TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

FARSHAD AHMADI GHOHANDIZI

CLOUD-BASED SOFTWARE DEVELOPMENT FOR A FEDERATED CLOUD

Master of Science thesis

Examiner: Prof. Kari Systä
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineering
on 8 October 2014.

ABSTRACT

FARSHAD AHMADI GHOHANDIZI: CLOUD-BASED SOFTWARE DEVELOPMENT FOR A FEDERATED CLOUD
Tampere University of Technology
Master of Science thesis, 50 pages
October 2014
Master's Degree Programme in Information Technology
Major: Cloud Computing and Software Development Automation Practice
Examiner: Prof. Kari Systä
Keywords: Cloud Computing, PaaS, IDE, Continuous Deployment, COAPS, ACCORDS, EASI-CLOUDS

Cloud computing provides on demand network access to a shared pool of computing resources that can be provisioned with minimal effort. These resources could be categorised in 3 different levels including software, platform, and infrastructure. As cloud computing gains traction, there is an increasing trend to switch from desktop application to cloud-based ones. In this transition IDEs are not an exception. Some cloud-based IDEs like MIDEaaS are already developed. Now the question is how these IDEs could perform SW development and deployment.

The present thesis implements a cloud-based SW development demonstrator and investigates issues of such an implementation. Developers should be able to use our demonstrator to develop an application and deploy it into the best Platform as a Service (PaaS) provider among the available PaaS providers. The SW development practice our demonstrator follows is continuous deployment which automates phases of SW development cycle including build and release. Since this thesis is part of EASI-CLOUDS project, adopting our demonstrator to multi-provider PaaS architecture is by utilising EASI-CLOUDS platform. EASI-CLOUDS platform federates available clouds (in our case PaaS providers) and negotiates relationship between them and cloud consumer (in our case developers). Due to these two goals, two components (ACCORDS and COAPS) are developed in EASI-CLOUDS project.

Based on the way ACCORDS performs to broker and find the most suitable PaaS provider, two use cases are presented. First use case is based on deferred deployment method of ACCORDS which allows a developer to have control (e.g. stop, start, undeploy, etc.) over his deployed applications on the target PaaS. This use case was not feasible since current implementation of ACCORDS does not support it. Second use case is based on immediate deployment method of ACCORDS. Its main pitfall is losing control over the deployed applications.

PREFACE

The research work related to this Master of Science thesis is conducted in EASI-CLOUDS (Extensible Architecture and Service Infrastructure for Cloud-Aware Software) project as part of ITEA (Information Technology for European Advancement) organisation, funded by Finnish Funding Agency for Technology and Innovation (Tekes).This thesis is done during the year 2014 at the department of Pervasive Computing, Tampere University of Technology.

I would like to express my great gratitude to my supervisor Prof. Kari Systä for the given opportunity. It was a great pleasure to work under his precious guidance and friendly support. I have learned a lot under his supervision. His patience and willingness to impart his knowledge is outstanding. I would like to extend my thank to Dr. Janne Lautamäki, M.Sc. Otto Hylli, and M.Sc. Antti Nieminen for sharing their work and experience and for such a great cooperation we have had during the research work.

I would also like to express my appreciation to all my friends in Tampere specially Nader Daneshfar, Mojtaba Sarooghi, Farid Shamani, and Seyed Abolfazl Hosseini who, in more ways than one, helped me in my academic and personal life.

My special thanks go to my family who has unconditionally supported me all the time. I owe everything I have achieved to them. Without their sustaining support, none of this would have been possible.

Finally I would like to express my deepest appreciation to my wonderful and patient love, Zeinab Ashjaei, for always firmly standing beside me and filling me with her peace.

Tampere, 31 December 2014

Farshad Ahmadi Ghohandizi

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

ACCORDS	Advanced Capabilities for CORDS
API	Application Programming Interface
App	Application
AWS	Amazon Web Services
CD	Continuous Delivery
CDMI	Cloud Data Management Interfaces
CI	Continuous Integration
COAPS	Compatible One Application and Platform Service
CORDS	CompatibleOne Resource Description System
CoRED	Collaborative Real-time Editor
CSB	Cloud Service Broker
DB	Data Base
EASI-CLOUDS	Extensible Architecture and Service Infrastructure for Cloud-Aware Software
Env	Environment
IaaS	Infrastructure as a Service
ID	Identifier
IDE	Integration Development Environment
IT	Information Technology
ITEA	Information Technology for European Advancement
MIDEaaS	Moblide IDE as a Service
NIST	National Institute of Standards and Technology
OCCI	Open Cloud Computing Interface
OS	Operating System
PaaS	Platform as a Service
SaaS	Software as a Service
SCM	Source Control Management
SLA	Service Level Agreement
SW	Software
TUT	Tampere University of Technology
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine
WAR	Web Application Resource

1. INTRODUCTION

1.1 Introduction

From the first day internet appeared to the present day, computation has been moving to different stages and different types of computing has been introduced, including parallel computing, distributed computing, and grid computing. One type of computing that has gained traction recently is cloud computing. Cloud computing enables convenient and on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, services, etc.) that can be rapidly provisioned and released with minimal management effort and lower cost [25]. Customer is not concerned about infrastructure and everything related to it like installation and maintenance. In addition to infrastructure level, this also holds true in platform level, where services like database can be offered to customers. This can even happen in application level. Applications could be installed and maintained in a centralised location and accessed remotely via internet [18]. Therefore, Infrastructure, platform and applications could be offered as a cloud resource as a service to consumers. There are a large number of theses service providers in real market and their number is increasing every day. This causes certain sort of problems. Firstly, use of services provisioned by these cloud providers needs knowledge specific to each of them. They do not follow any unified standard to offer their services. Communicating with them necessitates a consumer to invest time and learn their specific standards and APIs, resulting in vendor lock-in [29]. Secondly, due to high number and vast variety of cloud service providers finding the appropriate one that meets user's expectation (in terms of availability of requested services, security, cost, etc.) is a hard job for consumer [37].

This thesis is part of EASI-CLOUDS[1] project. EASI-CLOUDS stands for Extensible Architecture and Service Infrastructure for Cloud-Aware Software. The objective of EASI-CLOUDS is to provide comprehensive, innovative, and open-source cloud platform, EASI-CLOUDS platform. Major expected outcome of EASI-CLOUDS platform is to offer beneficial solutions for cloud end users, cloud providers, and de-

[1] `http://easi-clouds.eu`

velopers by removing obstacles cloud computing is facing. It helps cloud consumers adopt multi-cloud architecture and avoid the vendor lock-in problem. It tries to make cloud providers interoperable to interlink their clouds. EASI-CLOUDS platform also exploits cloud computing power to simplify software development practice. The latter one is also thesis contribution.

One of the main problems in software development cycle is creation of development environments, build environments, test environments, staging environments, and production environments. It is both time consuming and hard. Since cloud computing provides instant access to cloud resources, it simplifies and accelerates the creation of these environments. Another major problem in software development cycle is the gap between the development and test and the deployment process [11]. Usually the development and testing team and the operation team are distinct teams mostly concentrating on their own responsibilities. Lack of decent communication among them results in delay or even failure in software release. There is a need to simplify feedback cycle among these groups. Continuous delivery as a software development practice offers a way by automating build, test, and release cycle and providing visible feedback to all members involving in project. One of the best practices of CD is to make testing and staging environment as closely as possible similar to production environment [17]. This is where cloud computing could show its power. It can simplify the interact of developers and testers with supporting IT systems. Team members could request production-like cloud resources to create (test and staging) environments that perfectly mimic production environments [6].

The present thesis helps EASI-CLOUDS demonstrate and investigate issues of software development for a federated cloud. The whole lifetime of a software project from development to build to test and finally to deployment on the most appropriate cloud providers will be covered.

1.2 Purpose of the Thesis

Main purposes of this thesis are listed as following:

- Investigate issues of software development (continuous integration/delivery/deployment) from cloud-based IDE to a federated/brokered cloud.
- Development of a demonstrator to show cloud-based software development for a federated/brokered cloud.
- Study two main components of EASI-ClOUDS (COAPS and ACCORDS), and some technologies (e.g. Git, Jenkins, Openstack, CloudFoundry) to investigate

feasibility of development of the demonstrator previously mentioned as second item.

1.3 Research Question

Two research questions are:

1. How should 'cloud based software development for a federated cloud' be done?
2. Can available technologies and current implementations of EASI-Clouds components answer the first research question?

1.4 Structure of the Thesis

The present thesis is divided into 5 chapters. Chapter 1 is introduction in which the research question and structure of the thesis are provided. Chapter 2 explains cloud computing, EASI-CLOUDS project (the project this thesis is part of), objectives of EASI-CLOUDS and the thesis, software development practices including continuous integration, continuous delivery, and continuous deployment. Chapter 3 illustrates details about researcher's implementation. In chapter 4 results and findings achieved by implementation are presented. Conclusion and further work are stated in chapter 5.

2. BACKGROUND

2.1 Cloud Computing

There has been a substantial migration from desktop applications to cloud-based software/services/resources during past decades [31]. But cloud computing is a broad term offering a wide variety of services. In order to know how to use cloud computing appropriately, first we should define it and its characteristics (subsection 2.1.1) and then its different service models (subsection 2.1.2).

2.1.1 Definition and Characteristics

One of the most acceptable definitions of Cloud Computing is represented by National Institute of Standards and Technology (NIST)[1]. It says:

> Cloud computing is a 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 [25].

NIST also mentions five essential characteristics for Cloud Computing services:

- On-demand self-service. The ability of consumer to use resources without the need for interaction with vendor.
- Broad Network Access. Availability of resources through standard client platforms (e.g laptops, tablets, mobiles, etc.)
- Resource pooling: Vendors' resources are pooled so that multiple consumers can use them.

[1] http://www.nist.gov/

- Rapid elasticity: Scaling of resources with demand.
- Measured service: Monitoring and reporting resource usage.

2.1.2 Service Categories: SaaS, PaaS, IaaS

Let's first assume that a programmer wants to develop a software and deploy it into the cloud, but with one limitation: he should do all things from scratch. He is neither given a data center nor any higher level platform (Operating System) to deploy his software. He cannot even use any other ready-made running piece of software to develop his software. He should build all of them from a to z. First, he needs an infrastructure, a data center, to give him resources like processing, networking, and storage. Second, he needs a platform (alongside some services like Data Base Server, Application Server, etc.) on top of infrastructure in order to run the software. Now, he can develop and deploy his software. Figure 2.1 shows the process.

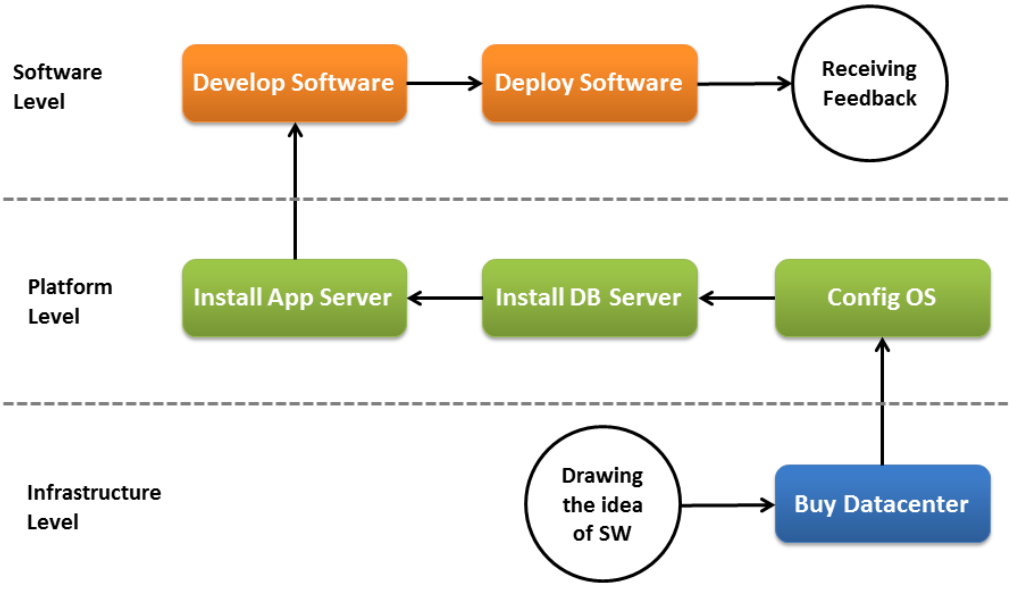

Figure 2.1 *Deploying software if forced to do all things alone*

Although nobody undergoes the whole process shown in figure 2.1 just to deploy a software, thinking about it can help a lot to understand what cloud is and what kind of cloud computing services organisations can use. Figure 2.1 could be divided into 3 parts, representing different types of Cloud Computing services named Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). Figure 2.2 classifies and shows these services.

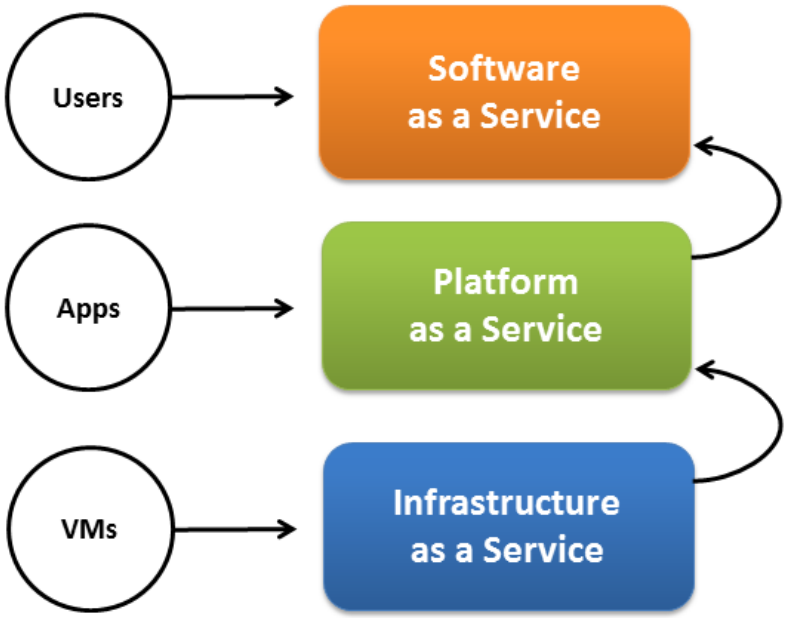

***Figure** 2.2 Different types of Cloud Computing services*

A simple conclusion could be drawn from figures 2.1 and 2.2; if there is no need to go down in the stack, do not go. If there is a software available to use (as a service), do not reinvent the wheel, just use it. For example, Google Map is already there, coming up with a new map application is a waste of time. But what if a person has a new idea? What if he wants to develop a new application? Now there is a need to go one level down in cloud computing stack (to Platform as a Service) in order to prepare all services needed for the application to be deployed and run. Finally, if he wants to launch a new Virtual Machine in cloud and prepare essential services on it to deploy his application, he should go to the lowest level, Infrastructure as a Service.

For each level of Cloud Computing stack, there exist some solutions. In this project we use solutions like CloudFoundry[2] for PaaS and OpenStack[3] for IaaS. The rest of this section goes into more detail about these solutions.

