique ID.



**Figure 5.4: CSB Enterprise Integration Tools Performances**

- In the section 5.2.2 and 5.2.3 we observed the application request time was 3.03pm and 3.23pm. It is obvious from the Figure that the CSB's performance is adequate as it received, processed and responded promptly both requests without any delay.

## 5.4. CSB -Cloud integration Performance analysis

In this subsection we analysed the performance of CSB's cloud integration tools by reviewing API request results and the how CSB stores and organizes cloud resources in its databases.

The figure below shows how CSB stores and organises multi-cloud resource information.

- CSB server table named `cloud_servers` shows all the clouds VMs' information in this table including the `ip_addr` that is the primary key, VM'S name, host name, VM state, hypervisors name, availability zone, ID, server location and DNS name etc. These information are used to compare both clients and destination servers' location, application types to select the nearest server that runs the clients requested applications. From our test we realized that the table information change every five minutes if cloud VMs' state is changed. This service is very important for CSB to decide which VM to assign for clients requests. Although the DB shows 6 VM but our CSB uses only 4 VM as other 2 VM's state were not active. So if any cloud VM' is in error or shutoff stage, instantly CSB will find the nearest server and will never forward the requests to non active servers.

1 • `SELECT * FROM `csb-db`.cloud_servers;`

Result Grid

Filter Rows:

Edit

Export/Import

Wrap Cell Content

sip	sname	sstatus	shost	svmstate	sinstancename	hypervisorhostname	availabilityzone	sid	longitude	latitude	sip_dns
192.168.1.129	Toronto-Srv01	ACTIVE	comp1	active	instance-00000019	comp1	nova	8a1de5c...	-79.2984	43.6891	http://a
192.168.1.130	NewYork-Srv01	ACTIVE	comp1	active	instance-0000001a	comp1	nova	34d4de4...	-79.2984	43.6891	http://a
192.168.1.202	London-Srv01	ACTIVE	comp1	active	instance-00000019	comp1	nova	8a1de5c...	-79.2984	43.6891	http://a
192.168.1.203	Paris-Srv01	ACTIVE	comp1	active	instance-0000001a	comp1	nova	34d4de4...	-79.2984	43.6891	http://a
192.168.1.226	UAE-Srv01	ERROR	comp1	error	instance-00000019	comp1	nova	8a1de5c...	-79.2984	43.6891	http://a
192.168.1.227	Tokyo-Srv01	SHUTOFF	comp1	stopped	instance-0000001a	comp1	nova	34d4de4...	-79.2984	43.6891	http://a
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

**Figure 5.5: CSB Cloud Integration Tools Performances**

- Again if the shutoff and error server become active, from that moment CSB will use these server and assign to nearest clients requests.
- Similar to above figure CSB stores cloud resources to 7 other tables for future purpose. Without these tables CSB will not be able to create/destroy/start/stop/suspend/pause and other additional cloud services such as VM, image, network, volume, flavour, etc.



CSB Performance Analysis: If all Cloud-VM’S are at the same Location/Data centre			
If all the Cloud-VMs’ are Active			
Server-Address, Name	Cloud-VM’s Status	CSB’s Selection	Results:  Correct/Wrong
{"192.168.1.129- Toronto-Srv01"}	active	√	100% Accurate
{"192.168.1.130-NewYork-Srv01"}	Active		
{"192.168.1.202-London-Srv01"}	Active		
{192.168.1.203-Paris-Srv01"}	Active		
{"192.168.1.226-UAE-Srv01"}	active		
{" 192.168.227-Tokyo-Srv01"}	active		
If the Cloud-VM with Lowest IP is down			
{"192.168.1.129- Toronto-Srv01"}	Paused/Shutoff		100% Accurate
{"192.168.1.130-NewYork-Srv01"}	Active	√	
{"192.168.1.202-London-Srv01"}	Active		
{192.168.1.203-Paris-Srv01"}	Active		
{"192.168.1.226-UAE-Srv01"}	Error		
{" 192.168.227-Tokyo-Srv01"}	shutoff		
If the Cloud-VM with Lowest IP is up again			
{"192.168.1.129- Toronto-Srv01"}	active	√	100% Accurate
{"192.168.1.130-NewYork-Srv01"}	Active		
{"192.168.1.202-London-Srv01"}	Active		
{192.168.1.203-Paris-Srv01"}	Active		
{"192.168.1.226-UAE-Srv01"}	Error		
{" 192.168.227-Tokyo-Srv01"}	shutoff		

**Table 5.1: CSB Performance Analysis of Context and Location Awareness**

## CSB Context Awareness:

We analyzed CSB Database tables' information against the cloud providers' information and found that the CSB database is 100% accurate.

- for example if any cloud-VM's status change from active to pause/shutoff/error, then CSB DB automatically gets updated after 5 minutes as CSB runs the daemon 'scheduled-executor' every 5 minutes.
- CSB does not store redundant information instead replaces the old data with the updated data.

