Université catholique de Louvain

Louvain School of Engineering

Computing Science Engineering Department

Migration of Mobicents Sip Servlets on a Cloud Platform

Promoteur: Peter Van Roy
Co-promoteur: Sabri Skhiri (Euranova)
Lecteur: Jean Deruelle (Mobicents)

Mémoire présenté en vue de l'obtention du grade de master ingénieur civil en informatique, option networking and security, par Thibault Leruitte.

Louvain-la-Neuve

Academic year 2010 - 2011

Abstract

Mobicents Sip Servlets helps the users to create, deploy, and manage services and applications that integrate voice, video and data in real time over IP networks. Mobicents Sip Servlets has not offered a cloud support yet. Running Mobicents Sip Servlets in a cloud would allow the users to adapt the resources used by Mobicents Sip Servlets to the real time load of the applications. We have designed and implemented a solution to migrate Mobicents Sip Servlets on a cloud platform. This document will present this solution.

Our solution provides a set of cloud images that support the Mobicents Sip Servlets platform. Besides, our solution provides an auto-scaling algorithm that automatically scales the number of instances of Mobicents Sip Servlets. It is now possible to deploy SIP applications in a cloud and the platform will adapt its resources according to the needs. The solution has been tested positively on two use cases. It is now ready to be tested on a real world example.

Contents

Introduction

Mobicents Sip Servlets is an application server which runs on top of JBoss AS and which provides an implementation of the SIP Servlet specification. The purpose of this platform is to help the users to create, deploy, and manage services and applications that integrate voice, video and data in real time over IP networks. Because of the large amount of data the applications must process, they require high resources to run.

Most vendors of application servers now offer an Amazon image in order to use their products on a cloud. Indeed, a cloud can provide the high resources needed by the applications since it is easy to add or remove some resources allocated to the applications in a cloud.

However, Mobicents Sip Servlets has not offered a cloud support yet. In order to offer a cloud support, Mobicents Sip Servlets should offer cloud images that would contain the Mobicents Sip Servlets platform. Besides offering these images, another problem arise in a cloud environment. This problem is how to scale the number of instances of these image according to the real time load. A cloud platform needs an entity which control the resources usage and which can allocate more or less resources depending on the load.

This thesis will study the migration of Mobicents Sip Servlets on a cloud platform. We will provide cloud images that will contain the Mobicents Sip Servlets platform. Besides, we will propose and implement a solution to scale up and down the resources used by the cloud.

The Part I provides an overview of the state of the art. The Part II states the problem that arise in the current situation. The Part III explains the design of our solution in order to resolve this problem. Finally, the Part IV concludes this thesis by summarizing the results achieved and listing the further works that need to be done.

Part I

State of the art

Chapter 1

Cloud computing

This chapter aims to introduce the reader to cloud computing.

In the first section, we will discuss general concepts of cloud computing. In the following sections, we will study two cloud services providers: Amazon Web Services and Eucalyptus. Amazon Web Services is a public cloud provider, while Eucalyptus is a private cloud provider.

Contents

1.1 The cloud architecture

Cloud computing is a client-server approach where the server is a network of nodes [55]. The job of the server is distributed among several nodes that form a cloud. Some nodes can be dynamically added or removed at runtime to the cloud (see Figure 1.1). Depending on the cloud, the client may either send its requests to any node in the cloud, or it may send its requests to a load balancer, which will act as an entry-point for the cloud and forward the requests to a node.

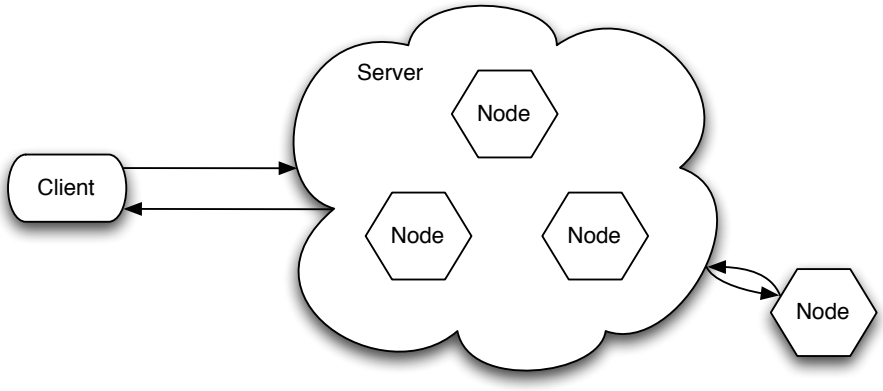

Figure 1.1: Cloud computing is a client-server approach. Nodes can be dynamically added or removed to the cloud.

In practice, a node is an instance of a virtual machine. Thus a node is also called an instance. The virtual machines are executed on a set of inter-connected physical machines. The set of physical machines is called a cluster.

A virtual machine is a machine that is not executed on a physical machine, but on a software that provides a virtualization layer. The software providing this virtualization layer is called hypervisor [64].

1.1.1 Service models

There exist three service models: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). The Figure 1.2 illustrates those three different models. Those three models can be defined by the unit the user is gaining if she uses them [19].

IaaS If the user uses an IaaS, she is basically gaining a computer. The user has access to a virtual machine. She can decide which OS and software install on it. Then, she can use it as any server. An example of IaaS provider is Amazon Web Services (see Section 1.2).