SaaS (Software as a Service)

SaaS applications are running on a cloud infrastructure and delivered to user over the web. Users can access these applications via thin client interfaces (like browsers) or

[2] `http://cloudfoundry.org/`
[3] `https://www.openstack.org`

application programming interfaces (APIs). Underlying cloud infrastructure, hardware resources (server, storage and network) and software resources (like operating systems), and even application configuration settings (except some customisations) could not be handled by users [25]. Characteristics of SaaS include [34] [15]:

- Web applications are available globally any time anywhere.
- Updates and patches can be distributed by updating application on server.
- No maintenance or back up is needed on client side, because software is managed from a central location.
- There is no need for installation on local machines; applications are already installed on server and delivered in a "one to many" model.
- Different pieces of software could be integrated by the help of Application Programming Interfaces (APIs)

PaaS (Platform as a Service)

The same benefits SaaS brings in the software domain, PaaS provides in software development world; instead of delivering software over the web, platform for offering and running applications is delivered [34]. PaaS enables users to deploy applications that are developed by programming languages, libraries and services supported by the PaaS provider. Similarly in this level (like in SaaS level), underlying cloud infrastructure could not be controlled by the user, but the deployed application and some settings of hosting environment is under control of the user [25]. Characteristics of PaaS include [34]

- Services to develop, test, deploy, host and maintain applications.
- Multiple users can use the same development tool simultaneously.
- Collaboration tools for development team.
- Tools for service management and billing.

CloudFoundry as PaaS Solution

As said in subsection 2.1.2, the application platform can be delivered over the web known as Platform as a Service (PaaS). PaaS offers a range of resources (application services, runtimes, development frameworks and languages, cloud deployment environments). Developers can deploy, run and scale their applications created using resources supported by a PaaS. Obviously, different PaaS provide developers different resources. Some of them have limited runtimes and frameworks. Others support

deploying to a single cloud (private, public or Hybrid). By using an open PaaS, a developer can choose resources to deploy and run his application.

CloudFoundry is an open source cloud computing PaaS developed in Ruby by VMware under the terms of Apache License 2.0. CloudFoundry is open PaaS, means while most PaaS offer restricted developer frameworks, application services and deployment clouds, Cloud Foundry avoids vendor lock-in because of its open and extensible nature. It supports multiple development frameworks like Ruby on Rails, Sinatra for Ruby, Node.js, Spring for Java. Although all frameworks are not supported, the open architecture will allow supporting new ones. Cloud Foundry supports application services including MongoDB, MySQL and Redis databases. Moreover, In Cloud Foundry, application deployment can be done to private and public cloud environments including VMware vSphere, non-VMware public clouds and AWS (Amazon Web Services). Developers can use Cloud Foundry specific tool to deploy their applications without the need for any change in their codes [33] [35].

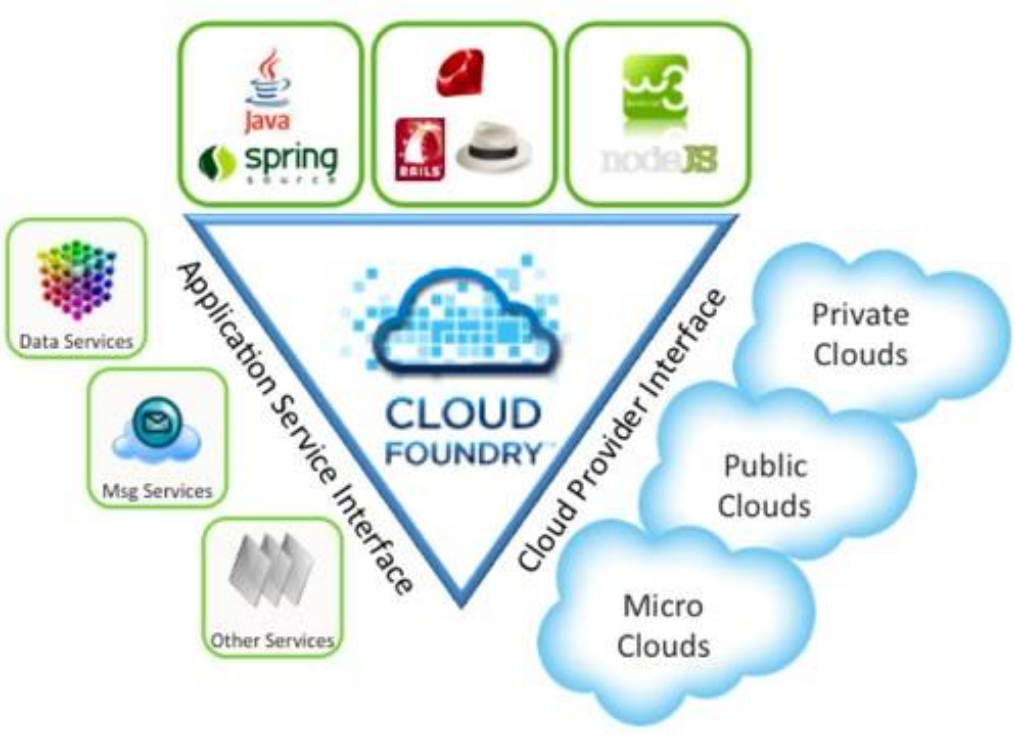

Figure 2.3 *Cloud Foundry [36]*

IaaS (Infrastructure as a Service)

In this level, Cloud Computing infrastructure is delivered over the web. Instead of buying the whole infrastructure, user can purchase compute, storage, network and other fundamental resources on-demand (as much as needed) [25]. Underlying cloud

infrastructure is not under control of user, but he can control operating systems, storage, deployed applications and some networking management. Characteristics of IaaS include [34]

- Distributed resources as services.
- Dynamic scaling.
- A piece of hardware can be used by multiple users.

Openstack as IaaS Solution

OpenStack[4] is an open-source cloud operating system (cloud computing platform) mainly used as an Infrastructure as a Service (IaaS) solution for public and private clouds. It contains several components that communicate with each other to control pools of (three main) resources of a data center: processing (compute), storage and networking. These components can be managed and controlled through command line, RESTful APIs or Openstack dashboard [26]. Figure 2.4 shows the whole idea in a simple picture.

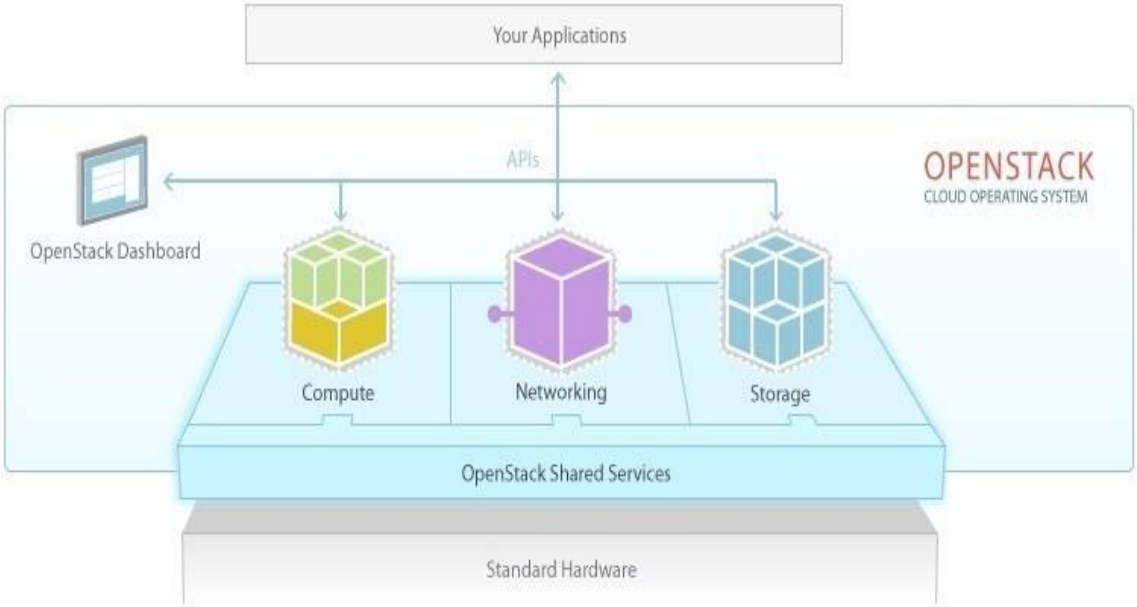

Figure 2.4 *Openstack Software Diagram [26]*

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

2.1.3 Cloud Federation/Brokerage

Interlinking clouds of cloud providers can play an important role in cloud computing success. It offers benefits for both cloud providers and cloud consumers. For example, one cloud provider which lacks or ran out of a certain resource can buy and utilise that of another cloud provider which is unused. In order to achieve this goal the terms cloud federation and cloud brokerage emerge in cloud computing world. Figure 2.6 depicts basic idea of cloud federation and cloud brokerage.

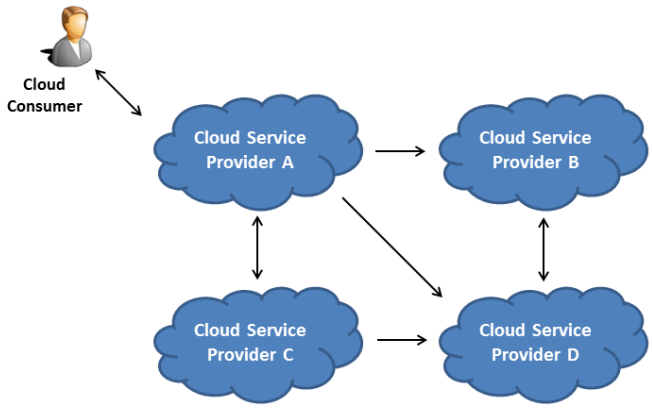

Figure 2.5 *Cloud federation*

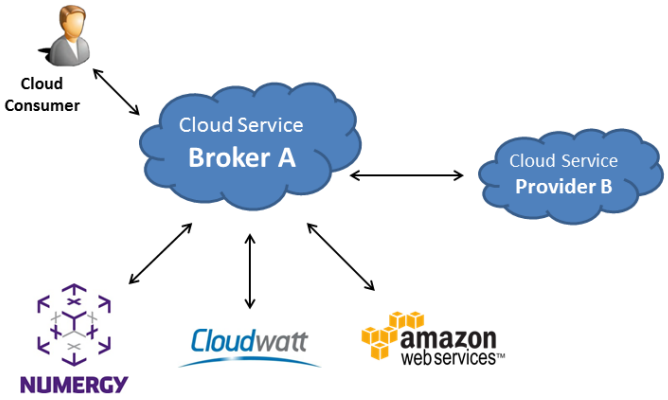

Figure 2.6 *Cloud brokerage*

As described in section 2.1.1, cloud resources could be delivered to consumers in 3 levels: SaaS, PaaS, and IaaS. Every day more cloud resource providers emerge to offer their resources as services to a large number of consumers. Although the great number of cloud providers means more resources in terms of both variety and quantity, it has its own challenges. In one hand, in order to request resources from a cloud provider, a consumer must learn the way of communicating and requesting those resources from that cloud provider. In other words, each cloud provider has its

own specific standards, interfaces, and APIs. Spending time to investigate how to use resources from each cloud provider is time-consuming and inefficient, resulting in vendor lock-in. On the other hand, a consumer always wants to have the best possible cloud providers in terms of cost, security, geographical location, etc. Finding the best offerings among enormous number of cloud providers is a real hard task, if not possible.

Vendor lock-in, finding the most appropriate cloud providers, and lots of other challenges are barriers on the way of cloud computing success. Most of these barriers could be handled if there was a way to interlink clouds of cloud providers. Cloud federation and cloud brokerage help cloud providers interlink their clouds [9]. The rest of this section explains these two terms in more details.

Cloud Federation

While cloud federation has been defined in different publications, certain commonalities among these different definitions cloud be noticed. The table 2.1 lists some of these definitions. All definitions agree that cloud federation relates to aggregation or sharing of cloud resources which can be accessed via each member of federation [9]. Extracting the most common elements of different definitions ends up in the following sentence which adopted by EASI CLOUDS project consortium (see subsection 2.2 for further information about EASI-CLOUDS):

> **Cloud federation** is the possibility for a cloud consumer to send a cloud request to multiple cloud providers as if they were a single cloud provider [10].

Cloud Brokerage

Table 2.2 presents some definitions of cloud brokerage retrieved from different sources. These definitions imply that cloud brokerage is an intermediary (person, organisation, technology) that bridges cloud consumers and cloud providers while offering value added services. On the other hand, according to Gartner, Cloud Service Broker (CSB) businesses are divided into three main categories [27]:

- Intermediation: A CSB takes services from cloud providers and adds value-added features (e.g. identity management, access management, etc.) to enhance and improve those services before delivering to cloud users.

No	Definition
1	Cloud federation manages consistency and access controls when two or more independent geographically distributed clouds share either authentication, files, computing resources, command and control, or access to storage resources [30].
2	Federated cloud is a cloud service model that connects (...) local infrastructure providers to each other, creating a global marketplace that enables each provider to buy and sell capacity on demand [1] .
3	A federated cloud (also called cloud federation) is the deployment and management of multiple external and internal cloud computing services to match business needs. A federation is the union of several smaller parts that perform a common action. [12]
4	Cloud federation refers to the unionization of software, infrastructure and platform services from disparate networks that can be accessed by a client via the internet. The federation of cloud resources is facilitated through network gateways that connect public or external clouds, private or internal clouds (owned by a single entity) and/or community clouds (owned by several cooperating entities); creating a hybrid cloud computing environment. [4]
5	Cloud federation comprises services from different providers aggregated in a single pool supporting three basic interoperability features ? resource migration, resource redundancy and combination of complementary resources resp. services [22].

Table 2.1 *Cloud federation definitions [9]*

- Aggregation: A CSB makes one or more new services by combining several services from different cloud service providers. Combining services are chosen statically.
- Arbitrage: Unlike aggregation in which fix services are integrated, service arbitrage gives flexibility to service aggregator to choose and combine services dynamically.

Generalising from these matters, two main points of view should be considered while describing cloud brokerage: from technical perspective or from organisational perspective. From technical perspective, I adopt the definition given by cloud computing reference architecture of National Institute of Standards and Technology [NIST]:

> A **cloud broker** is an entity that manages the use, performance and delivery of cloud services, and negotiates relationships between cloud providers and cloud consumers [24].

No	Definition
1	A cloud broker is a third-party individual or business that acts as an intermediary between the purchaser of a cloud computing service and the sellers of that service. In general, a broker is someone who acts as an intermediary between two or more parties during negotiations [28].
2	A cloud broker is a software application that facilitates the distribution of work between different cloud service providers. This type of cloud broker may also be called a cloud agent [28].
3	A cloud broker is a strategic mediator who performs a selection of cloud services for enterprises and consults the companies in this regard. Therefore, the cloud broker builds up contacts with respect to multiple cloud service providers, checks their services and selects out of the comprehensive service offering the platforms and services that support [the] customers cloud computing [activities] optimally. Cloud brokers are experts who assist the clients regarding the selection and integration of the services and applications and ensure a smooth transition between services from multiple cloud service providers [3].
4	A cloud broker is an individual or organization that consults, mediates and facilitates the selection of cloud computing solutions on behalf of an organization. A cloud broker serves as a third party between a cloud service provider and an organization buying the provider's products and solutions [19].
5	A cloud broker is defined as an entity (person or organization) that provides intermediary-type services between a cloud consumer and multiple cloud providers [8].
6	A second definition of cloud broker pertains to a new type of software that sits on top of cloud providers to abstract, simplify and map various cloud offerings to your environment. Cloud broker software assists organizations in creating solutions in the cloud, migrating solutions to the cloud and moving solutions between clouds [8].
7	An entity that manages the use, performance and delivery of cloud services, and negotiates relationships between Cloud Providers and Cloud Consumers [24].