## CSB Automated Cloud Resource Provisioning:

To test the resource provisioning and automation of cloud services, we modified CSB to monitor cloud VM'S status. If the status of the VM is shutoff then CSB will start the VM via API request parameter. The test result is 100% success as CSB was able to start and stop the VM remotely in our multiple cloud providers' environment.

```
List<? extends Server> servers = os.compute().servers().list();
for (Server server : servers) {
    try{
        //bw.write(getServerInfo(server));

        if(server.getName().equals("London-Srv01") && server.getVmState().equals("stopped"))
            os.compute().servers().action(server.getId(), Action.START);
        if(server.getName().equals("Toronto-Srv01") && server.getVmState().equals("stopped"))
            os.compute().servers().action(server.getId(), Action.START);

        if(server.getName().equals("Paris-Srv01") && server.getVmState().equals("stopped"))
            os.compute().servers().action(server.getId(), Action.START);

        if(server.getName().equals("UAE-Srv01") && server.getVmState().equals("active"))
            os.compute().servers().action(server.getId(), Action.STOP);

        storeServerInfo(server);
    }
}
```

Figure 5.6: CSB Automated Cloud Resource Provisioning

# Chapter 6: Conclusion

## 6. Conclusion

In this chapter, we conclude the thesis by summarizing its contributions and their implications for the advancement of multi-Cloud service brokering research. We present a summary of thesis contribution and discuss the constraints of the achieved contributions, and the possible extensions to mitigate them. Finally, we present potential future research directions.

### 6.1. Summary

The main objective of this research work is to find an answer to the following fundamental question: How can an adequate broker framework for automating and optimizing the deployment of Hybrid or multi-Cloud applications be achieved?

To the answer to the question lies in the design of a cloud service broker framework to assist users in selecting and managing their single and composite services on top of heterogeneous IaaS Clouds. To address this question, we presented the design of a multi-Cloud service broker framework. We further implemented the CSB using Openstack cloud API. We demonstrated the proof of concept working of the CSB through our own Android application that we developed using Spring framework. We chose the more difficult path of developing and implementing the CSB in a test-bed instead of simulating it within CloudSim because it allows our design to integrate the core CSB module with Open Stack cloud for multi-cloud solution, and with other components such as MySQL database and interworking with Android client. It also provides us vehicle to experiment with real life scenarios that can later be developed

into a complete multi-cloud solution. The CSB includes components for enterprise integration tools, queue handler, query analyzer, decision maker, cloud integration tools to monitor and discover clouds and live repositories of cloud resources.

We deployed multi-cloud Openstack and developed android client application to evaluate the broker framework and validate its functionality. Furthermore, to make our test-bed set up more realistic, we used the WiFi and 3G data services with CSB and made the Cloud VM accessible over internet. Using the emulation testbed, we demonstrated the good scalability of the framework with single and multi-Cloud workflow services. From our experiments with simple matching policies, we identified the need for efficient location and context aware necessities in the broker in order to make the trade-off between cost, performance, and quality. The evaluation of real data analysis proved the benefit from deploying CSB on a multi-Cloud compared to a single Cloud in reducing the user payment and improving the service quality, and identified the need for clustering and enhanced data management and scheduling policies.

Overall, this work distinguishes itself from current research in CSB with the following unique contributions:

- A scalable generic multi-Cloud service broker framework
- An efficient location and context aware CSB for selecting composite Cloud services
- Effective platform independent CSB with the ability to handle different types of client types such as RPC, socket, JSON, REST, HTTP, SOAP, XML, etc.
- Most of affordable and popular mobile devices oriented client application developed to benefit from cloud service broker.

## 6.2. Future works

Our CSB framework can be a very useful tool for future research in inter-cloud services consumer. Although we have not implemented many services such as CRM, Helpdesk, Billing, security, user management, monitoring, etc. By using telemetry API we can easily develop billing module for inter-cloud environment. The CSB can play a vital role to save money for consumers by routing cheaper cloud services based on priority of user requests. It can save power consumption for provider by destroying/stopping unused cloud VMs' and other resources. The can also provide security by context awareness for both consumer and provider. We provided building block for future implementation of more elaborate client context management in the CSB. Further, the decision engine in our CSB needs to be integrated with SLA management to ensure the promised user QoS delivery.

# Appendix – A

## 1.1. Openstack Multi-node Cloud Configuration

In our provider cloud Test-bed, we configured 3 cloud provider and named them North America, Europe and Asia. In each cloud providers' environment, we initially deployed a single compute node to test the set up, but we can simply add more compute nodes to make it a multi-node installation if needed:

- Controller Node: It runs cloud management services (keystone, Horizon) needed for OpenStack to function.
- Network Node: It runs networking services and also responsible for virtual network provisioning, connecting virtual machines to external networks.
- Compute Node: It runs the virtual machine instances in OpenStack.

Cloud Provider 1: North America	System	External Network (eth0)	Management Network (eth1)	VM Data Network (eth2)
	Controller	192.168.1.40	192.168.100.140	(not connected)
	Network	192.168.1.41	192.168.100.141	Layer-2 only
	Compute1	192.168.1.42	192.168.100.142	Layer-2 only
Cloud Provider 2: Europe	System	External Network (eth0)	Management Network (eth1)	VM Data Network (eth2)
	Controller	192.168.1.17	192.168.100.3	(not connected)
	Network	192.168.1.18	192.168.100.4	Layer-2 only
	Compute1	192.168.1.19	192.168.100.5	Layer-2 only
Cloud Provider 3: Asia	System	External Network (eth0)	Management Network (eth1)	VM Data Network (eth2)
	Controller	192.168.1.80	192.168.100.180	(not connected)
	Network	192.168.1.81	192.168.100.181	Layer-2 only
	Compute1	192.168.1.82	192.168.100.182	Layer-2 only

Table 1.1: Openstack Multi-node Network types and Interfaces details

For OpenStack Multi-Node setup we need to create three networks:

- External Network: It allows users to access the OpenStack services, interfaces and connects to the network nodes for providing VMs with publicly routable traffic functionality.
- Management Network: It is used for cloud administration only and it is not accessible from public network.
- VM Traffic Network: It is used as internal network for traffic between VM's and also between the VM's and network nodes that provide L3 routes out to the public network.