PaaS If the user uses a PaaS, she is gaining a software or a framework. A PaaS is a hosted tool that the user can leverage to build something on. An example of PaaS is Google AppEngine [26]. With AppEngine, the user can run her web

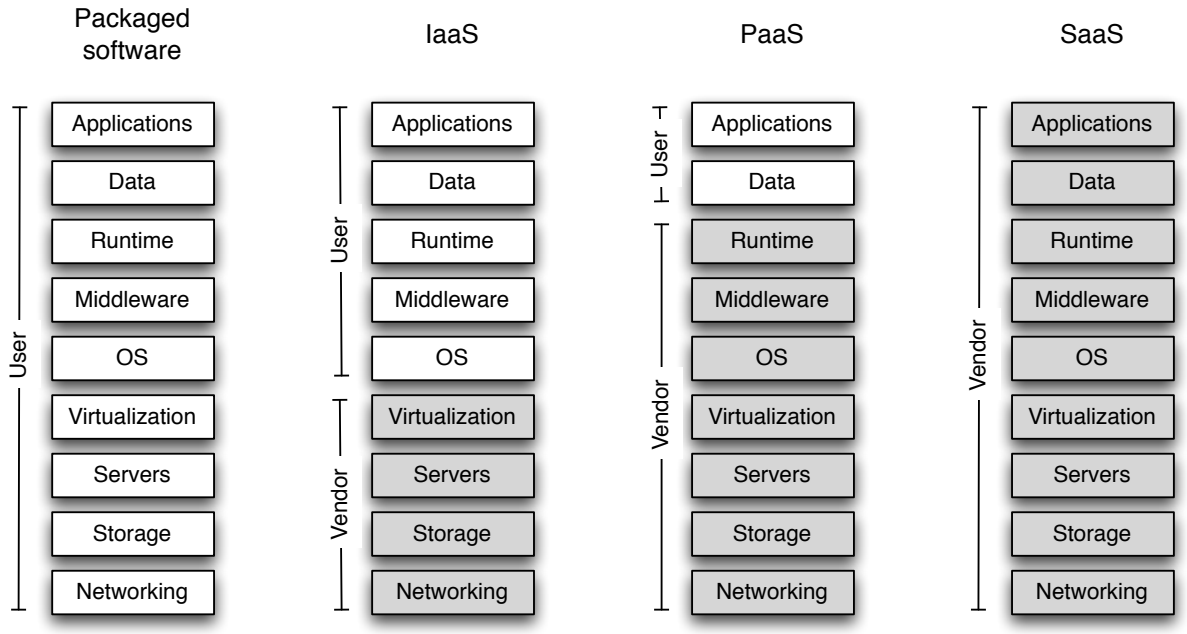

Figure 1.2: In the packaged software model, the user is in charge of all the components. In the IaaS model, the user is in charge of the components from the applications to the OS. In the PaaS model, the user is in charge of the applications and the data. Finally, in the SaaS model, the user has anything to do in order to use the software. Figure inspired by [19].

applications on Google's infrastructure. AppEngine provides the hardware, the OS and the framework. The user has to provide only her applications. Then, the clients can access her applications.

SaaS If the user uses a SaaS, she is gaining a business functionality. Basically, a SaaS is an software the user uses, but that is installed on a provider's hardware. For example, all web applications are SaaS. The user uses the software, but she has not had something to install.

1.1.2 Deployment models

There also exist three deployment models: public cloud, private cloud and hybrid cloud [35].

Public The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

Private The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.

Hybrid The cloud infrastructure is a composition of a public cloud and a private cloud. Usually, we try to maximize the usage of the private cloud. The public cloud is used only for periodically bursts, when the private cloud is overloaded.

1.1.3 Properties

Finally, a cloud has two main properties: elasticity and pay per use.

Elasticity The user can scale resources usage up and down rapidly. In the IaaS cloud model, this means that it is possible to start new instances or stop existing instances rapidly. The newly created instances must be used by the cloud in order to decrease the load of the other instances.

Pay per use The user only pays for the resources that she actually uses. The IaaS cloud providers usually charge an instance on an hourly basis.

1.1.4 Definitions

To conclude this section, we can give the definition of some terms for the IaaS cloud model:

Image An image is a virtual machine that can run on a cloud.

Instance An instance is a running image.

Service A service is what an image provides.

Artifact An artifact is a deployable file on an image.

Platform A platform is a set of images.

Environment An environment is a set of instances of a platform's images, plus the artifacts that are deployed on those instances.

1.2 Amazon Web Services

Amazon Web Services (AWS) provide a set of services related to cloud computing [16]. We will discuss first about Amazon EC2, an IaaS cloud provider, and then about Amazon S3 and Amazon EBS, two storage services.

1.2.1 Amazon Elastic Compute Cloud (Amazon EC2)

Amazon EC2 is a web service that provides an IaaS. EC2 let the user run Amazon Machine Images (AMI). An AMI is basically a cloud image, as defined in the Section 1.1. An AMI can be either S3 backed or EBS backed (see next sections.)

As a cloud provider, EC2 respects the properties of a cloud:

Elasticity The user is free to run as many instances of an AMI as she wants. She can easily scale up or scale down the number of instances. Starting an instance is really fast, it does not take more than a few minutes [37].