Table 2.2 *Cloud brokerage definitions [9]*

From this perspective, cloud brokerage is an essential requirement of cloud federation; At least some participants of cloud federation should perform the technical function of a CSB, otherwise cloud federation could not be carried out [10].

From business perspective, depending on which functions participants are responsible for to carry out cloud federation, several business models emerge. These business models can be viewed as more or less "broker-driven" or more or less "federation-driven" [10].

EASI-CLOUDS follows business perspective and the business model EASI-CLOUDS platform applies is broker driven.

2.2 EASI-CLOUDS and CompatibleOne

2.2.1 Introduction

EASI-CLOUDS project stands for Extensible Architecture and Service Infrastructure for Cloud-Aware Software. It is part of EUREKA framework. It has 29 partners from 6 countries. Tampere University of Technology is one partner participating from Finland.

2.2.2 Objectives

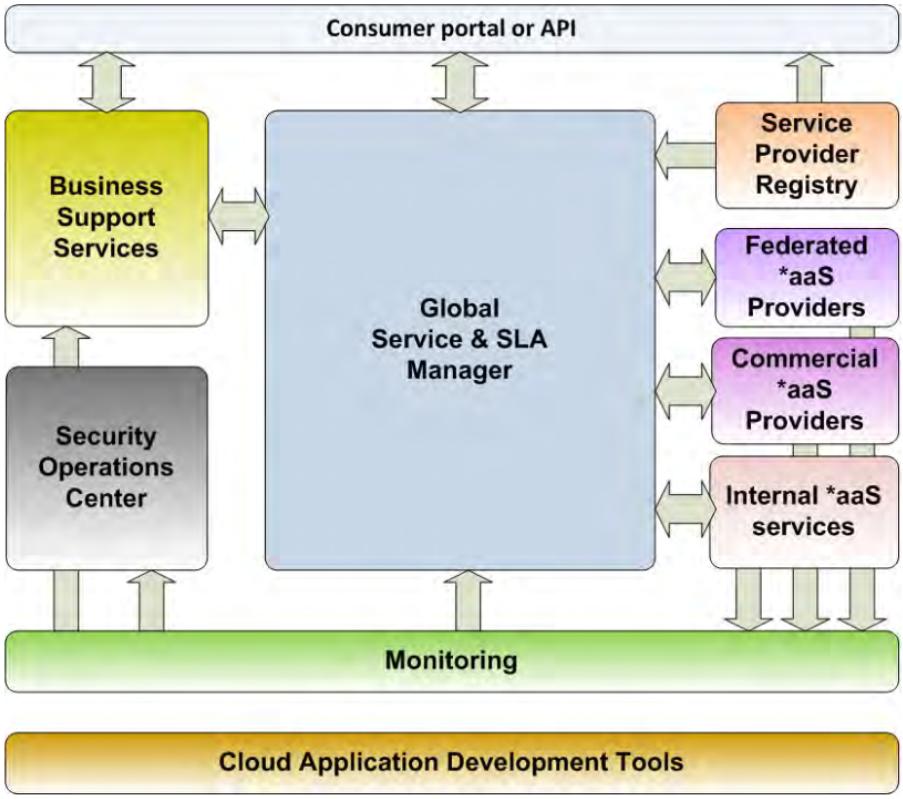

Figure 2.7 *EASI-CLOUDS High Level Architecture [13]*

The goal of EASI-CLOUDS project is improving cloud computing in Europe and the countries contributors are from. The major expected outcome of EASI-CLOUDS project is a comprehensive cloud computing platform that remove obstacles and

problems in all three main cloud computing categories: SaaS, PaaS and IaaS. High level architecture of EASI-CLOUDS is depicted in figure 2.7.

Users of EASI-CLOUDS platform can be divided into three groups: cloud consumers, cloud providers, and developers. Each of these groups has their own expectations. Hence, objectives of EASI-CLOUDS platform can be introduced according these different classes of users and their requirements [11]:

- **For cloud consumers:** EASI-CLOUDS facilitates adoption to multi-cloud architecture, instead of using one cloud provider. Working with multitude of cloud service providers simultaneously brings many advantages: security (e.g. confidential data is distributed across and processed in several providers), cost reduction (by moving data to lower price providers), risk reduction (service replication across multiple providers reduces complete failure at the same time), and etc. EASI-CLOUDS carries out this objective by providing tools to support heterogeneous providers. Consumers selects cloud services they need in form of a simple file named SLA (Service Level Agreements). SLA gives freedom of choice to cloud consumers regardless of which cloud providers are provisioning cloud services (Figure 2.8).
- **For cloud providers:** EASI-CLOUDS enables partnership between cloud providers through federation and brokerage (Figure 2.8). European market consists small providers comparing American giants. EASI-CLOUDS tries to help providers use each other's resources as simple as possible.
- **For developers:** Since developers can be considered as cloud consumers, previously mentioned objective for cloud consumers (as first item in this list) also hold true for developers. Moreover, EASI-CLOUDS provides tools to create cloud-aware applications. This is also thesis contribution to EASI-CLOUDS project (see subsection 2.2.3).

To reach its goals, EASI-CLOUDS does not reinvent the wheels. It uses CompatibleOne as the main component and plugs advanced features, capabilities, and components to it. CompatibleOne has done cloud federation and brokerage to some extent. It helps cloud consumers to select appropriate provider offers in terms of supported technologies, security, billing, etc and cloud providers to provision their resources to consumers. Due to this goal, some components like COAPS (subsection 2.2.4) and ACCORDS (subsection 2.2.5) have been developed in CompatibleOne project.

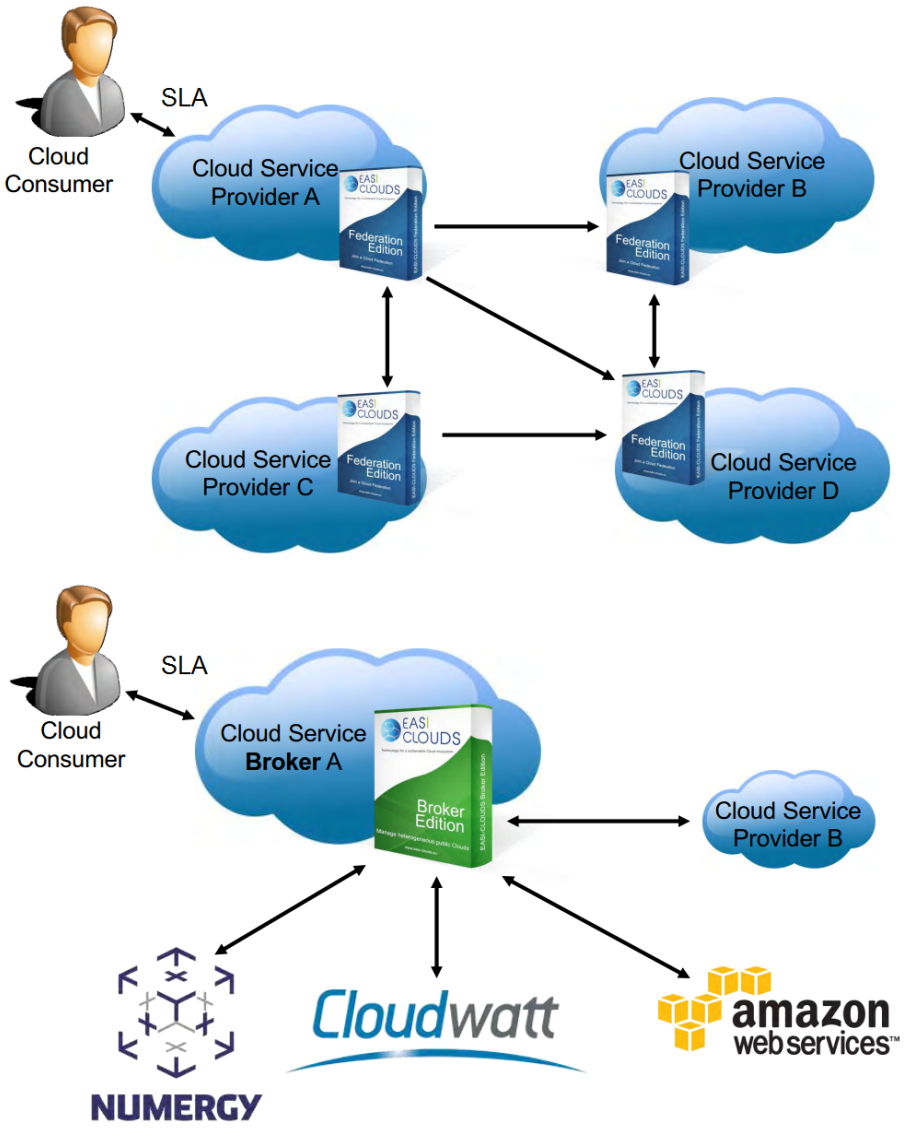

Figure 2.8 *Cloud federation and cloud brokerage in EASI-CLOUDS [13]*

2.2.3 Thesis Contribution in EASI-CLOUDS

The present thesis is part of EASI-CLOUDS project. Consequently, the objectives of this thesis are aligned with objectives of EASI-CLOUDS project (objectives are described in section 2.2.2). Some objectives have been followed by other partners participating in EASI-CLOUDS project. As the result, some tools and components have been developed. TUT, as a partner in this project, is responsible for investigating issues of software development for a federated cloud and provides a tool to perform it.

2.2.4 COAPS

In order to achieve cloud federation and cloud brokerage CompatibleOne developed some components. One of these components is COAPS which aims to facilitate provisioning PaaS resources through making PaaS providers interoperable.

Different PaaS providers use different PaaS solutions (CloudFoundry, Openshift[5], etc.) which have their own specfic APIs. No standardised interface is followed by all or at least some PaaS providers. If a consumer using one PaaS wants to change his provider, he encounters vender lock-in problem. Moreover, the interactions and communications between PaaS providers itself could be problematic, resulting in their isolation. Therefore, there is a need for a PaaS independent solution. COAPS [29] offers a solution to federate available PaaS providers instead of isolating them (Figure 2.9).

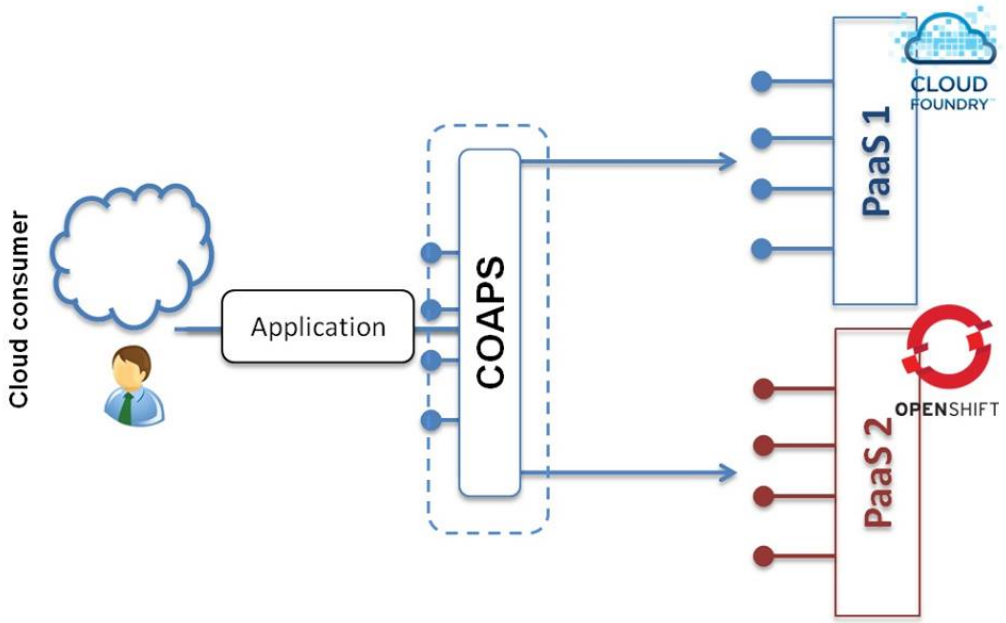

Figure 2.9 *COAPS federates existing PaaS providers [5]*

COAPS proposes the solution by the separation of 2 main concepts. Firstly, it describes PaaS resources as a unified representation model (called PaaS resources and Application Description model) regardless of which PaaS provider will provision those resources. Secondly, it offers PaaS application provisioning and management API (called COAPS API). By using COAPS API, consumers can manage and provision PaaS resources whatever PaaS provider is selected [5]. Figure 2.10 shows all COAPS APIs required to manage PaaS resources of a PaaS provider.

[5] `https://www.openshift.com`

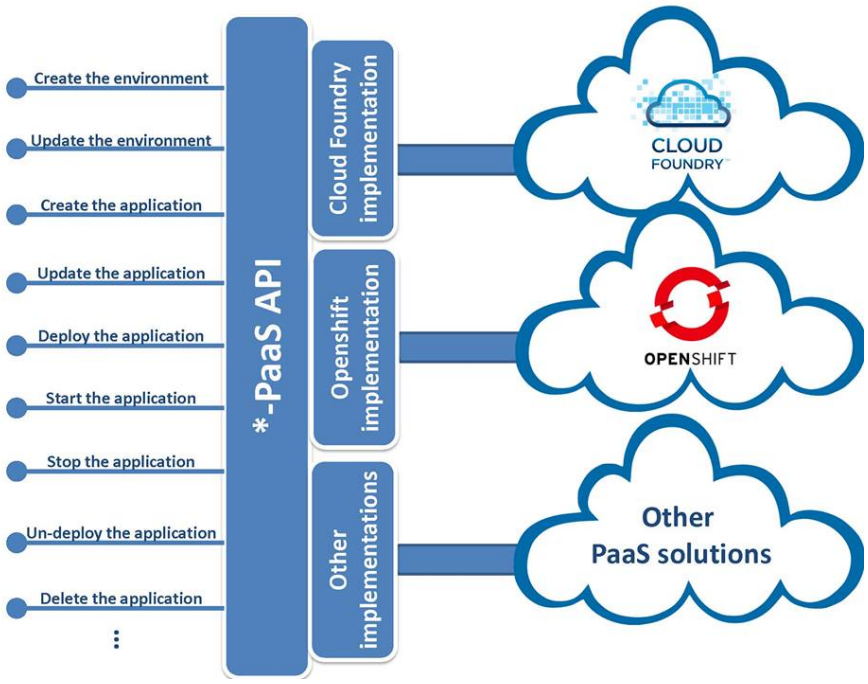

Figure 2.10 *COAPS PaaS resource provisioning and management API [32]*

In COAPS point of view, PaaS resources are divided into two different kinds: **Applications** to deploy and **Environments** to host applications. Environment resource defines set of configuration options needed in the PaaS to host and run application, including runtimes (Java7, ruby, etc.), frameworks/containers (spring, tomcat, ruby, etc.) and services (databases, messaging, etc.). Application resource specifies attributes of application itself including version, number of instances, source archive type, etc. For each resource type there are a description model and several management methods. Application and environment management methods are listed in figure 2.11 .

To get the idea how COAPS performs application management and provisioning, figure 2.12 shows a basic process to simply deploy and run an application through COAPS. As the first step, specification of environment which is going to host the application should be determined in the format of COAPS description model for environment resource. This environment manifest will be sent to Create Environment method. A unique ID will be assigned to the newly created environment and will be send back. Then application specifications in the form of application manifest is sent to Create Application method. Similarly a unique App ID is received. As the third step, Deploy Application method deploys the created application to the created environment by referring to their IDs. Finally Start Application method runs the deployed application.