### 1.2.1. Openstack Installation

We have decided to use the Red Hat RDO distribution on CentOS 6.5 64-bit as it is one of the leaders in the OpenStack community and it comes with the “packstack” installer. Packstack can install a multi-node setup in an automated way, based on an “answer file”.

We configured each VM as follows:

- Centos Installation using the “Minimum” profile with yum update to get the latest updates, with Enabled NTP, and SELinux set ‘permissive’ and ‘EPEL’ is installed.
- Static networking configuration for all nodes.
- The network node has a special network configuration. We created an openvswitch bridge called “br-ex” for external access, and added the physical port “eth0” to it. This is how packstack expects it to be.

ifcfg-br-ex	ifcfg-eth0
DEVICE=br-ex DEVICETYPE=ovs TYPE=OVSBridge ONBOOT=yes	DEVICE=eth0 DEVICETYPE=ovs TYPE=OVSPort OVS_BRIDGE=br-ex



BOOTPROTO=static IPADDR=192.168.1.18 NETMASK=255.255.255.0 NOZEROCONF=yes IPV6INIT=no	ONBOOT=yes NOZEROCONF=yes IPV6INIT=no
---	---

**Table 1.2: OpenStack Network Nodes Interfaces Configuration**

- There's a single SSH key that is installed on every node and that allows each node to ssh to every other node. The key can only be used from the external and management networks. This is achieved by prefixing the public key in “~/.ssh/authorized\_keys” with from=”192.168.0.0/16”

In the controller node, we installed RDO using the “packstack” installer. The screen shot below shows the installation command at the output of packstack

```
[Ahsan.Ahsan-PC] > ssh root@ahsan.sytes.net
Last login: Tue Apr 28 17:30:57 2015 from 192.168.1.11
[root@ctl1 ~]# yum install -y http://repos.fedorapeople.org/repos/openstack/openstack-icehouse/rdo-release-icehouse-3.noarch.rpm
[root@ctl1 ~]# yum install -y openstack-packstack

[root@ctl1 ~]# cd /root/
[root@ctl1 ~]# packstack --gen-answer-file answers.txt
```

**Figure 1.1: Openstack Packstack Answer file generation process**

Once packstack installer is installed, we generated the following ‘answer file’. Then we edited the answer file so that it can install OpenStack on our multi-node and multi-network setup.

<pre>[general] CONFIG_KEYSTONE_INSTALL=y CONFIG_KEYSTONE_HOST=192.168.1.17 CONFIG_SSH_KEY=/root/.ssh/id_rsa.pub CONFIG_MYSQL_INSTALL=y CONFIG_GLANCE_INSTALL=y CONFIG_GLANCE_HOST=192.168.1.17 CONFIG_CINDER_INSTALL=y  # (Nova) CONFIG_NOVA_INSTALL=y CONFIG_NOVA_(API CERT VNCPROXY CONDUCTOR SCHED)_HOST=192.168.1.17 CONFIG_NOVA_COMPUTE_HOSTS=192.168.1.19</pre>	<pre># OpenStack hosts CONFIG_NAGIOS_INSTALL=y # controller service CONFIG_CONTROLLER_HOST=192.168.1.17 # compute service CONFIG_COMPUTE_HOSTS=192.168.1.19 # network service CONFIG_NETWORK_HOSTS=192.168.1.18 # address of DB server CONFIG_MYSQL_HOST=192.168.1.17 # Username for the MySQL admin user CONFIG_MYSQL_USER=root # Password for the MySQL admin user CONFIG_MYSQL_PW=Passw0rd</pre>
---	---