Pay per use The user only pays for the instances that are running. The price is computed on an hourly basis for each instance. She also pays the internet traffic that goes in or out an instance.

In the next paragraphs, we will present some features that Amazon EC2 provides.

Regions and availability zones

The Amazon Web Services are located on multiple locations called regions. Each region is independent and geographically dispersed from the others. When an user chooses to use the Amazon Web Services, she has to choose first in which region she wants to use them. Amazon EC2 is currently available in five regions: US East (Northern Virginia), US West (Northern California), EU (Ireland), Asia Pacific (Singapore), and Asia Pacific (Tokyo).

Moreover, each region is divided into availability zones. Availability zones are distinct locations within the same region. If the user wants to build a reliable application protected from failure of a single location, she can launch instances in separate availability zones.

As a side story, during the elaboration of this thesis, Amazon EC2 and EBS have been unavailable during more than 24h [59]. The service disruption started on April the 21st, 2011. As a result, many well-known websites such as Foursquare, Heroku and Reddit have been taken down [2]. However, the disruption mostly affected only one availability zone. In order to build high-reliable applications, it is important to take advantage of multiple availability zones. Those well-known websites would not have been so hardly affected if they had been deployed on multiple availability zones.

Networking

When an instance is started, it gets two IP's: one public IP and one private IP. The private IP can be used only within the region the instance is running in. For example, the private IP should be used if two instances need to communicate together. On the other hand, the public IP can be used from anywhere.

The traffic sent though the private IP within the same availability zone is free. The traffic sent though the private IP from one availability zone to another one is charged at a small rate. Finally, the traffic sent thought the public IP is charged at a full rate.

Elastic IP is a feature that provides a static IP. The user can reserve an elastic IP independently from one instance. So the lifecycle of the elastic IP is independent from the lifecycle of any instance. Then, the user can attach and detach the elastic IP to an instance. The elastic IP behaves like a public IP. This service is free only if the elastic IP is attached to an instance. If the user reserves an elastic IP but she does not attach it to an instance, she will have to pay.

Finally, one trick is worthy of note: each IP is associated to a hostname, and there is a DNS server which is used to translate the hostname to the IP address. If the user

queries the DNS server with the hostname of the public IP, she gets the public IP. However, if she does so from an instance within the availability zone, the DNS server will return the private IP of the instance which is associated to the hostname's public IP.

Security groups

A security group is a set of ACCEPT firewall rules for incoming packets that can apply to TCP, UDP or ICMP [56]. An instance must belong to at least one security group, and may belong to several security groups.

By default, all incoming traffic to any instance is dropped, even if the traffic comes from inside the availability zone. An instance must be put in a security group if it should receive some traffic.

Community AMI's

The community AMI's is a repository which contains many AMI's. Everyone can submit an AMI to the community AMI.

1.2.2 Amazon Simple Storage Service (Amazon S3)

Amazon S3 is a web service that provides scalable and reliable storage [15]. The user can store any amount of data on Amazon S3.

A piece of data that is stored on Amazon S3 is called an object. An object's size can vary from 1 byte to 5 terabytes. Each object must be stored within a bucket, which is a container for objects. A bucket can contain an unlimited number of objects. A bucked is stored within a region, and will never be transferred to another region unless the user requests it.

Amazon S3 can be used to store S3 backed AMI. It is possible to run the AMI's that are stored on Amazon S3 with Amazon EC2. The drawback of this solution is that the instance is not persisted. It means that all the modifications that have been done to the instance are not saved.

1.2.3 Amazon Elastic Block Store (Amazon EBS)

Amazon EBS volumes are storage volumes that can be attached to a running instance and exposed as a device [14]. Amazon EBS volumes persist independently from the life of the instance. Note that an EBS volume can belong to at most one instance at a time. There are two usages possible of an EBS volume.

First, EBS volumes can be seen as USB keys or external hard-drives. The volume can be attached or detached to an instance. Some data may be stored on the volume. If the instance is terminated, the data stored on the EBS volume will persist.

Secondly, EBS volumes can be used as a root volume for an AMI. This is the case for the EBS backed AMI's. When an EBS backed AMI is started, a new EBS volume is created. This volume will handle the root device of the instance and will persist the data. Thus, the instance can be stopped and restarted later on.

It is also possible to take a snapshot of a volume. A snapshot is basically a copy of the volume. For example, a snapshot can be useful to save the data to a certain state. Snapshots can also be useful to create an AMI. If the snapshot is made from an root volume, a new EBS backed AMI can be created from the snapshot. The new AMI will contain exactly the same data that are contained in this volume.

1.3 Eucalyptus

Eucalyptus is an open-source software that provides on-premise IaaS cloud [22]. Eucalyptus stands for Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems.

One goal of Eucalyptus is to be interface-compatible with Amazon EC2. It means that Eucalyptus delivers the same API as Amazon EC2.

Ubuntu provides Ubuntu Enterprise cloud, which transforms Ubuntu into an IaaS cloud provider. Ubuntu Enterprise cloud is fully powered by Eucalyptus [62].