Application management operations	
Operation	**Command**
Create Application	POST /app
Update Application	POST /app/{appId}/update
Find Applications	GET /app
Start Application	POST /app/{appId}/start
Stop Application	POST /app/{appId}/stop
Restart Application	POST /app/{appId}/restart
Describe Application	GET /app/{appId}
Destroy Application	DELETE /app/{appId}
Destroy Applications	DELETE /app/delete
Deploy Application	POST /app/{appId}/action/deploy/env/{envId}
Undeploy Application	POST /app/{appId}/action/undeploy/env/{envId}
Environment management operations	
Operation	**Command**
Create Environment	POST /environment
Update Environment	POST /environment/{envId}/update
Destroy Environment	DELETE /environment/{envId}
Find Environments	GET /environment
Describe Environment	GET /environment/{envId}
Get Deployed Applications	GET /environment/{envId}/app
Get information	GET /environment/info

Figure 2.11 *COAPS application and environment management methods [5]*

[32] explains in details about PaaS resource types in COAPS, their description model, and their management methods.

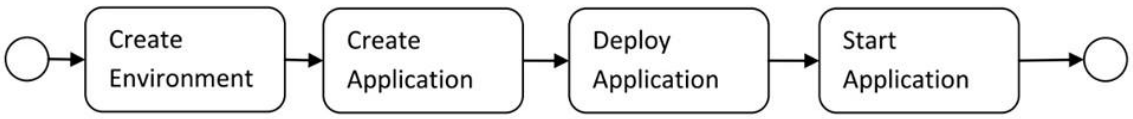

Figure 2.12 *Basic application deployment process through COAPS API [32]*

2.2.5 CompatibleOne

As cloud computing grows dramatically, the number of cloud service providers increase rapidly. Cloud consumers therefore encounter a challenging task of finding proper cloud providers that meet their needs in terms of cost, security, offered services, etc. The advent of cloud brokers helped cloud end users find decent cloud providers.

CompatibleOne[6] is an open source cloud broker. It meets all three service categories defined by NIST. As one proof of meeting intermediation, CompatibleOne defines who can access what by Service Level Agreements (SLA) negotiation. Meeting aggregation and arbitrage, cloud users can combine different cloud services provisioned by different cloud providers through CompatibleOne. That is the result of the method CompatibleOne applies for provisioning cloud providers' services; This method is based on open standards, mainly Cloud Data Management Interfaces (CDMI), Open Cloud Computing Interface (OCCI)[7] and a new feature called CORDS (CompatibleOne Resource Description System) which is itself based on OCCI.

CompatibleOne has a model and an execution platform. The model is called CORDS and the execution platform is called ACCORDS.

CORDS

CORDS is an object-based description model for cloud resources. It is designed based on OCCI open standard and makes CompatibleOne an interoperable middleware that helps cloud end users switch between cloud service providers. CORDS is the core concept behind the description and federation of heterogeneous cloud resources offered by different cloud providers. These resources could be in different levels (IaaS and PsaS). In figure 2.13, CORDS manifest that describes PaaS resource is depicted. This PaaS CORDS manifest is similar to the combination of COAPS environment manifest and application manifest. Later in this section the reason behind this similarity will be discovered. As stated in section 2.2.4, to see XML schema of COAPS application and environment manifest refer to [32].

ACCORDS

ACCORDS, Advanced Capabilities for CORDS, is the execution platform of CompatibleOne. ACCORDS uses CORDS to model and manage cloud resources.

ACCORDS launches several components and services (e.g. Publisher, Parser, Broker, etc.) during starts-up. Each component is an OCCI server that can communicate with other components through HTTP requests. First of all, Publisher is started and then others register themselves to Publisher in a certain order. These

[6] http://www.compatibleone.com/community/conferences/
[7] http://occi-wg.org

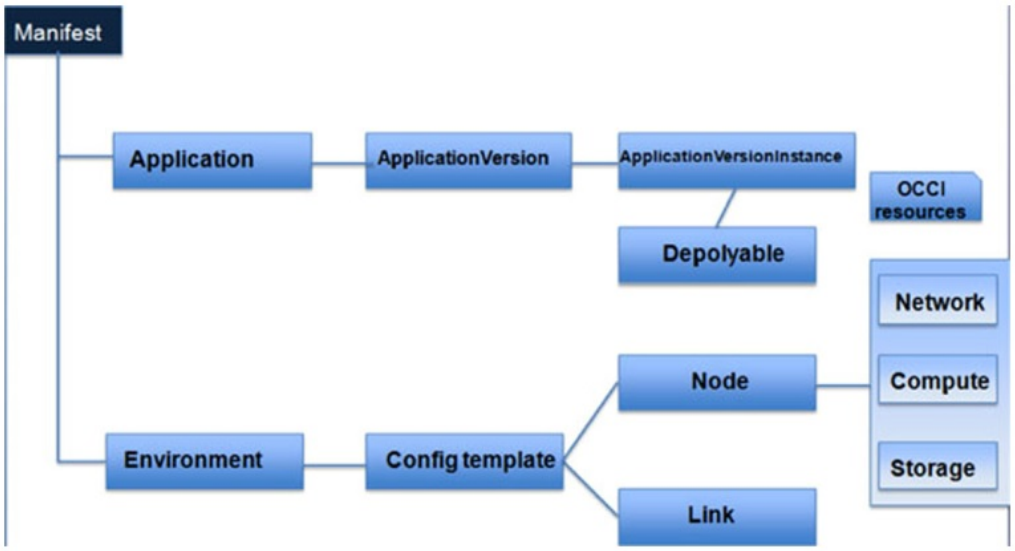

Figure 2.13 *ACCORDS PaaS Logical Data Model*

published services specify categories (of services) ACCORDS are able to provision.

As shown in figure 2.14 cloud resource provisioning is done in 4 steps, each of which briefly described as following:

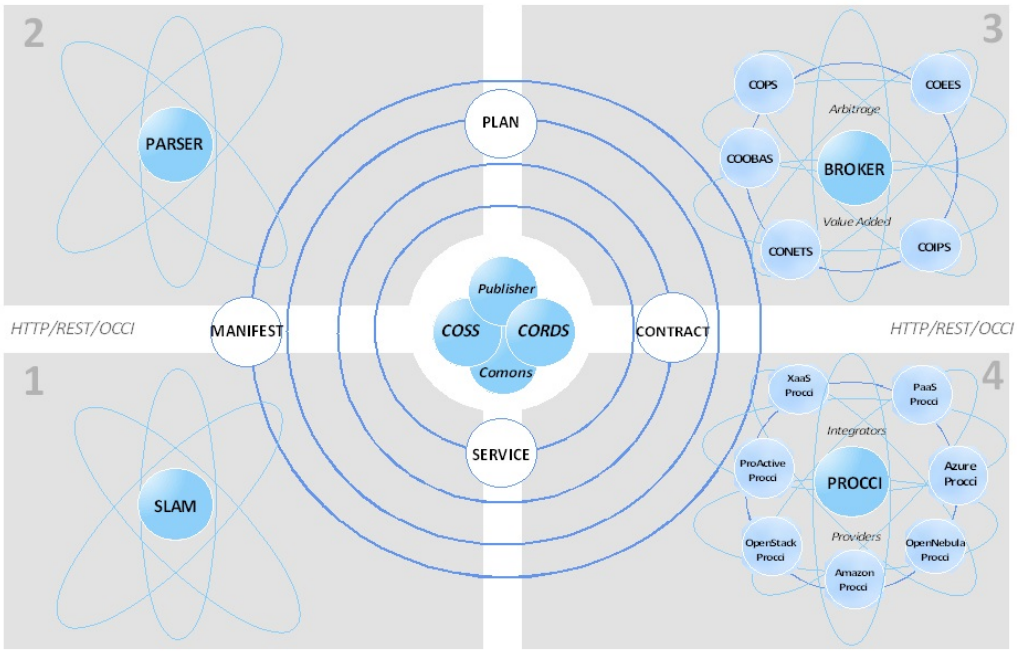

Figure 2.14 *ACCORDS Platform Architecture*

1. Handling the user's requirements. Consumer specifies the needed (IaaS or PaaS) resources and turns those into (PaaS or PaaS) CORDS manifest using

an SLA template production tool.

2. Validation and provisioning plan. CORDS manifest are parsed into provisioning plan by Parser service. Provisioning plan contains blueprints for services requested by consumers. Moreover, parser checks validation of requested services though communication with Publisher.

3. Execution of the provisioning plan. Broker chooses proper cloud providers based on requirements and constrains specified in provisioning plan. There are also services that their participation depends on requirements stated in SLA. Finally, broker aggregates selected services and collects them in a contact.

4. Delivering the Cloud services. Finally, Broker sends contract to proper ACCORDS Procci (a proxy which is heavily designed based on OCCI categories is called Procci) to carry on the actual provisioning of correct requested resources provided by cloud providers.

ACCORDS and COAPS

This theses is more interested in PaaS (not IaaS) provisioning aspect of ACCORDS, that is why only PaaS CORDS manifest schema is presented (figure 2.13).

Each box (element) of CORDS manifest represents a cloud resource that resides in an ACCORDS category. In other words, during second phase of CompatibleOne functional cycle, Parser goes through all elements of PaaS CORDS manifest (e.g. application, element, etc.), asks Publisher to correspond each element to a category and instantiate related OCCI servers (ACCORDS services), adds OCCI server URIs to each element to creates provisioning plan. As you can see, provisioning plan structure is the same as that of CORDS manifest except that for each element, URI of the related ACCORDS service instance is added.

During the third phase, Broker converts each element of provisioning plan into a contract. These contracts will be sent to ACCORDS PaaS Procci. For each contract, PaaS Procci communicates with a specified PaaS Provider (CloudFoundry, Openshift, etc.) through CompatibleOne Application Platform Service (COAPS). Here, as it is supposed, COAPS acts as a middleware (abstraction layer) between CompatibleOne ACCORDS platform and existing PaaS Providers (Figure 2.15).

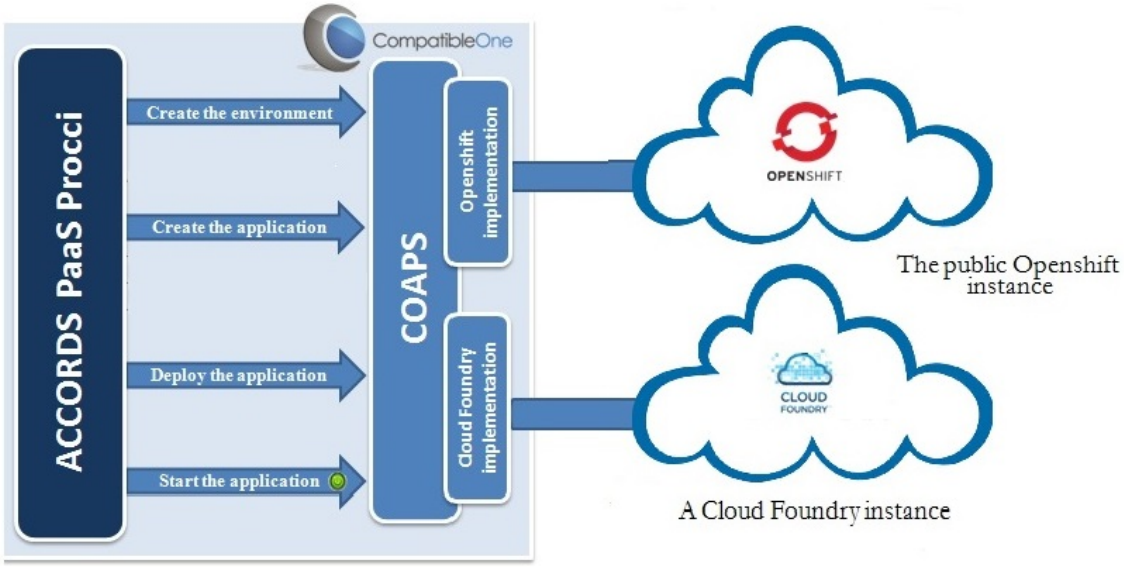

Figure 2.15 *ACCORDS Application PaaS Resource Provisioning*

2.3 Software Development Automation Practices

2.3.1 Continuous Integration (CI)

Suppose a software project in which programmers are coding individually. After a long time, when it is the time to make the deployable artifact and send it to operation team, their codes must be merged and built. These codes are written in isolation for a long period, therefore integrating them will result in lots of conflicts and errors. Under these circumstances, integration is a long, error-prone, and unpredictable process.

Continuous integration is a software development practice in which all project members merge and verify their codes frequently, at least once a day. Subsequently, each developer is coding at the most for a few hours without merging her codes to the shared project state and when it turns to integration, spending a few minutes is enough. Although continuous integration practice does not require any special tool, it is beneficial to verify integration by build automation (including test) to identify integration errors rapidly. The more integration period decreases, the less errors rooting from integration could be found. Hence, integration could be carried out as fast as possible [14].

According to Fowler [14], there are some elements to perform CI as well as possible:

- **Single source repository.** A software project contains many files. In one hand, these files are distributed among project members. On the other hand, all files are essential while building different versions of product. Therefore, there is a need for a tool to gather these distributed files and keep track of their changes. Many tools (e.g. Git, Subversion, etc.) are available to use. Regardless of which tool is chosen, one important note is that all project files including (test scripts, properties files, database schema, install scripts, etc.) must be in the shared repository.

- **Build Automation.** Getting project files and turning them into a running software is a complicated and hard task. Automation of this process could help a developer change some files and launch the final running software by execution of a simple script. Many build automation tools (e.g. Ant, Maven, etc.) are available. As the result of this variety, build scripts come in different forms. So it is important to have a master build on development servers that can be triggered from other scripts. Although it is acceptable that developers use their IDE build management methods or any other build automation tool while developing locally on their own machines, it is highly recommended that developers utilise the same build automation tool which is used on the development server.

- **Making your build self-testing.** Having a running software does not guarantee absence of bugs. Although testing does not remove all bugs, automated tests could help a lot to catch many bugs. Automated tests investigate the code base and announce the result as failure if there are bugs. Every developer should commit to mainline, the unique and main path of development, at least every day. There are some steps that make a developer eligible to commit to the mainline. First updating the local working copy of the project by merging it with the mainline, resolving all conflicts, and doing the build. If the result of this local build (including tests) is successful, developer could commit to the shared state single source repository.

- **Building mainline on integration server after every commit.** Doing update and build on developers' local machine before commit is not enough, committing changes to mainline may cause conflicts and errors on integration server. Therefore, regular builds should be done on integration server to confirm the quality of the mainline of the code. The developer who commits changes is also responsible to monitor integration server build and further corrections in the case of failure. It is really important to fix mainline as soon as possible in the case of build break.

- **Keeping the build fast.** Since one of the most important features of CI is providing rapid feedback on errors and conflicts, build on integration server must not take a long time. In order to reduce build time, first we need to know about deployment pipeline (aka build pipeline or stage build). Deployment pipeline consists of several builds done sequentially. These builds could be divided into two main stages: build that can be carried out quickly and builds that are slower. First-stage build is triggered when developer commits changes to mainline. Hence it is called 'commit build'. During commit build, compilation and some rapid tests (like unit tests) are performed. In other words, making commit build rapid is done by removing time-consuming tests enough to balance speed and the needs of bug finding. Second-stage builds are further, slower tests (like every test involving real database) that can also be done on other machines. One important note is that the first-stage (commit) build is supposed to be in the main CI cycle; Failure of this stage will result in failure of integration. In contrast, since second-stage build takes the executable from the last healthy commit build and run some tests on it, failure of this stage does not mean integration failure; it alarms the team to fix bugs as rapidly as possible.