<pre> CONFIG_NOVA_NETWORK_HOST=  # Networking (Neutron). CONFIG_NEUTRON_INSTALL=y CONFIG_NEUTRON_SERVER_HOST=192.168.1.18 CONFIG_NEUTRON_L3_HOSTS=192.168.1.18 CONFIG_NEUTRON_L3_EXT_BRIDGE=br-ex CONFIG_NEUTRON_DHCP_HOSTS=192.168.1.18 CONFIG_NEUTRON_L2_AGENT=openvswitch CONFIG_NEUTRON_L2_PLUGIN=openvswitch CONFIG_NEUTRON_METADATA_HOSTS=192.168.1.18 CONFIG_NEUTRON_OVS_TENANT_NETWORK_TYPE=vlan CONFIG_NEUTRON_OVS_VLAN_RANGES=vmdata:1:4094 CONFIG_NEUTRON_OVS_BRIDGE_IFACES=br-eth2:eth2  # Dashboard (Horizon) CONFIG_HORIZON_INSTALL=y CONFIG_HORIZON_HOST=192.168.1.17 # Storage (Swift) CONFIG_SWIFT_INSTALL=y # Metering (Ceilometer) CONFIG_CEILOMETER_INSTALL=y # Orchestration (Heat) CONFIG_HEAT_INSTALL=y # Client packages. An admin "rc" file will also be installed CONFIG_CLIENT_INSTALL=y </pre>	<pre> # The password to use for Keystone to access DB CONFIG_KEYSTONE_DB_PW=Passw0rd # The token to use for the Keystone service api CONFIG_KEYSTONE_ADMIN_TOKEN=Passw0rd # The password to use for the Keystone admin user CONFIG_KEYSTONE_ADMIN_PW=Passw0rd # The password to use for the Keystone demo user CONFIG_KEYSTONE_DEMO_PW=Passw0rd # The password to use for the Glance to access DB CONFIG_GLANCE_DB_PW=Passw0rd # Glance to authenticate with Keystone CONFIG_GLANCE_KS_PW=Passw0rd # The password to use for the Cinder to access DB CONFIG_CINDER_DB_PW=Passw0rd # The password to use for the Cinder to authenticate with Keystone CONFIG_CINDER_KS_PW=Passw0rd # file-backed volume group and is not suitable for production usage. # Nova to authenticate with Keystone CONFIG_NOVA_KS_PW=Passw0rd # disable RAM overcommitment CONFIG_NOVA_SCHED_RAM_ALLOC_RATIO=1.5 # The password of the nagiosadmin user on the Nagios server CONFIG_NAGIOS_PW=Passw0rd </pre>
--	--

**Table 1.3: Openstack Multi-node Answer file configuration**

The resulting answer file that we used by executing following command:

```
[root@ctl1 ~]# packstack --answer-file answers.txt
```

Installation usually takes about 20 minutes. After we installed OpenStack, the dashboard was available on the http port of the public IP of the controller node. The username and password was stored by packstack in the file `"/root/keystonerc_admin"`. The snapshot of the dashboard is shown in Figure 4.9:

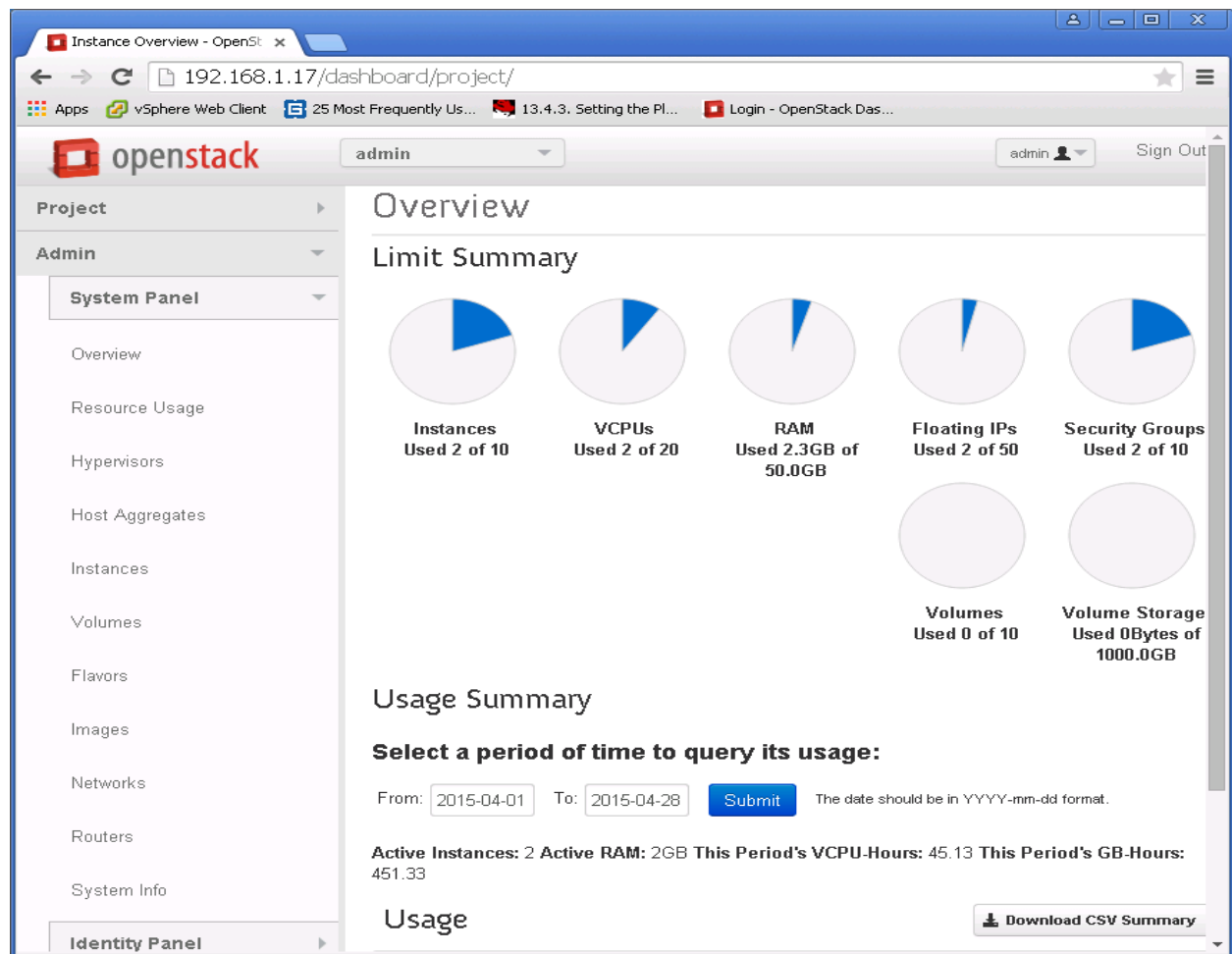


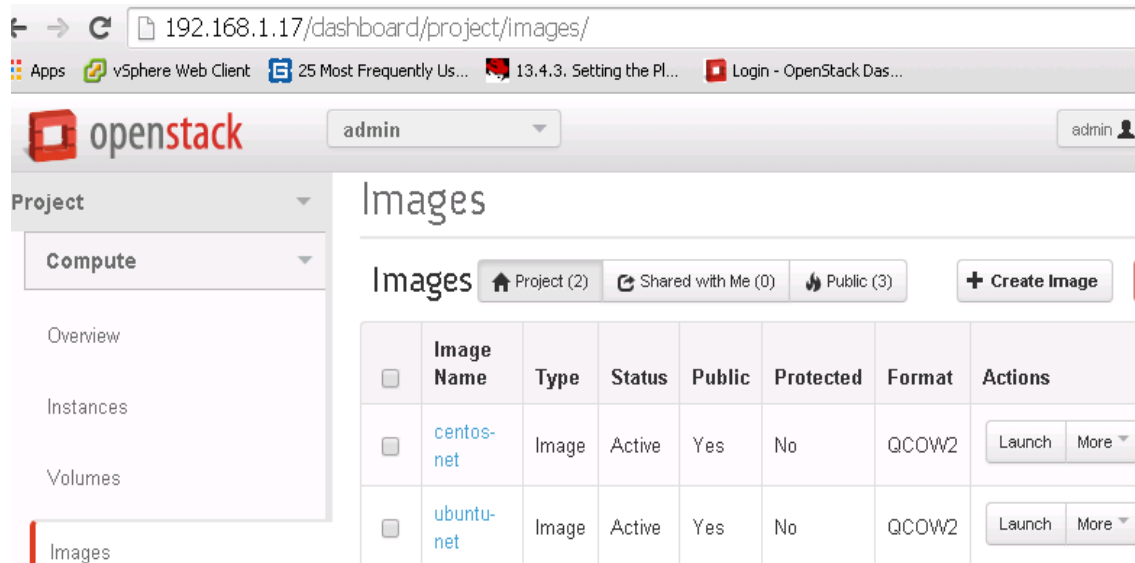
Figure 1.2: Openstack Dashboard for web access

On this screen we can see hypervisor's resources and their CPU, memory, etc that's available.

### 1.2.3. OpenStack Configuration

After installation, we had to do some tweaks because In the IceHouse release there are some missing feature in Packstack where it doesn't configure a new mandatory setting in network and compute node (e.g. hardware virtualization, root password injection, etc which are not shown in this report). To run the first virtual machine, we took the following one-off preparation steps:

- Uploaded Ubuntu and CentOS image into the Glance service. The fact that Glance can download an image straight from URL makes the task very easy. To create the image, go to Admin -> Images -> Create Image.



**Figure 1.3: OpenStack Image Configuration**

- We create a private network and public network and a router between the private and the public network. We did this via the command-line as it didn't seem possible to do all of these from the admin interface:

```
$ source /root/keystonerc_admin
$ neutron router-create router1
$ neutron net-create private
$ neutron subnet-create private 10.0.0.0/24 --name private_subnet \
--enable-dhcp --gateway 10.0.0.1 --dns-nameserver 8.8.8.8
$ neutron router-interface-add router1 private_subnet
$ neutron net-create public --router:external=True
$ neutron subnet-create public 192.168.1.0/24 --name public_subnet \
--disable-dhcp --gateway 192.168.1.1 \
--allocation_pool start=192.168.1.200,end=192.168.1.250
$ neutron router-gateway-set 002718c9-ead1-4908-bc70-e325ac996e94
b1123a0b-afd1-4a80-bd9f-50fde860d663
```

**Table 1.4: Openstack Cloud router configuration**

In the table above, the private network is created using the Google DNS server 8.8.8.8 as we had some trouble getting the OpenStack built-in DNS server to work. The public network is created using the gateway of the external network.

we have also created an allocation pool for floating IPs. In the last command, the <router-id> and <subnet-id> are the router and subnet IDs returned by the subnet-create and router-create commands.

```
[root@ctl1 ~(keystone_admin)]# cd /root/
[root@ctl1 ~(keystone_admin)]# source keystone_admin
[root@ctl1 ~(keystone_admin)]# neutron net-list
+-----+-----+-----+
| id | name | subnets |
+-----+-----+-----+
| 7a10dc36-96c1-4c60-a045-958115d13045 | public | b1123a0b-afd1-4a80-bd9f-50fde860d663 192.168.1.0/24 |
| e583eafa-a82c-47ef-bc67-0a47b34ae840 | private | 15bee16a-906e-4246-965c-b593bb9cdcff 10.0.0.0/24 |
+-----+-----+-----+

[root@ctl1 ~(keystone_admin)]# neutron subnet-list
+-----+-----+-----+-----+
| id | name | cidr | allocation_ |
+-----+-----+-----+-----+
| b1123a0b-afd1-4a80-bd9f-50fde860d663 | public_subnet | 192.168.1.0/24 | {"start": "192.168.1.201", "end": "192.168.1.251"} |
| 15bee16a-906e-4246-965c-b593bb9cdcff | private_subnet | 10.0.0.0/24 | {"start": "10.0.0.2", "end": "10.0.0.254"} |
+-----+-----+-----+-----+
```