1.3.1 Eucalyptus elements

Eucalyptus has been designed to reflect the physical architecture below it. Eucalyptus may be installed on several machines in order to compose a cloud. Each physical machine contains a controller (node controller). The machines may be physically or logically put together in some sets. Those sets are called clusters. A machine is responsible for a cluster and hosts a controller (cluster controller). Finally, there should be a single entry-point for the cloud. This is the cloud controller. This design is illustrated by the Figure 1.3.

Node Controller (NC) The role of the NC is to interact with the OS and the hypervisor. The hypervisor can be KVM or Xen. The NC has to discover the resources available of the host it runs on. It also has to be aware of the virtual machines that run on the host. The NC responds to the start instance request, the stop instance request, and the availabilities queries from the CC.

Cluster Controller (CC) The CC operates as an intermediate between the CLC and the NC's. Its role is to forward the requests from the CLC to a NC. When the CLC requests to start an instance, the CC will choose the NC on which the instance will be started. This decision is based on the available resources of each host.

Cloud Controller (CLC) The CLC provides the interface the users is interacting with. The CLC handles the requests of the users and forward them to the CC. The

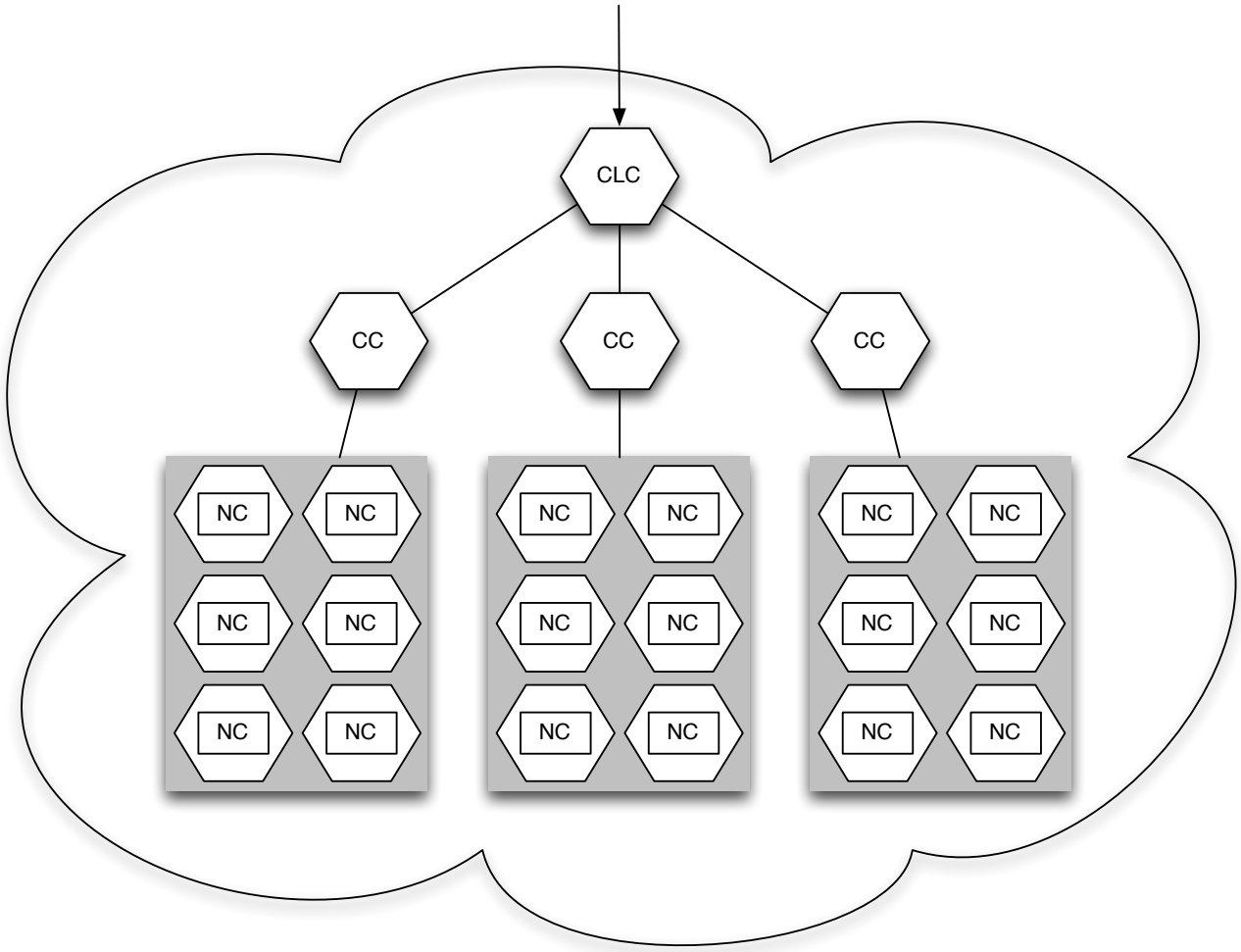

Figure 1.3: Eucalyptus has 3 main elements: a Cloud Controller (CLC), some Cluster Controllers (CC) and several Node Controllers (NC).

CLC chooses which cluster a new instance should be started on. Unfortunately, the CLC is a single point of failure. If the CLC goes down, the users will not be able to control their instances. However, the running instances stay up and available.

Besides, two storage controllers may be used.