Now that we know elements of best practices for CI, steps to show how to perform CI could be stated [7]:

1. Developers check out the code base into their local working place.
2. Developers edit code and then commit changes to mainline.
3. When changes happen to mainline, CI server check out these changes.
4. CI server builds the product and runs unit tests.
5. If the result of build and tests is successful, CI server makes the deployable artifact for further tests and notifies team about the success.
6. If the result of build and tests is not successful, CI server informs team about failure for rapid fix.
7. The process of integration, building, and testing continues as development goes on.

Figure 2.16 depicts the simplified version of steps to perform CI.

2.3.2 Continuous Delivery (CD)

As a development and delivery methodology, continuous delivery keeps application in a release ready state at all times during software development [6]. A key pattern

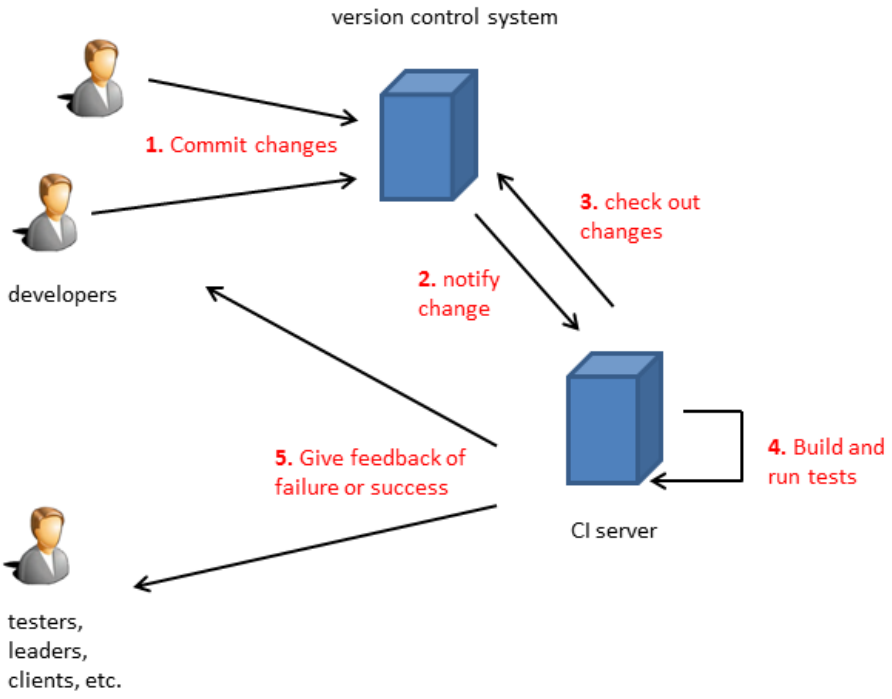

Figure 2.16 *Continuous integration cycle*

to carry CD out is deployment pipeline. As described in subsection 2.3.1 (read the last element of CI best practices, 'keeping build fast'), deployment pipeline is the collection of multiple sequential builds that could be divided into two stages. First-stage build or commit build is considered as a main element of CI cycle. Hence, continuous integration is an inevitable feature of deployment pipeline. But CI is not enough to carry out CD. CI mainly concentrates on development team. It produces executable for further tests and the rest of release process. But these two processes (testing and operations) are the most time-consuming ones in software development cycle [17]. For example:

- Operation team waits for fixes to be done.
- Test team members wait for successful commit builds.
- Development team gets notification of issues from test team members or operation team late.

In one hand, these issues makes application's way to reach production environment a long process, resulting in an undeployable application. On the other hand, long feedback cycle among the development team and the testing and operation team leads to a software with many bugs [17].

By automating build, test, and release processes, an end-to-end solution could be implemented in order to deliver software fast and safely. Multiple environments are needed, one to run commit builds, one or more to run tests (test environment), one or more to deploy into staging (staging environment), and finally one or more to deploy into production (production environment). CI server should move executable which is output of CI (commit build) to these environments and triggers their associated (CI server) builds successively and automatically (except production environment) based on feedback of build done in previous environment. This process is what is called deployment pipeline. Figure 2.17 depicts CD steps and shows deployment pipeline concept. With this strategy, two important goals are achieved: speed and safety. Team members (developers, testers, etc.) could see which build is in which stage by checking feedback of each step visible to all, resulting in rapid actions to issue fixing in case of any failure. [17].

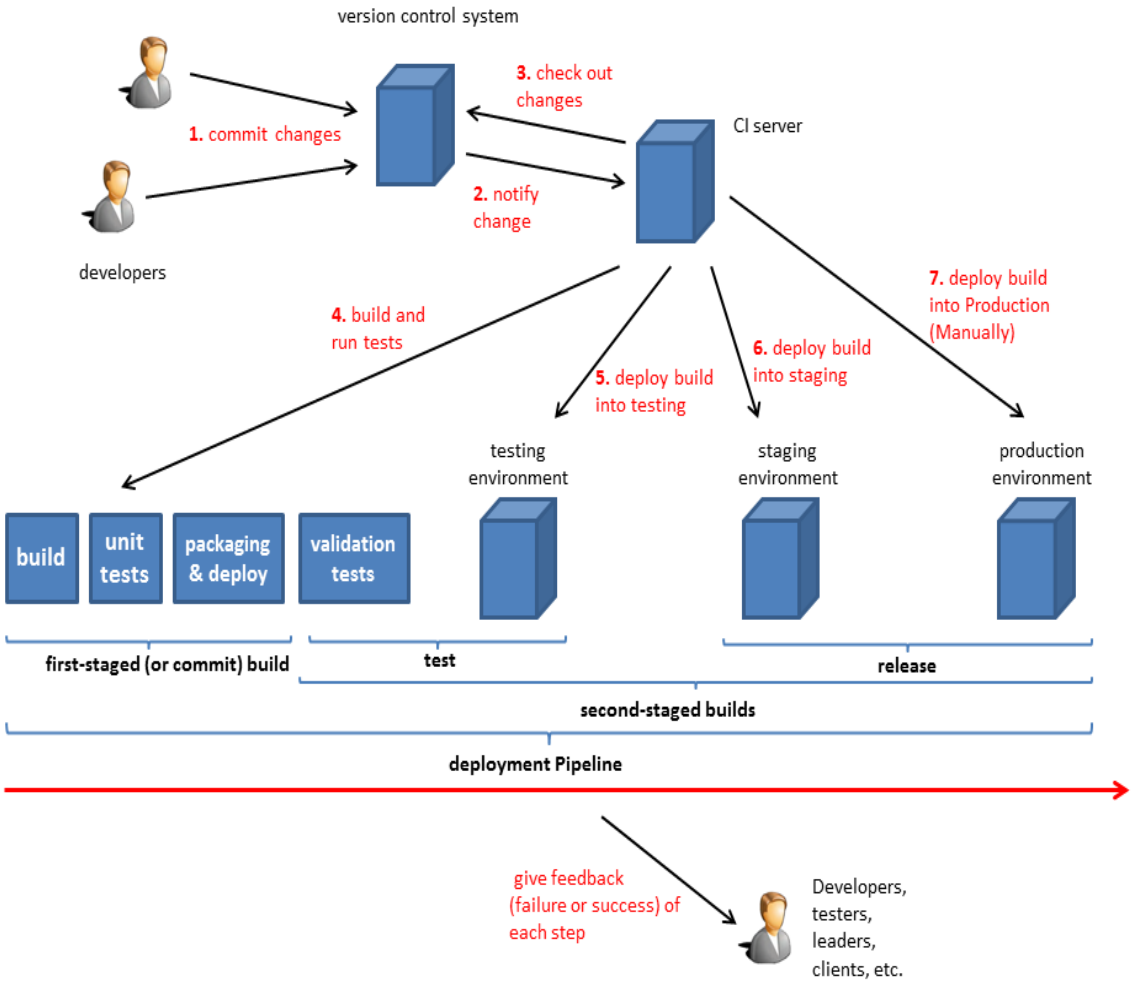

Figure 2.17 *Continuous Delivery cycle [20]*

The following scenario represents a usual CD workflow [20]. The present CD Workflow is the continuation of the CI workflow illustrated on subsection 2.3.1, that is why it starts from number seven.

7. After packaging files into executable, the CI server deploys it into test server to validate package and basic functionality of system by running automated acceptance tests.
8. The CI server deploys the same package into testing environment for further, comprehensive UAT (User Acceptance Test).
9. The CI server deploys the same package into staging environment.
10. The CI server deploys the same package into production environment (by allowance of operation teams and not automatically).

2.3.3 Continuous Deployment

Understanding the difference between continuous delivery and continuous deployment (figure 2.18) could help a lot to figure out continuous deployment. While as the result of CD practice, software is ready for release at all times during development, it does not actually mean automatic releasing. Software is only in a release ready state (in staging environment) and can be deployed into production by simply pressing a button manually at any time. Now the question is: what is continuous deployment? Continuous deployment is the practice of deploying every build that passes CD successfully into production. It means releasing every change to actual users automatically [16].

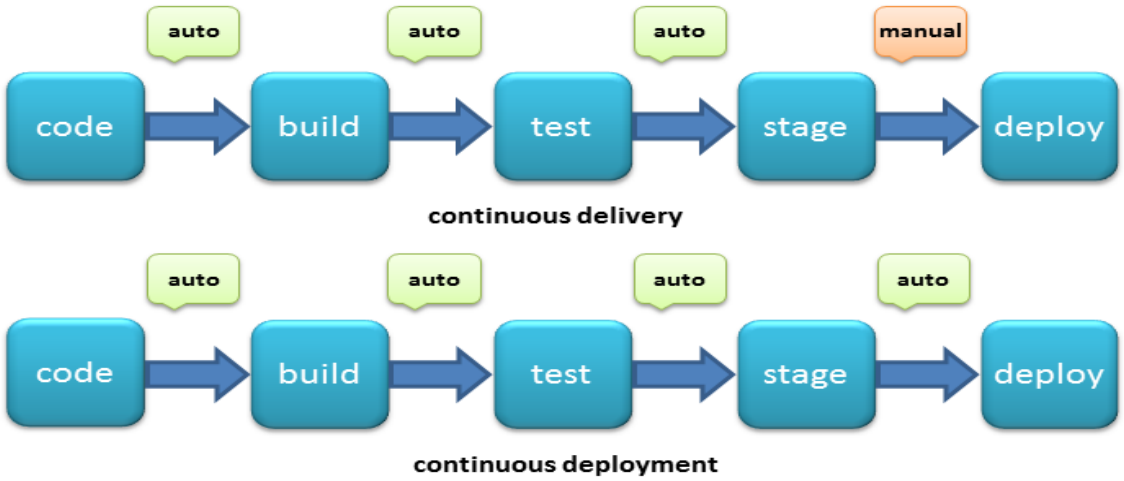

Figure 2.18 *continuous delivery vs continuous deployment [6][2]*

3. IMPLEMENTATION

The role of the present thesis in EASI-CLOUDS project (see subsection 2.2.3 'Thesis Contribution') is investigating issues of cloud-based software development for a brokered/federated cloud and providing a tool to perform it. The rest of this chapter is structured as follows: section 3.1 provides the research method and materials used to carry out the cloud-based SW development solution for a brokered/federated cloud. Section 3.2 describes two different use cases to develop the SW development tool using research methods and materials mentioned in section 3.1 'Research methodology and materials'.

3.1 Research methodology and materials

As stated already, mission of thesis is to develop cloud-based SW development tool for a brokered/federated cloud and to offer it as a cloud service (SaaS). The starting point of development of such a tool has been CoRED[1] which is a Collaborative Real-time Editor developed in Tampere University of Technology. It allows several programmers write code simultaneously while communicating and collaborating with each other [23]. Later generations of CoRED has been named as MIDEaaS, stands for Mobile IDE as a Service. In EASI-CLOUDS projects MIDEaaS has been extended for deployment (besides development) of applications. Flexible architecture of MIDEaaS facilitates adding plugin for further development (Figure 3.1). An interface (MideaasEditorPlugin) is defined in MIDEaaS as figure 3.1 shows. This interface is an extension point; implementing MideaasEditorPlugin will add an item to MIDEaaS menu bar (Figure 3.2).

Now that the IDE is selected, the second step is investigating an end-to-end solution for SW development cycle and integrating this solution into MIDEaaS as a plugin. Although different SW development models (e.g. Water fall model, V-model, Agile model, etc.) follow different phases in their life cycle, a general model includes the most common phases:

[1] `http://cored.cs.tut.fi`

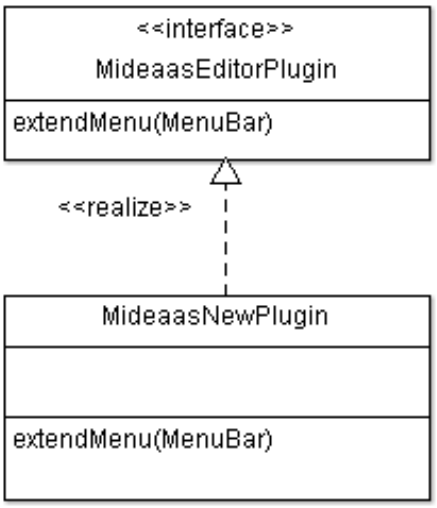

Figure 3.1 *MideaaS architecture facilitates adding new plugins*

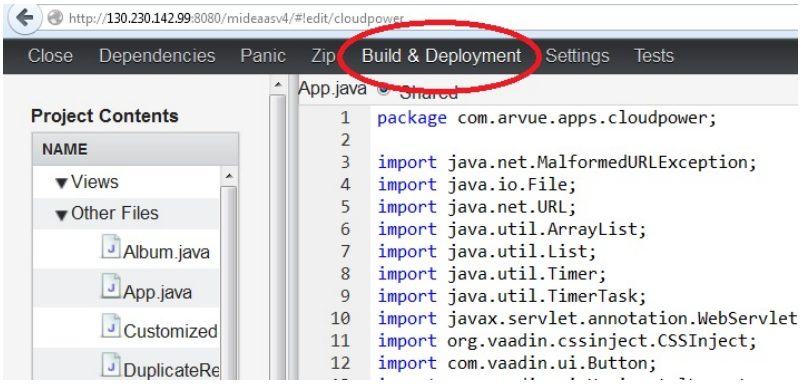

Figure 3.2 *Item added to menu bar of Editor View for continuous delivery*

- Requirements
- Design
- Implementation (Coding)
- Testing
- Release

One approach which recently gain traction is automating the phases of SW development from build to testing to release. One SW development automation practice is continuous integration (section 2.3.1 'Continuous Integration'). In continuous integration, codes written by project members are merged and verified frequently, at least once a day. In order to perform CI, there need to be some elements:

- Single source repository: This is a shared repository where all project files provided by project members are gathered together. One important note is

that before committing changes to mainline, team members should update and build on their local machines. But writing code in MIDEaaS frees developers of performing this task. MIDEaaS is a real time editor; it means all changes made by one developer will be applied and shown immediately to all other developers (like Google Docs[2]). Developers therefore have the last and updated version of the project at every single moment. The present thesis chooses Git[3] as its Source Control Management. Git is fast, able to handle large projects, open source, simple, and fully distributed.

- Build automation: By build automation project members can execute a simple script and automate phases of software development including compilation, running tests, and packaging. Built automation tool utilised in this thesis is Maven[4].