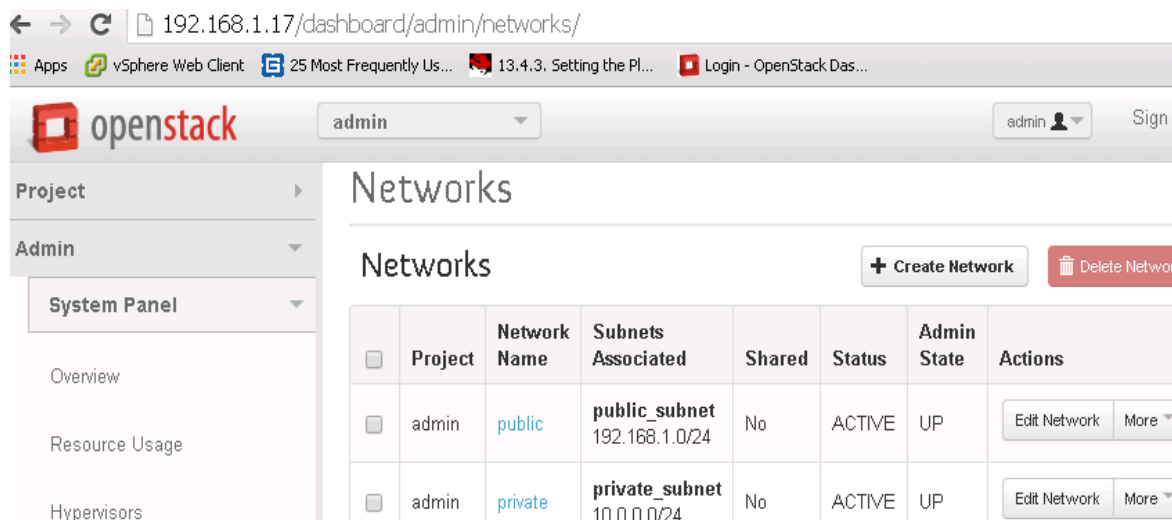


Figure 1.4: Openstack neutron network for Instances

- We updated the flavour list such as 'm1.small' flavor to have 1vCPU, 1200MB of memory and 10GB of ephemeral storage.
- Security Group & key pair: we created security group named SSH and added rules for HTTP, ICMP, SSH for ingress and egress traffic. We also created a key pair and uploaded controller public key. As Key pairs are SSH credentials which are injected into images when they are launched
- Launching an Instance: we launched a new instance by following these steps: Create a new instance by going to Project -> Instances -> Launch Instance. Then Select the "m1.small" flavor, "boot from Image", and then the Ubuntu14 image. On the "Access and Security" tab it is recommend setting a root password. On the "Networking" tab we connected to the 'private & public network'. Then finally click "Launch".

## Instances

Instances											
			Filter		Filter		+ Launch Instance		Soft Reboot Instances		Terminate Instance
	Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Uptime	Actions
<input type="checkbox"/>	NewYork-Srv01	centos-net	10.0.0.36 192.168.1.130	m1.small   1GB RAM   1 VCPU   10.0GB Disk	ssh	Active	nova	None	Running	1 year	Create Snapshot More
<input type="checkbox"/>	Toronto-Srv01	ubuntu-net	10.0.0.35 192.168.1.129	m1.small   1GB RAM   1 VCPU   10.0GB Disk	ssh	Active	nova	None	Running	1 year	Create Snapshot More

**Figure 3: Openstack Cloud Instances**

Once the instance is running then we can access the console over VNC. We can access instances console both Openstack dashboard and CLI.

```
[root@ctl1 ~](keystone_admin)]# nova get-vnc-console 8a1de5c3-5176-4ae6-9382-59c70b217de1 novnc
+-----+
| Type | Url |
+-----+
| novnc | http://192.168.1.17:6080/vnc_auto.html?token=adc21c34-d835-4bfd-926b-fb77f3cf616a |
+-----+
```

**Figure 1.5: Openstack Instances VNC access process**

### 1.2.3.1. Openstack Instance's web-server Configuration

We need at least one cloud instances up and running that can act as an application server for our CSB client. Since our CSB client application is designed for web service, we

need to configure web server in one of the cloud instances. We have configured our cloud instance 'Ubunut' as a web server by installing apache2 web server.

```
[root@ctl1 ~]# ssh stack@192.168.1.202
stack@192.168.1.202's password:
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-34-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

System information as of Thu Apr 30 01:35:19 EDT 2015

System load:  0.22               Processes:            100
Usage of /:   31.6% of 8.56GB    Users logged in:     0
Memory usage: 14%               IP address for eth0: 10.0.0.35
Swap usage:   0%

Graph this data and manage this system at:
  https://landscape.canonical.com/

339 packages can be updated.
147 updates are security updates.

Last login: Thu Apr 30 01:36:46 2015 from 192.168.1.17
stack@host-10-0-0-35:~$ su root
Password:
root@host-10-0-0-35:/home/stack# sudo apt-get install apache2
```

Figure 1.6: Openstack Instances configuration as a web server

After we have configured the apache web server in the cloud instance and added dns name ahsan.sytes.net, now the cloud VM is accessible via internet from any devices with standard and mobile browser application.