Walrus Storage Controller (WS3) WS3 is used to store the Eucalyptus Machine Images (EMI) that can be instantiated on the cloud. The WS3 element is hosted on the same machine that hosts the CLC. The API of WS3 is compatible with the API of Amazon S3.

Elastic Block Storage Controller The elastic block storage controller allows the users to create persistent block devices that can be mounted on running instances. The elastic block storage controller is hosted on the same machines that host the CC's.

The set formed by one cluster controller and several node controllers is equivalent to an Amazon EC2 availability zone.

1.3.2 Networking

When the administrator installs the Eucalyptus software, he must choose the networking configuration. He can choose among the four following configurations [62].

System This is the simplest networking mode. In this mode, the ethernet interface of the instance is connected to the physical ethernet interface of the host through a bridge. The instances may get an IP address from the DHCP server of the network the host is connected to. The instances will get an IP address the same way a physical machine connected to the network would.

Static In this mode, Eucalyptus hosts a proper DHCP server and hosts a map of MAC addresses and IP addresses. The ethernet interface of the instance is connected to the physical ethernet interface of the host through a bridge, like in the System mode. When a new instance is started, Eucalyptus assigns a free MAC address and an IP address from the map to the instance.

Managed This mode provides the most features. Like in Static mode, Eucalyptus hosts a proper DHCP server. The server is configured with a pool of IP addresses. This mode provides some features similar to the security groups and the elastic IP of Amazon EC2. Besides, this mode ensures network isolation between the instances of different security groups. This features is implemented with VLAN. A VLAN ID is associated to each security group. An instance receives only the traffic that is tagged with the VLAN ID of the security group the instance belongs to.

Managed-novlan This mode provides the same features as the Managed mode, except the network isolation. Indeed, the VLAN are not used, so the network isolation is not guaranteed.

The Table 1.1 presents a summary of the features of each mode.

	System	Static	Managed	Managed-novlan
Auto DHCP			✓	✓
Elastic IP			✓	✓
Security groups			✓	✓
Network isolation			✓	

Table 1.1: Features of Eucalyptus networking modes.

Chapter 2

J2EE and JBoss AS

In this chapter, we will first introduce Java Enterprise Edition (J2EE). Java Enterprise Edition is a Java framework developed by Oracle, and widely used in enterprises.

Afterwards, we will discuss JBoss AS, which provides the Java Enterprise Edition services.

Finally, we will discuss RHQ. RHQ is a tool that can be used to monitor and manage JBoss AS.

Contents

2.1 Java EE

Java Enterprise Edition (also known as J2EE) is a specification based on the Java language. The purpose of this specification is to help to build large-scale, multi-tiered, scalable, reliable and secure network application [23].

2.1.1 Multi-tiered application

An enterprise application is very often composed of tiers. A tier, or a layer, is a component within an application. A tier aims to isolate the functionality of an application between different components. The Figure 2.1 illustrates the different tiers.

For each tier, Java EE specifies a container. The containers provide the functionalities that are needed by the applications within each tier.

The client tier is an networked application that communicates with a Java EE application through the application client container. Usually, the applications that run in the client tier are not Java programs. This tier runs on the client's machine.

The web tier handles the interactions between clients and the business tier. It is mainly used to maintain the state of data for a client's session. The web tier uses the web container. This tier runs on the server machine.

The business tier handles the logic of the application. The business tier uses the Enterprise Java Bean (EJB) container, which manages the execution of the beans (see Section 2.1.3). This tier runs on the server machine.

The data tier is responsible for handling the data of the application. Very often, this tier is a database. This tier usually runs on a separate machine.

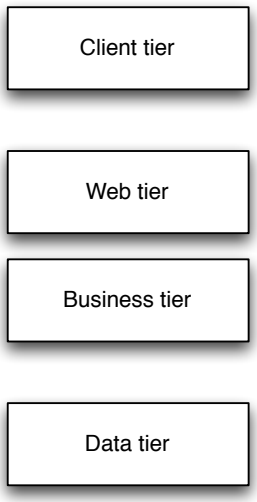

Figure 2.1: An application is composed of several tiers.

2.1.2 Application server

A Java EE application server is a framework that implements (a subset of) the Java EE APIs and provides the standard Java EE services. The application server provides the web tier and the business tier containers.

An application server implements the most used services in web applications, so the developers can focus on implementing the business logic [63]. An application server especially provides those features:

- Handle the communication with the client.
- Persist the data and handle communication with the database.
- Execute the Beans.
- Provide clustering, fail-over and load-balancing.
- Provide caching.
- Manage transactions.

JBoss AS is the most used Java application server (see Section 2.2).

2.1.3 Enterprise Java Bean

An Enterprise Java Bean (EJB) is a server-side component that encapsulates the business logic of an application [29].

The enterprise bean lifecycle is completely managed by the application server. The application server is responsible for creating and destroying the enterprise beans. The beans may also be passivated or activated by the application server. Passivation is the action of moving the resources of the bean to a persistent storage in order to free some memory.

An enterprise bean can be accessed by local clients of the application server or remote clients. The clients can access the enterprise bean by its logical name via JNDI [39]. JNDI is a naming and directory services for Java applications. In this case, a bean is registered to the JNDI server with the bean's name. Any client can retrieve the bean from the JNDI server by providing the bean's name.