- Continuous integration server: After every commit to mainline, there can be conflict and errors during integration. Main task of CI server is to build mainline after committing every change to ensure that there are no conflicts and errors during integration. Based on EASI-CLOUDS' decision Jenkins[5] is selected as CI server for our demonstrator. It is written in Java and originated form Hudson[6] project but later was forked after a dispute with Oracle[7]. Jenkins is known to be the dominant CI server as shown in figure 3.3. In order to perform different tasks, Jenkins has a concept called 'job'. A job is a runnable task that is controlled by Jenkins and goes through different phases in its life cycle. For example lifecycle of a job which is configured to poll and check out source code from Source Control Management (SCM) and build it is depicted in Figure 3.4. For each newly created project in MIDEaaS, a job will be instantly created in Jenkins and configured to check out the project remote Git repository. After editing project source code in MIDEaaS, user pushes changes to remote Git repository. Git receives pushed commit and then triggers associated Jenkins job to poll, check out, and build the project.

CI performs frequent integrations, builds the executable, and runs some simple tests like unit tests. It therefore does not cover testing and release phase of SW development cycle. But our goal is to introduce an end-to-end solution that automates the whole lifecycle of SW development. Due to this aim, continuous delivery (see

[2] https://docs.google.com/
[3] http://git-scm.com
[4] http://maven.apache.org
[5] http://jenkins-ci.org
[6] http://hudson-ci.org
[7] http://www.oracle.com/

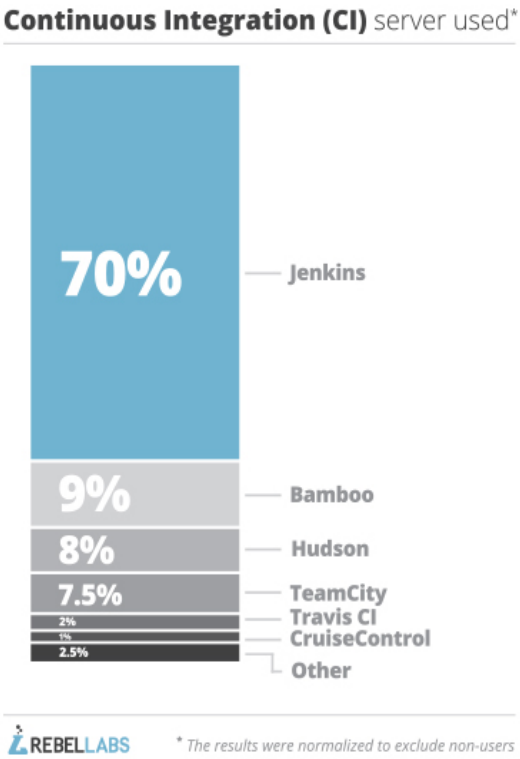

Figure 3.3 *CI server used [21]*

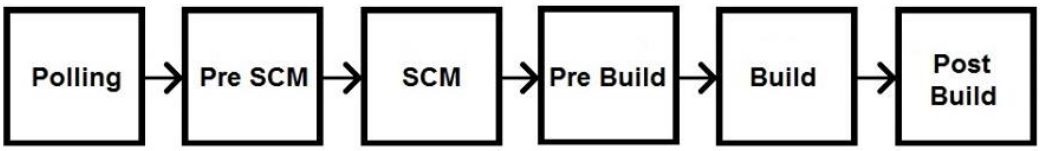

Figure 3.4 *Jenkins Job Life Cycle*

subsection 2.3.2 'continuous delivery') is chosen. CD automates build, testing, and deployment into staging environment. Note that testing and deployment into staging are not needed in our demonstration; that is why these steps are not realised in the current implementation of our demonstrator. On the other hand, according to CD practice, deployment into production should be done by operation team manually, not automatically. But in our demonstration Jenkins deploys WAR file into production automatically after commit stage. Actually, the automation practice of our SW development solution is continuous deployment (see section 2.3.3 'Continuous Deployment'). Figure 3.5 depicts continuous deployment workflow used in this thesis.

Now a MIDEaaS user is able to develop software and deploy it into production en-

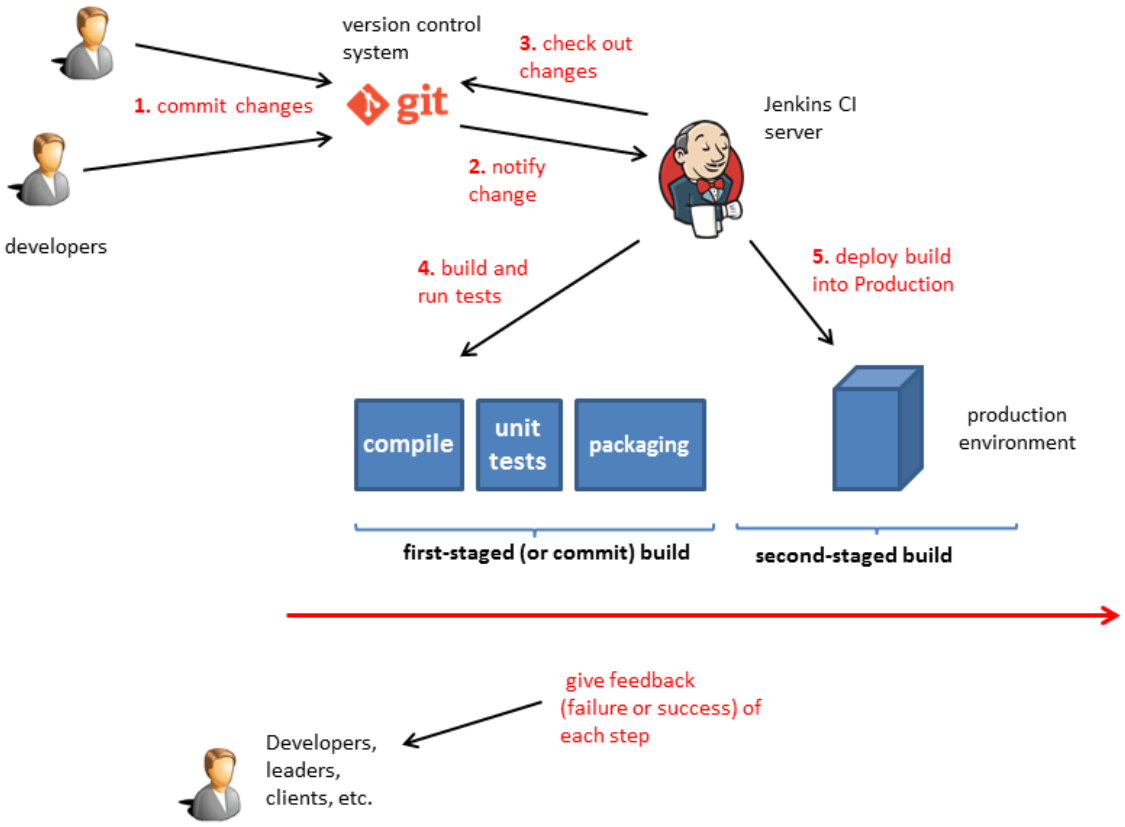

Figure 3.5 *continuous deployment scenario utilised in the present thesis*

vironment. But our goal is to perform deployment into a brokered/federated cloud, not into a predefined production environment or a specific PaaS provider. There are many PaaS providers offering their resources. Among these providers, a developer wants to find the best ones in terms of security, geographical location, cost, memory, storage, supported services (like databases) and so on. There should be a mechanism that gets the list of resources and specifications the developer expects from a PaaS provider. In MIDEaaS, developers are given a tool which is called 'Resource Specifier' to specify their required PaaS resources. Resource specifier is developed by a Finnish company named Lenonidas[8]. Figure 3.6 shows a snapshot of the resource specifier. The next step is to find the most appropriate PaaS provider based on the required resources. This means cloud brokerage; therefore EASI-CLOUDS platform (section 2.2 'EASI-CLOUDS and CompatibleOne') plays the role of a cloud broker. In our demonstration, depending on which use case is followed (see section 3.2 'Use Cases'), MIDEaaS or Jenkins interacts with ACCORDS (see subsection 2.2.5 'ACCORDS') to carry out brokerage to find the most proper PaaS provider. After choosing the target PaaS provider, it turns to PaaS resource provisioning and

[8] `https://leonidasoy.fi`

then actual deployment. For PaaS resource provisioning, we need communication with the target PaaS provider; COAPS (see subsection 2.2.4, 'COAPS') offers a PaaS independent solution. It frees users from thinking about which PaaS solution (e.g. CloudFoundry) is applied by the PaaS provider to provision its resources. ACCORDS or MIDEaaS therefore (again depending on which use case is followed) needs to talk to COAPS. For our demonstration, CloudFoundry, COAPS and ACCORDS must be installed:

- The unofficial one machine Cloudfoundry installer we used is called CF Nise installer. One instance of CloudFoundry therefore is installed on a VM at TUT.
- The CloudFoundry implementation of COAPS API is installed on a VM at TUT and configured to talk to REST APIs of TUT CloudFoundry instance.
- ACCORDS is installed on a VM at TUT and configured to talk to REST APIs of TUT COAPS instance.

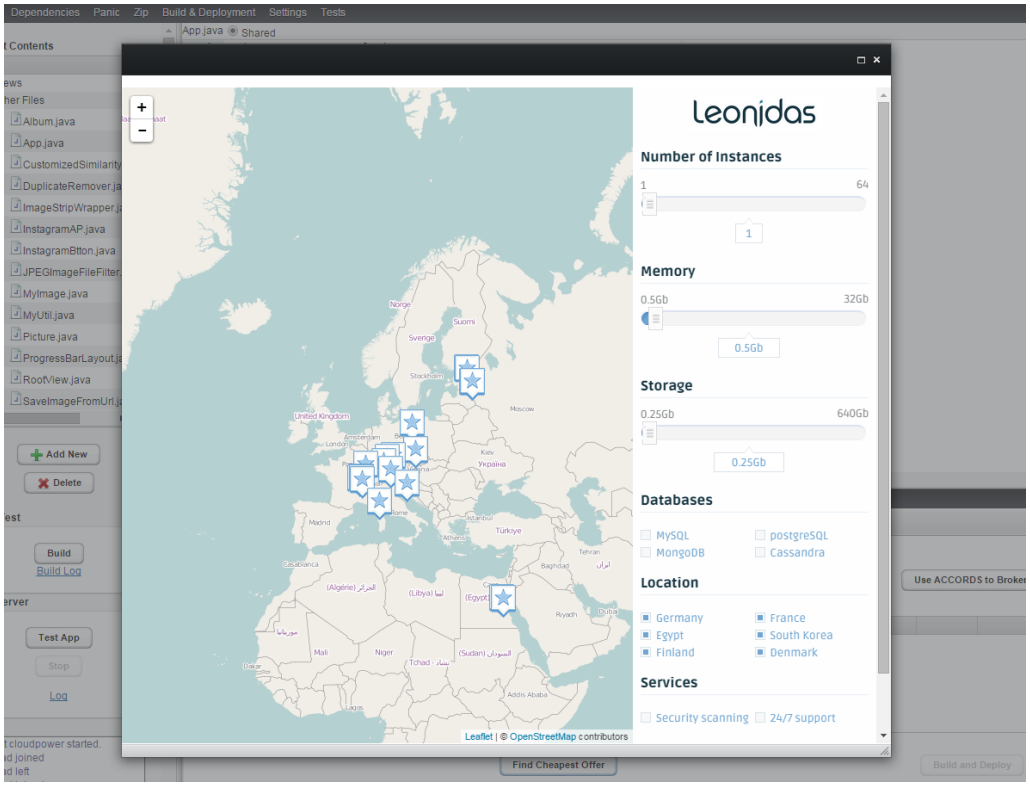

Figure 3.6 *A developer specifies required PaaS resources via Resource Specifier*

As stated before, depending on which use case is followed, Jenkins needs to communicate with ACCORDS or COAPS. Jenkins job could be extended in different phases of its life cycle (Figure 3.4) by implementing extension points related to that

phase. Implementing extension points enables writing plugins for Jenkins. In our demonstration communication with ACCORDS or COAPS must happen after build is successfully done, so extension point (Notifier[9]) related to the 'post build' phase must be implemented. Two Jenkins plugins have been developed:

- Jenkins COAPS plugin has been developed to communicate with heterogeneous PaaS providers.
- Jenkins ACCORDS plugin has been developed to communicate with ACCORDS plugin to do brokerage.

Figure 3.7 shows all required components to perform cloud-based SW development solution for a brokered/federated cloud offered in this thesis. But how these components works together? Next section presents two different use cases followed in this thesis to perform continuous deployment.

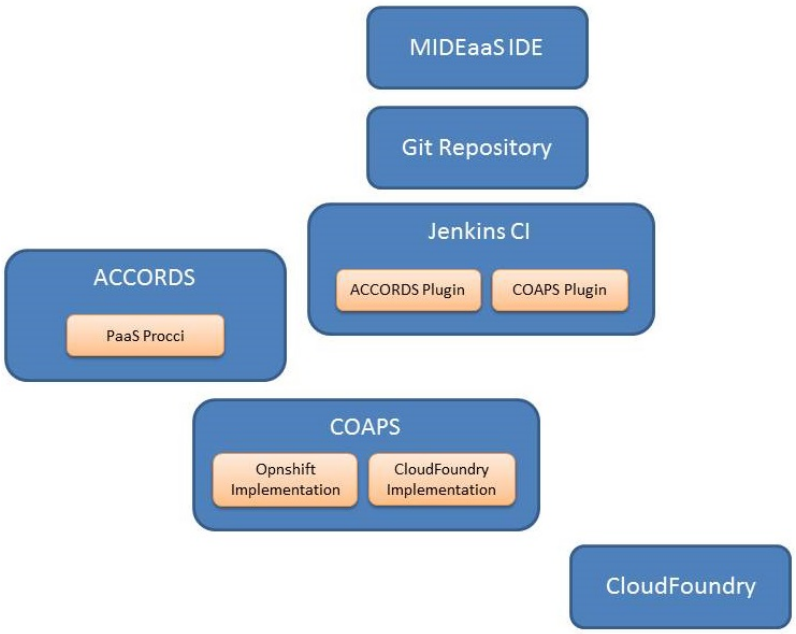

Figure 3.7 *All required components to carry out continuous deployment offered in this thesis*

3.2 Use Cases

In section 3.1 'Research Method and Materials', required components to perform continuous deployment are stated. These components can communicate with each

[9] `http://javadoc.jenkins-ci.org/hudson/tasks/Notifier.html`

others in two different ways resulting in two different scenarios. This difference has its roots in the way ACCORDS follows to perform deployment. The rest of this section presents two use cases to perform continuous deployment.

3.2.1 First Use Case

First use case is based on 'deferred deployment' method of ACCORDS. Deferred deployment means ACCORDS first prepares environment on the target PaaS to host and run application. Later, the actual deployment could be done in any time. To get the idea better, take a look at figure 3.8. In Figure 3.8, ACCORDS sends four requests to COAPS for completing deployment chain. In deferred deployment, as the first step, ACCORDS sends the first two requests (create environment and create application) in order to specify different characteristics of the application's environment and application itself. Later, when it comes to deployment, developer sends the second two requests (deploy application and start application) via MIDEaaS directly to COAPS in order to deploy and start application. Continuous deployment blueprint utilised in this use case is depicted in figure 3.9.