Figure 1.7: Openstack Instance acting as a public web server

# Appendix - B

## List of Table

Table 3.1: CSB Algorithm (Context & Location Aware) -----	40
Table 4.1: POM.XML of CSB Java Project-----	47
Table 5.1: CSB Performance Analysis of Context and Location Awareness-----	65

## List of Acronyms

API	Application Programming Interface
ARM	Advanced RISC Machines
AWS	Amazon Webs Services
CSB	Cloud Service Broker
CSC	Cloud Service Provider
CSP	Cloud Service Consumer
DC	Data Centre
DCS	Desktop Cloud Service
EC2	Elastic Compute Services
GA	Genetic Algorithm
GUI	Graphic User Interface
HPC	High Performance Computing
HTTP	Hyper Text Transfer Protocol
IaaS	Infrastructure as a Service
ID	Identifier
I/O	Input/output
J2EE	Java 2 Platform, Enterprise Edition
J2ME	Java 2 Platform, Micro Edition
JSON	JavaScript Object Notation
MCA	Mobile Cloud Application
OCCI	Open Cloud Computing Interface
DB	Database
PaaS	Platform as a Service
QoS	Quality of Service
RISK	Reduced Instruction Set Computer
REST	Representational State Transfer
S3	Simple Storage System
SaaS	Software as a Service
SLA	Service Level Agreement
SME	Small and Medium Enterprises
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UI	User Interface



URI	Uniform Resource Identifier
SAN	Storage Area Network
VM	Virtual Machine
WWW	World Wide Web
XML	Extensible Markup Language

## Appendix – C

### Bibliography

- [1] B. Furht and A. Escalante, editors. Handbook of Cloud Computing. Springer, 2010.
- [2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical report, University of California at Berkeley, February 2009.
- [3] G. Modica and O. Diand Tomarchio. A Semantic Discovery Framework to support supply-demand Matchmaking in Cloud Service Markets. In Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER2012), Porto, Portugal, 2012.
- [4] Forrester;Tools And Technology: The I&O Practice Playbook; Prepare Your Infrastructure And Operations For 2020 With Tools And Technologies; Jean-Pierre Garbani et all; July 25, 2013
- [5] Global Inter-Cloud Technology Forum. Use Cases and Functional Requirements for Inter-Cloud Computing. [Online], 2010.  
[http://www.ttc.or.jp/files/8614/1214/5480/GICTF\\_Whitepaper\\_20100809.pdf](http://www.ttc.or.jp/files/8614/1214/5480/GICTF_Whitepaper_20100809.pdf) (accessed: 2014-10-13).
- [6] Inc Gartner. Gartner Says Cloud Consumers Need Brokerages to Unlock the Potential of Cloud Services. [Online], 2009. <http://www.gartner.com/newsroom/id/1064712> (accessed: 2014-09-15).
- [7] Gideon Juve, Ewa Deelman, G.Bruce Berriman, Benjamin P. Berman, and Philip Maechling. An Evaluation of the Cost and Performance of Scientific Workflows on Amazon EC2. Journal of Grid Computing, 10:5–21, 2012.

- [8] SensibleCloud. [Online], 2014. <http://www.sensiblecloud.com/> (accessed: 2014-03-20).
- [9] Rajkumar Buyya, Suraj Pandey, and Christian Vecchiola. Market-Oriented Cloud Computing and The Cloudbus Toolkit, pages 319–358. John Wiley, Inc, 2013.
- [10] Ana Juan Ferrer, Francisco Hernandez, Johan Tordsson, Erik Elmroth, Ahmed Ali-Eldin, Csilla Zsigri, Ral Sirvent, Jordi Guitart, Rosa M. Badia, Karim Djemame, Wolfgang Ziegler, Theo Dimitrakos, Sriji K. Nair, George Kousiouris, Kleopatra Konstanteli, Theodora Varvarigou, Benoit Hudzia, Alexander Kipp, Stefan Wesner, Marcelo Corrales, Nikolaus Forg, Tabassum Sharif, and Craig Sheridan. Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66–77, 2012.
- [11] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: The montage example. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC '08*, pages 50:1–50:12, Piscataway, NJ, USA, 2008. IEEE Press.
- [12] Hamid Mohammadi Fard, Radu Prodan, Juan Jose Durillo Barrionuevo, and Thomas Fahringer. A Multi-objective Approach for Workflow Scheduling in Heterogeneous Environments. *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 300–309, May 2012.
- [13] J. Octavio Gutierrez-Garcia and Kwang Mong Sim. Agent-based Cloud Workflow Execution. *Integrated Computer-Aided Engineering*, 19:39–56, 2012.
- [14] Suraj Pandey, Dileban Karunamoorthy, and Rajkumar Buyya. *Workflow Engine for Clouds*, pages 321–344. John Wiley, Inc, 2011.
- [15] Nikolay Grozev and Rajkumar Buyya. Inter-Cloud architectures and application brokering: taxonomy and survey, pages 1–22. John Wiley, Inc, 2012.
- [16] Foued Jrad, Jie Tao, and Achim Streit. Simulation-based evaluation of an intercloud service broker. In *CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDS, and Virtualization*, pages 140–145, 2012.