There exist two sorts of enterprise beans: session EJB and message-driven EJB. Session EJB's are used to perform a task for a client. Session EJB's can be stateless or stateful. Message-driven EJB are used to execute asynchronous tasks.

2.1.4 Servlet

A servlet is a Java class that receives HTTP requests and sends back HTTP responses (see Figure 2.2). A servlet is always executed within a servlet container in

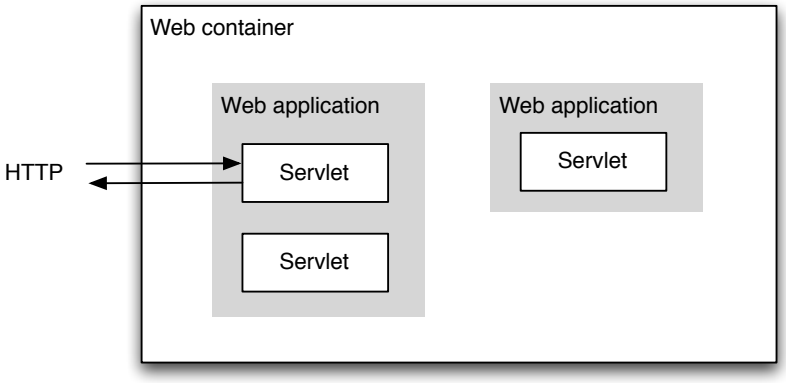

Figure 2.2: A servlet is a Java class that sits in the web container and handles HTTP requests.

the web tier. The servlet container is used in order to persist the servlets among the requests of the clients.

A servlet can be loaded automatically when the server starts, or it can be loaded when the first request from a client is received. Once a servlet is loaded, it will stay active within the container and will wait for other requests from the client.

A web application is composed of one or more servlets.

Apache Tomcat is a servlet container [60]. JBoss AS uses JBoss Web which is derived from Apache Tomcat.

2.2 JBoss AS

JBoss AS [43] is an open-source Java EE application server, written in Java, and developed by Red Hat [47]. JBoss AS implements the Java EE part of Java.

We will first discuss some general concepts about JBoss AS. Among all its features, JBoss AS provides clustering facilities. We will then discuss this feature [17].

2.2.1 General concepts

Server configuration JBoss AS aims to provide services to applications. However, an application does not always need all of these services. JBoss AS 5.1 ships with five different server configurations. These configurations are: *minimal*, *default*, *all*, *standard* and *web*. Each of these configurations provides a different set of services.

The *minimal* configuration does not provide any service, thus the developers can choose which services they want to include in the application server. The *default* configuration contains the more frequently used J2EE services required to deploy an application. The *all* configuration starts all the services, including clustering services. The *standard* configuration is the J2EE 5 certified configuration of

services. The *web* configuration can be used to build simple web applications.

Application An application is a J2EE program that takes advantage of the features provided by JBoss AS. When an application is destined to be deployed on JBoss AS, the application must be packaged as a Java archive (JAR, WAR, SAR or EAR).

Deploy An application can be deployed on JBoss AS. During its deployment, the application is installed on the application server and the application is launched. The archive of the application must be put in the deploy folder of the JBoss AS server. The application will be automatically deployed. JBoss also supports hot deployment. It means that it is not needed to restart the JBoss AS server when an archive is put in the deploy folder, the archive will be deployed automatically at runtime.

Configuration The server also contains a conf folder. All the configuration relative to the server is put in this folder.

2.2.2 Clustering

Clustering allows the user to run an application on several servers. Each server runs on a node of the cluster. The *all* server configuration automatically provides the clustering feature. All the user has to do in order to have a JBoss AS cluster is to start several instances of JBoss AS in the *all* configuration. The Figure 2.3 illustrates a JBoss AS cluster.

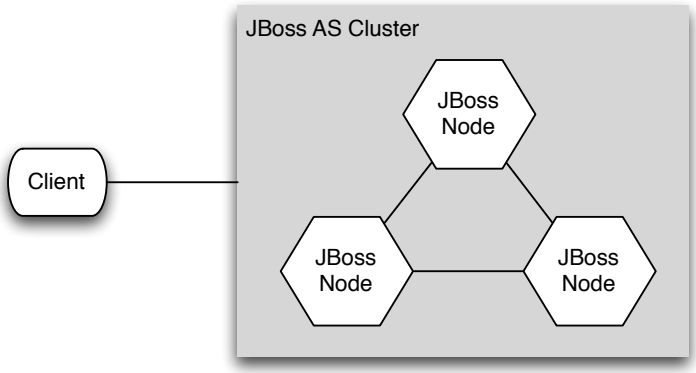

Figure 2.3: In a JBoss AS cluster, all the nodes are identical. The client can contact any node.

Modes

An application can be deployed in two modes: farming deployment or HASingleton deployment.

Farming In this mode, the application will be deployed on each node of the cluster, and each node will execute the application.

HASingleton In this mode, the application will be deployed on each node of the cluster. However, only one node will be allowed to execute the application. This node is called the master node. If the master node fails, another node will be elected as master node and will execute the application. Thus, a leader election algorithm is used in order to guarantee that there is only one master node at a time.

Properties

JBoss AS clustering provides three properties: elasticity, high availability and failover.