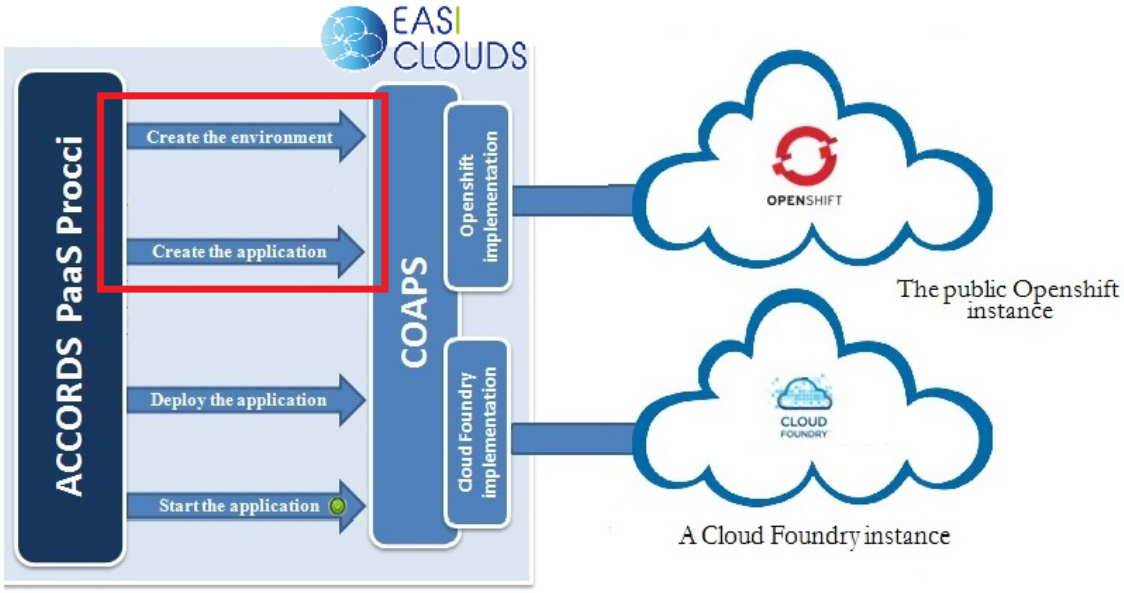

Figure 3.8 *ACCORDS calls two specific COAPS API for PaaS resource provisioning*

Exact details of deployment process from editing code in MIDEaaS to completing deployment on the target PaaS are investigated in sequence as following:

1. User creates a project in MIDEaaS and writes source code to develop an application.

First Use Case - Continuous Deployment Blueprint						
EASI-CLOUDS Platform		Version Control	Continuous Integration			EASI-CLOUDS Platform
Brokerage Automation	Platform Automation		Build	Test	Packaging	Release Automation
Automated brokerage to find the most approperiate cloud provider.	Automated provisioning of platform resources.	Developers trigger CI when merging their code	Pull source code and organize dependencies	Run unit tests to ensure code quality	Create deployable artifact	Automated application deployment into production

Figure 3.9 *First use case continuous delivery blueprint which is based on deferred deployment method of ACCORDS*

2. When user wants to deploy application, he opens 'resource specifier' to specify required PaaS resources (e.g. memory, disk, geographical location of PaaS, etc.). Resource specifier sends these resources to ACCORDS.
3. Based on the requested resources, ACCORDS brokers and finds the most appropriate PaaS provider.
4. PaaS procci server instance (a dynamically instantiated object in ACCORDS) talks to COAPS associated with the target PaaS.
5. COAPS creates environment resource to specify characteristics of the applications environment (e.g. MySQL, Tomcat7, etc.) and application resource to specify characteristics of application itself (application name, path to deployable articat, etc.).
6. Application ID, Environment ID, and COAPS URL will be sent back to MIDEaaS.
7. Now user pushes the source code along with Application ID, Environment ID, and COAPS URL to the Git repository of the project.
8. Post-push command of Git repository triggers build of associated Jenkins Job. Maven will be invoked and deployable artefact will be created as the result.
9. Jenkins COAPS plugin talks to COAPS of target PaaS (defined by PaaS URL) to deploy and starts application (defined by Application ID) on previously created environment (define by Environment ID).

Figure 3.10 shows outline of this use case. Main benefit of this use case is that it allows management of deployed applications on different PaaS. In other words, whenever a PaaS is selected by ACCORDS, it will be added to a table in MIDEaaS plugin written for continuous deployment. Figure 3.11 shows a snapshot the man-

agement screen. Using management screen, the developer can stop, start, and delete application deployed on a certain PaaS.

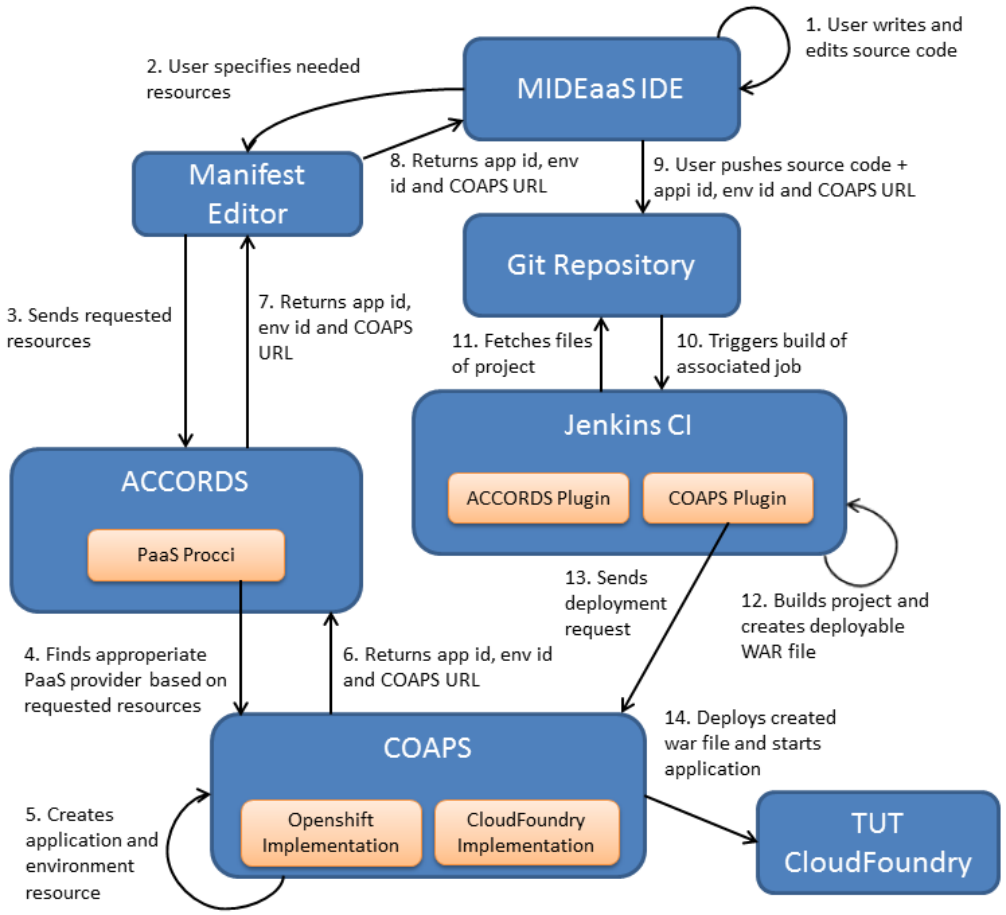

Figure 3.10 *Use case based on deffered deployment method of ACCORDS*

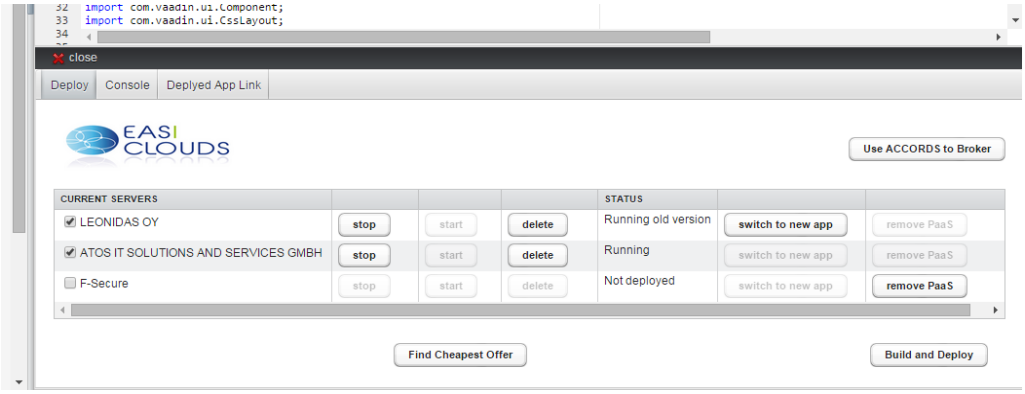

Figure 3.11 *PaaS Management Screen of MIDEaaS continuous deployment plugin*

3.2.2 Second Use Case

While first use case was based on deferred deployment method of ACCORDS, second use case is based on 'immediate deployment' method. Immediate deployment means completing deployment chain right after preparing environment and describing application on the target PaaS. Again figure 3.8 could help to describe what immediate deployment is. To implement immediate deployment, ACCORDS sends all four requests (create environment, create application, deploy application, and start application shown in figure 3.8) to COAPS one after another sequentially without pause. Continuous deployment blueprint utilised in this use case is depicted in figure 3.12.

Second Use Case - Continuous Deployment Blueprint						
Version Control	Continuous Integration			EASI-CLOUDS Platform		
	Build	Test	Packaging	Brokerage Automation	Platform Automation	Release Automation
Developers trigger CI when merging their code	Pull source code and organize dependencies	Run unit tests to ensure code quality	Create deployable artifact	Automated brokerage to find the most approperiate cloud provider.	Automated provisioning of platform resources.	Automated application deployment into production

Figure 3.12 *Second use case continuous delivery blueprint which is based on immediate deployment method of ACCORDS*

Exact details of deployment process from editing code in MIDEaaS to completing deployment on the target PaaS are investigated in sequence as following:

1. User creates a project in MIDEaaS and writes code to develop an application.
2. When user wants to deploy application, he opens 'resource specifier' to specify required PaaS resources (e.g memory, disk, geographical area of PaaS, etc.). Resource specifier sends these resources back to MIDEaaS.
3. Now user presses **Use ACCORDS to Broker** button to push the source code along with the list of requested resources to Git repository of the project.
4. Post-push command of the Git repository triggers associated Jenkins Job. Jenkins job checks out Git repository and builds project by invoking Maven. As the result, deployable artefact (WAR file) will be created.
5. Jenkins ACCORDS plugin sends deployment request to ACCORDS (along with required resources).

6. Based on required resources, ACCORDS brokers and finds the most appropriate PaaS provider.
7. PaaS procci server instance (a dynamically instantiated object in ACCORDS) talks to COAPS associated with the target PaaS.
8. COAPS creates environment resource on the target PaaS to specify characteristics of the applications environment (e.g. MySQL, Tomcat7, etc.) and application resource to specify characteristics of application itself (application name, path to deployable artifact, etc.). It also deploys the created application to the created environment and starts application. Now application is running on the target PaaS provider.

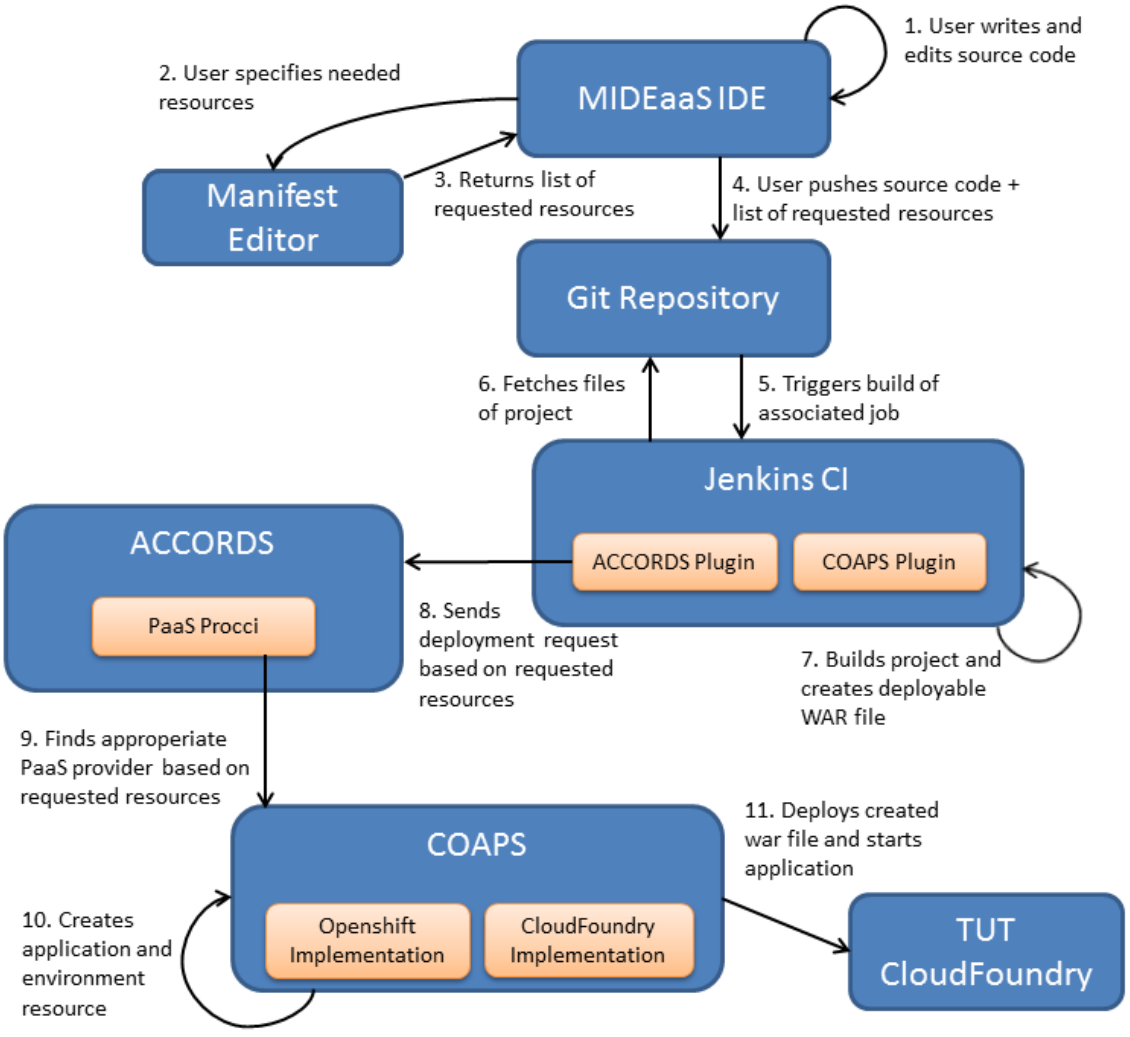

Figure 3.13 *Use case based on immediate deployment method of ACCORDS*

Figure 3.13 shows outline of this use case. The main pitfall of this use case is that user misses the advantage of having control over the applications he has deployed

previously. This defect is the result of hidden communication between ACCORDS and COAPS from the MIDEaaS point of view.