- [17] Elarbi Badidi, "A Cloud Service Broker for SLA-based SaaS Provisioning," Proc. of Information Society, 2013, pp. 61-66.
- [18] Alba Amato, Beniamino Di Martino, Salvatore Venticinqu, "Cloud Brokering as a Service," Proc. of P2P, Parallel, Grid, Cloud and Internet Computing, 2013, pp. 9-16.
- [19] P. Mell and T. Grance. The NIST Definition of Cloud Computing. [Online], 2011.  
<http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf> (accessed: 2014-12-20).
- [20] David S. Linthicum. Cloud Computing and SOA Convergence in your Enterprise. Addison-Wesley Professional, 2010.
- [21] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. University of California at Berkley, USA. Technical Rep UCB/EECS-2009-28, 2009.
- [22] Gridbus Project. Gridbus service broker. [Online], 2014. <http://www.cloudbus.org/broker/> (accessed: 2014-12-22).
- [23] Inc Gartner. Three Types of Cloud Brokerages Will Enhance Cloud Services. [online], 2009.  
<https://www.gartner.com/doc/973412/types-cloud-brokerages-enhance-cloud> (accessed: 2014-12-25).
- [24] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger and Dawn Leaf; NIST Cloud Computing Reference Architecture. [online] 2011.  
[www.nist.gov/customcf/get\\_pdf.cfm?pub\\_id=909505](http://www.nist.gov/customcf/get_pdf.cfm?pub_id=909505) (accessed: 2014-12-25).
- [25] Inc Jamcracker. Enabling Cloud-Based Digital Business Models. [online], 2015.  
[www.jamcracker.com/sites/default/files/enabling\\_cloud-based\\_digital\\_business\\_models\\_0.pdf](http://www.jamcracker.com/sites/default/files/enabling_cloud-based_digital_business_models_0.pdf) (accessed: 2014-12-27).
- [26] Stefan Ried, Ph.D. with Pascal Matzke, Jean-Pierre Garbani, Reedwan Iqbal; FOR CIO PROFESSIONALS; Cloud Broker — A New Business Model Paradigm Deriving More Business And Economic Models From Cloud Computing; forrester research reports, 2011
- [27] CCUC-DG. Cloud Computing Use Cases White Paper Version 4.0. Technical report, Cloud Computing Use Case Discussion Group, 2010.

- [28] Amazon Elastic Compute Cloud. [Online], 2014. <http://aws.amazon.com/ec2/> (accessed: 2015-01-20).
- [29] Amazon Simple Storage Service. [Online], 2014. <http://aws.amazon.com/s3/> (accessed: 2015-01-21).
- [30] Claudia Szabo, Quan Z. Sheng, Trent Kroeger, Yihong Zhang, and Jian Yu. Science in the Cloud: Allocation and Execution of Data-Intensive Scientific Workflows. *Journal of Grid Computing*, October 2013.
- [31] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. Intercloud: Utilityoriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I, ICA3PP'10*, pages 13–31, Berlin, Heidelberg, 2010. Springer-Verlag.
- [32] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, June 2009.
- [33] Rodrigo N. Calheiros, Christian Vecchiola, Dileban Karunamoorthy, and Rajkumar Buyya. The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds. *Future Generation Computer Systems*, 28(6):861–870, June 2012.
- [34] R. Knapper, C.M. Flath, B. Blau, A. Sailer, and C. Weinhardt. A multi-attribute service portfolio design problem. In *Service-Oriented Computing and Applications (SOCA)*, 2011 IEEE International Conference on, pages 1 –7, dec. 2011.
- [35] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Software: Practice and Experience*, 41(1):23–50, January 2011.
- [36] D. Ardagna, E. Di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D’Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu, A. Gericke, and C. Sheridan. ModacLOUDS: A model-driven approach for the design and execution of applications on multiple clouds. In *Modeling in Software Engineering (MISE)*, 2012 ICSE Workshop on, pages 50–56, June 2012.

- [37] Inc Spring; Introduction to the Spring Framework. [online] 2015.  
<http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html>  
(accessed: 2015-02-15).
- [38] Inc Openstack4j. What is OpenStack4j?. [online] 2015. [www.openstack4j.com/learn/](http://www.openstack4j.com/learn/)  
(accessed: 2015-02-19).
- [39] Inc Openstack. Openstack architecture design guide; [online ], 2015.  
<http://docs.openstack.org/arch-design/content/arch-design-architecture-hardware.html>  
(accessed: 2015-02-22).
- [40] Tom Fifield; Diane Fleming; Anne Gentle; Lorin Hochstein; Jonathan Proulx; Everett Toews; Joe Topjian; OpenStack Operations Guide; O'Reilly Media, Inc; May 8, 2014; ISBN-13: 978-1-4919-4695-4
- [41] Inc Android. Android Studio Overview. [online], 2015.  
<http://developer.android.com/tools/studio/index.html> (accessed: 2015-02-23).
- [42] Inc. Eclipse. COMPARE ECLIPSE PACKAGES. [online], 2015.  
<http://eclipse.org/downloads/compare.php?release=luna> (accessed: 2015-02-25).
- [43] Inc MySQL. MySQL Workbench 6.3- Enhanced Data Migration. [online] 2015.  
<https://www.mysql.com/products/workbench/> (accessed: 2015-02-27).