Elasticity This property ensures that it is possible, at any time, to add or remove a node to the cluster. New nodes are automatically discovered and added to the cluster. Nodes that are properly shut down leave the cluster and the cluster is still functional.

High availability This property ensures that is it always possible to serve new requests. In order to ensures this property, several nodes must be able to deliver the same service.

Failover This property ensures that, if a node fails, the existing sessions will not be affected and that the requests linked to these sessions will be served. If a node fails, another node takes over. This property has a very important consequence: in order to guarantee failover, all the data of a node must be replicated on all other nodes. This consequence is discussed more in the Replication section.

Replication

In order to provide failover, all the data of a node must be replicated on all nodes. This is particularly true for HTTP sessions handled by web applications. Note that a web application must be annotated as distributable in order to have its sessions replicated.

JBoss cache is a fully distributed cache framework that is used by the standard JBoss AS services. A node will store two types of data: active sessions and passive sessions. The active sessions are used by the node and stored in cache. The passive sessions are the replicas from the other nodes' active sessions. The passives sessions are stored on a persistent storage.

When a session is unused, it can be removed from memory and stored on a persistent storage. This process is called passivation. If the session is requested by a client, it can be activated back into memory and removed from persistent storage. A session can be passivated for one of the following reasons:

- The session has not been used in greater than a configurable maximum idle time.

- The number of active sessions exceeds a configurable maximum and the session has not been used in greater than a configurable minimum idle time.

Load balancing

In case of HTTP-based JBoss AS services, an external load balancer must be used in order to balance the HTTP requests among the nodes of the cluster. The client is only aware of the load balancer, it is not aware of any other part of the cluster. The JBoss AS cluster is often used with the mod_jk Apache module, but other load balancers can be used as well.

It is highly recommended for the load balancer to support sticky sessions. With sticky sessions enabled, once the load balancer routes a request from a client to a node and the server initiates a session, all future requests associated with that session must be routed to this node, as long as this node is available. Sticky sessions are useful because they allow a session to be active on only one node, and passive on the other nodes.

When a node of the cluster fails, the load balancer is responsible to be aware of the failure and route the traffic among the other nodes of the cluster. This operation is called failover and is illustrated by the Figure 2.4.

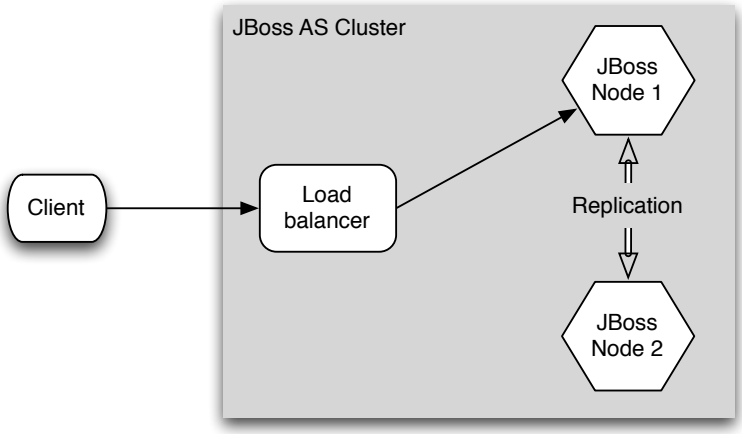

Figure 2.4: If the node 1 fails, the load balancer is responsible for failover and must send the traffic to the node 2 instead.

A potential problem with an external load balancer architecture is that the load balancer itself may be a single point of failure. It needs to be monitored closely to ensure high availability of the entire cluster's services. Another solution is to replicate the load balancer itself among several instances. If this last solution is chosen, DNS load balancing can be used in order to balance the requests from the clients to the available load balancers [30].

2.3 RHQ

RHQ (formerly known as Jopr) is a system management suite that provides extensible and integrated systems management for multiple products and platforms [48]. RHQ provides the following features:

- Monitoring and graphing of values.
- Alerting on error conditions.
- Remote configuration of managed resources.
- Remote operation execution.

RHQ is the *de facto* way to monitor and manage JBoss AS instances.

RHQ is designed as a server/agent architecture. An agent runs on each resource that should be monitored or managed. The server is executed on an independent resource and the agents communicate with the server in order to report values and execute operations (see Figure 2.5).

The agent supports plugins, which can be used to manage specific resources. For example, the agent ships with a JBoss AS plugin which allow RHQ to manage JBoss AS 6.

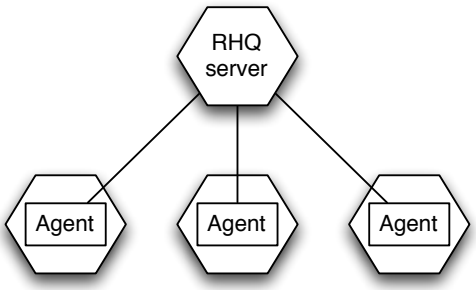

Figure 2.5: RHQ supports a server-agent architecture.

Chapter 3

JBoss AS meets the clouds

In this chapter, we will discuss some tools that are useful to carry JBoss AS in the clouds. Although all those tools are not specific to JBoss AS and can be used in another contexts as well.

First we will see Cantiere, which is used to build RPM packages.