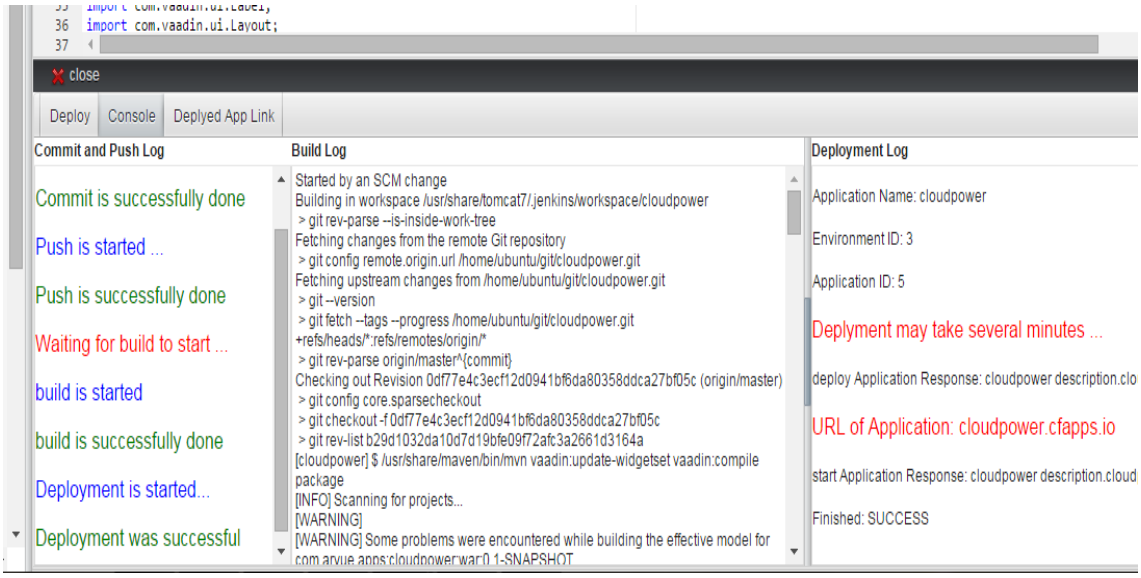

Figure 3.14 *Deployment steps report shown to user of MIDEaaS*

Both use cases have been implemented in this thesis. Success or failure of primary steps of each use case will be reported to user during deployment process, including:

- Committing changes to source code
- Pushing changes to remote Git repository
- Build log
- Deployment log in the first use case.

Figure 3.14 portrays report of each step in first use case which ended up in a successful deployment.

4. RESULTS AND DISCUSSION

4.1 Version Management System

In order to integrate version management system with MIDEaaS, two main questions should be answered:

- Does version management system applied in previous version of MIDEaaS meet continuous deployment goal targeted for this thesis? If not, what should be done? (Section 4.1.1)
- What are the issues of version management system for a real time editor? (Section 4.1.2)

4.1.1 Selecting Version Management System

Previous versions of MIDEaaS (before exploiting in EASI-CLOUDS project) had Github as its Git server. There were several problems with GitHub. First, every user must have GitHub account in advanced. Remember that in order to make MIDEaaS usage as easy as possible, even for signing into MIDEaaS only the name of user is required. With this philosophy, MIDEaaS could not expect users to have GitHub accounts already. Second, for a specific project, when some users try to push project code on their GitHub accounts, Jenkins must be configured to fetch that project from all GitHub accounts of those users. This makes the process of building a project complicated.

4.1.2 Issues

For every IDE, it is important to have version management system. Designing version management system for a web-based IDE where several users are working in a same project may seem tricky, because there is a chance for each of them to push a different version of the project and cause conflict. Fortunately, the way MIDEaaS

is designed prevents these kind of conflicts. In one hand, in MIDEaaS, changes to code are shown in a real time manner; this means that if one user changes some part of project, these changes will be applied and shown to the project itself. To get the idea better, an analogy to Google Doc may work. When a person is editing some part of a text in Google Doc, the changes are immediately applied to the document and shown to others. Therefore only one and the last version of the document - in our case, project - is available every single moment of time for all users. When a user decides to push the project, only the last version of that project is pushed. On the other hand, there is only one remote Git repository associated to each project. A user does not have local or remote Git repository exclusive for him/her. Therefore there cannot be several Git repositories with different versions of the same project. Otherwise, merging them into one Git repository would have been a real problem.

4.2 Limitations of ACCORDS

In both use cases described in section 3.2, ACCORDS lacks some certain features. In this section, I am going to investigate those defects.

4.2.1 COAPS URL in First Use Case

In first use case described in subsection 3.2.1, at sixth step (in both figure 3.10 and the list which states all steps necessary for deferred deployment), three important info must be sent from ACCORDS to MIDEaaS. This info is Application ID, Environment ID, and target COAPS URL. Application ID and Environment ID can be accessed through ACCORDS, but COAPS URL could not be achieved. In order to have management screen, COAPS URL is a must; otherwise MIDEaaS could not manage deployed application on the target PaaS by directly communicating with COAPS installed on that PaaS. It seems that current implementation of ACCORDS lacks this feature.

4.2.2 Deployable Artifact in Second Use Case

Take a look at fifth step of the list (which states all steps necessary for immediate deployment) or eighth step in figure 3.13 in second use case (subsection 3.2.2). When ACCORDS plugin for our Jenkins-based CI sends deployment request to ACCORDS, it specifies the path to the deployable artifact inside Jenkins machine in a manifest (called deployment manifest). Consequently, ACCORDS must download

war file and send it to COAPS for completing deployment. But it seems that ACCORDS is not able to download a file which is on a machine other that ACCORDS machine. We came up with a solution to handle this problem. The defect is compensated by copying war file via scp command programmatically from Jenkins machine to ACCORDS machine and making deployment manifest point to this copied war file on ACCORDS machine.

5. CONCLUSION

In this thesis a demonstrator for cloud-based SW development was developed. The demonstrator could be used by programmers to develop and deploy applications into a brokered/federated cloud.

Our SW development demonstrator utilised MIDEaaS as its IDE for two main reasons. First, MIDEaaS is a cloud-based IDE that can be offered as SaaS to developers. There is no need for developers to install and maintain IDE on their local machines. Second, MIDEaaS supports real-time collaboration so that every change is applied immediately and shown to others (like Google Docs) in a real-time manner. MIDEaaS therefore has only one and the last version of each project at every single moment of time. Hence, when a developer decides to push a project into remote Git repository, the last version is the only available version that can be pushed. Now assigning only one remote Git repository to every project in MIDEaaS makes pushing to and updating Git repository as simple as possible. Having one remote Git repository avoids further merging of several repos.

SW development practice followed by our demonstrator was continuous deployment. Continuous deployment delivers every change to actual users. Typical continuous deployment approach automates build, testing, and release (deployments both into staging and production) phases of SW development cycle. But our demonstrator skipped testing and deployment into staging; so the risk of ending up in a buggy software is high in our case. CI, CD, and continuous deployment as SW development automation practices are hot topics nowadays. Automating SW development phases accelerates feedback cycle by making reports of result (success or failure) of each phase instantly available to all project members. These practices therefore make time to release shorter and risk of buggy software lower.

For cloud federation/brokerage, COAPS and ACCORDS components of EASI-CLOUDS project are used to find the most suitable PaaS provider and negotiate relationships between that PaaS provider and MIDEaaS. Based on the two ways ACCORDS carries out deployment into the target PaaS provider, two use cases were offered, each of which has its own advantages and disadvantages. The use case which is based on

deferred deployment method of ACCORDS allows a developer to have control (e.g. stop, start, update, undeploy, etc.) over his deployed applications; the problem is current implementation of ACCORDS lacks some features to carry out this use case. The second use case is based on immediate deployment method of ACCORDS. Although (with some manipulation) current implementation of ACCORDS supports the second use case, losing control over deployed application is the main pitfall of this method.

Our demonstrator was presented in the final ITEA review meeting held in Berlin and received highly positive feedback.

As stated previously current version of our demonstrator lacks testing and deployment into staging phases. As future work theses phases of SW development could be added to our continuous deployment solution. Furthermore typically there is no need to deliver every change to actual users. If the testing and staging environment are available for performing further comprehensive tests, it is recommended to apply continuous delivery instead of continuous deployment. CD keeps software in a release ready state in staging environment; actual release into production could be done manually by simply pressing a button in any time.

Another plan for later work should be implementing missing features of ACCORDS so that it suits both use cases presented in this thesis.

MIDEaaS IDE could be improved on various ways. Main culprit for failing or delaying a project is communication and collaboration of developers involved in the process of development. Since MIDEaaS is a cloud-based IDE where all developers log in to it in order to write code, it is the best place to provide communication tools for developers. Although current version of MIDEaaS has some collaborative features like chatting, more innovative ideas like combining MIDEaaS with tools similar to Confluence[1] and JIRA[2] could be interesting for future work. For example when a feature is implemented successfully, a notification to other developers could be sent. Furthermore, using current version of MIDEaaS, a developer can only develop Vaadin projects. Making MIDEaaS support more programming languages like C++, Python, etc. would be a nice idea.

[1] `https://www.atlassian.com/software/confluence`
[2] `https://www.atlassian.com/software/jira`

BIBLIOGRAPHY

[1] J. Ames, "Federated cloud: A new frontier," January 2013, Available (accessed on 17.11.2013): http://blog.appcore.com/blog/bid/170244/Federated-Cloud-A-New-Frontier.

[2] C. Caum, "Continuous delivery vs. continuous deployment: what's the diff?" August 2013, Available (accessed on 26.11.2014): http://puppetlabs.com/blog/continuous-delivery-vs-continuous-deployment-whats-diff.

[3] "Cloud-broker," Available (accessed on 21.11.2014): http://www.itwissen.info/definition/lexikon/Cloud-Broker-cloud-broker.html.

[4] "Cloud federation," Apprenda Inc. Available (accessed on 17.11.2013): http://apprenda.com/library/glossary/definition-cloud-federation/.

[5] "Coaps api: A generic cloud application provisioning and management api," institut Mines-Telecom, Telecom SudParis. Computer Sciences department, SIMBAD team Available (accessed on 2.09.2014): http://www-inf.int-evry.fr/SIMBAD/tools/COAPS/.

[6] "Continuous cloud delivery," [White paper], CloudBees, Inc. Available (accessed on 26.08.2014): http://pages.cloudbees.com/rs/cloudbees/images/Continuous-Cloud-Delivery.pdf.

[7] "Continuous integration," ThoughtWorks, Inc. Available (accessed on 25.11.2014): http://www.thoughtworks.com/continuous-integration.

[8] M. C. Daconta, "Cloud broker software an emerging force for enterprise migrations," April 2013, Available (accessed on 21.11.2014): http://gcn.com/articles/2013/04/23/cloud-broker-software-enterprise-migrations.aspx.

[9] "Deliverable1.3 - business models for the easi-clouds use cases and analysis of market impact," Available (accessed on 17.11.2013): https://redmine.dai-labor.de/EASI-CLOUDS/projects/easi-clouds/dmsf?folder_id=214.

[10] "Deliverable1.5 - final business models for easi-clouds," ITEA3. Available (accessed on 17.11.2013): https://itea3.org/project/workpackage/document/download/1931/10014-EASI-CLOUDS-WP-1-D15-Finalbusinessmodels.

[11] "Easi-clouds project flyer," Available (accessed on 21.10.2014): https://itea3.org/project/workpackage/document/download/964/10014-EASI-CLOUDS-WP-5-D51A-Projectflyer.pdf.

[12] "Federated cloud (cloud federation)," Available (accessed on 17.11.2013): http://whatis.techtarget.com/definition/federated-cloud-cloud-federation.

[13] M. L.-R. Florian Chazal, "Easi-clouds kick-off preparation," Available (accessed on 2.09.2014): http://www.compatibleone.com/community/wp-content/uploads/2013/06/20130618-COandEASICloudsv03.pdf.

[14] M. Fowler, "Continuous integration," May 2006, Available (accessed on 24.11.2014): http://www.martinfowler.com/articles/continuousIntegration.html.

[15] B. Gopularam, C. Yogeesha, and P. Periasamy, "Highly scalable model for tests execution in cloud environments," in *Advanced Computing and Communications (ADCOM), 2012 18th Annual International Conference on*, Dec 2012, pp. 54–58.

[16] J. Humble, "Continuous delivery vs continuous deployment," August 2010, Available (accessed on 26.11.2014): http://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/.

[17] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.

[18] Y. Jadeja and K. Modi, "Cloud computing - concepts, architecture and challenges," in *Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on*, March 2012, pp. 877–880.

[19] C. Janssen, "Cloud broker," Available (accessed on 21.11.2014): http://www.techopedia.com/definition/26518/cloud-broker.

[20] S. Karadzhov, "Fundamentals of continuous integration with jenkins and zend server," Zend Technologies Ltd, Available (accessed on 26.11.2014): http://static.zend.com/topics/WP-Fundamentals-of-Continuous-Integration-with-Jenkins-and-Zend-Server-2014-03-31-EN pdf?

[21] K. Kawaguchi, "Geek choice awards 2014," July 2014, Available (accessed on 29.08.2014): http://jenkins-ci.org/content/geek-choice-awards-2014.

[22] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze, "Cloud federation," in *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011)*. IARIA, September 2011, Inproceedings, best Paper Award.

[23] J. Lautamäki, A. Nieminen, J. Koskinen, T. Aho, T. Mikkonen, and M. Englund, "Cored: Browser-based collaborative real-time editor for java web applications," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, ser. CSCW '12. New York, NY, USA: ACM, 2012, pp. 1307–1316. [Online]. Available: http://doi.acm.org/10.1145/2145204.2145399

[24] F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf, *NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology (Special Publication 500-292)*. USA: CreateSpace Independent Publishing Platform, 2012.

[25] P. M. Mell and T. Grance, "Sp 800-145. the nist definition of cloud computing," Gaithersburg, MD, United States, Tech. Rep., 2011.

[26] "Openstack: The open source cloud operating system," OpenStack Foundation, Available (accessed on 8.09.2014): http://www.openstack.org/software/.

[27] C. Pettey and R. van der Meulen, "Gartner says cloud consumers need brokerages to unlock the potential of cloud services," September 2009, Available (accessed on 21.11.2014): http://www.gartner.com/newsroom/id/1064712.

[28] M. Rouse, "cloud broker," Available (accessed on 21.11.2014): http://searchcloudprovider.techtarget.com/definition/cloud-broker.

[29] M. Sellami, S. Yangui, M. Mohamed, and S. Tata, "Paas-independent provisioning and management of applications in the cloud," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, June 2013, pp. 693–700.

[30] T. Sridhar, "Cloud computing: A primer, part 2: Infrastructure and implementation topics," *The Internet Protocol Journal*, vol. 12, no. 4, pp. 1–40, December 2009.

[31] K. Srinivasa, C. Harish Raddi, S. Mohan Krishna, and N. Venkatesh, "Meghaos: Cloud based operating system and a framework for mobile application development," in *Information and Communication Technologies (WICT), 2011 World Congress on*, Dec 2011, pp. 858–863.

[32] "The compatible one application and platform service api specification," telecom SudParis. Computer Sciences department. Available (accessed on 2.09.2014): http://www.compatibleone.com/community/wp-content/uploads/2014/05/COAPS-Spec.v1.5.3.pdf.

[33] "Understanding the bosh deployment manifest," Pivotal Software Inc. Available (accessed on 13.09.2014): http://docs.cloudfoundry.org/bosh/deployment-manifest.html.

[34] "Understanding the cloud computing stack saas, paas, iaas," [White paper], Rackspace, US Inc. Available (accessed on 04.09.2014): http://radar.oreilly.com/2009/03/continuous-deployment-5-eas.html.

[35] "Vmware news releases, vmware delivers cloud foundry, the industry's first open paas," VMware Inc. Available (accessed on 8.09.2014): http://www.vmware.com/company/news/releases/cloud-foundry-apr2011.

[36] "Vvmware cloud foundry - an insight," October 2014, Available (accessed on 8.09.2014): http://www.eukhost.com/blog/webhosting/cloud-foundry-an-insight/.

[37] S. Yangui, I.-J. Marshall, J.-P. Laisne, and S. Tata, "Compatibleone: The open source cloud broker," *Journal of Grid Computing*, vol. 12, no. 1, pp. 93–109, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10723-013-9285-0