Then we will discuss BoxGrinder, which create appliances in order to deploy them in the cloud. BoxGrinder can use the RPM packages that are built with Cantiere.

CirrAS is a set of appliances that provides a JBoss AS environment for the clouds. Those appliances are created by BoxGrinder.

Finally, we will discuss SteamCannon, which manages the appliances in a cloud.

Contents

3.1 Cantiere

Cantiere is a set of Rake tasks to build RPM files [41]. Rake is a build tool for Ruby. Rake is comparable to Make. A Rake task is basically the same thing than a rule of a Makefile.

As illustrated by the Figure 3.1, Cantiere takes as input a spec file and some sources files and products a RPM package as an output.

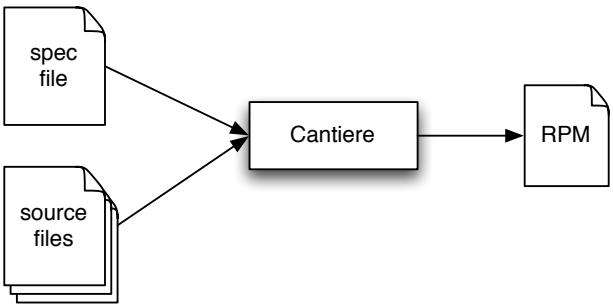

Figure 3.1: Cantiere is used to build RPM packages

A spec file, short for specification file, is a file that describes how a package is built. The RPM Guide explains how to build spec files [24]. A spec file contains the following sections:

Introduction The introduction section contains meta informations about the package, such as the name and the description of the package.

Prep The prep section contains the commands necessary to prepare the build of the package.

Build The build section contains the commands to build the software.

Install The install section contains the commands that install the newly built software.

Clean The clean section cleans up the files created during the building process.

Files The files section indicates which files will be included in the package.

The sources files can be URL or archives (for instance the program that is packaged), scripts files (for instance to control the service that is packaged), or whatever needs to be packaged.

3.2 BoxGrinder

BoxGrinder is an application that creates appliances from appliance definition files (see Figure 3.2).

The appliance definition file is a very short text file which describes the content of the appliance.

An appliance is basically a virtual machine, with an OS and with some applications preinstalled and preconfigured. The appliance also specifies the minimal hardware configuration required. BoxGrinder supports many virtualization and cloud platforms such as Amazon EC2, ElasticHosts clouds and VMware.

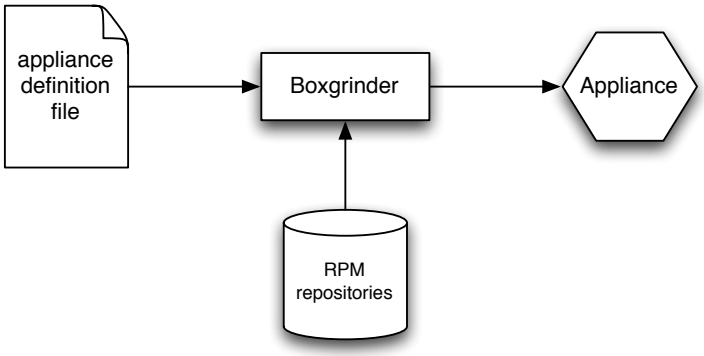

Figure 3.2: Boxgrinder build appliances from appliance definition files.

3.2.1 Appliance definition

The appliance definition file is written in YAML [7]. It contains the following sections:

Prolog The prolog section contains the name of the appliance, the version number and a summary of its content.

OS The OS section describes the OS that will be installed on the appliance.

Hardware The hardware section describes the minimal hardware configuration required in order to run this appliance.

Appliances An appliance definition file can inherits from one or more other appliance definition files. This section describes the appliance definition files that this definition inherits from. All the parameters are inherited, but this definition file will overwrite the parameter it defines.

Packages This section describes all the RPM packages that will be installed on the appliance. The packages must be present in the repositories of the host during the building process (see Section 3.2.2), or in the repositories listed in the repos section of this appliance definition file.

Repos The section can be used to list additional RPM repositories.

Post This section can be used to indicate commands that should be executed on the appliance just after the build.

3.2.2 Building process

With BoxGrinder, it is only possible to build appliances with the same OS as the OS on which the appliance is built. Likewise, it is only possible to build appliances of the same architecture (i386 or x86_64) as the architecture of the machine the appliances are built on. BoxGrinder can be used to build Fedora, Red Hat Enterprise Linux (RHEL) or CentOS appliances.

The easiest way to build an appliance with BoxGrinder is to use the BoxGrinder Meta appliance. This meta appliance contains all the tools that are needed to build an appliance. BoxGrinder Meta appliance is a Fedora instance, so it will allow the user to built only Fedora appliances.

The building process is actually a pipeline, as illustrated by the Figure 3.3. The first step of the pipeline is the building strictly speaking. The building produces a raw file. The second step is the conversion to a platform format. The conversion produces an image dependent of the platform the appliance is built for. The third and last step is the delivery of the appliance. The appliance can be delivered locally, on a sFTP, on Amazon S3, on Amazon S3 and be registered as an AMI, or on Amazon EBS and be registered as an AMI.

The conversion and delivery steps are implemented as plugins. So it is easy to extend BoxGrinder in order to support more platforms or delivery targets.

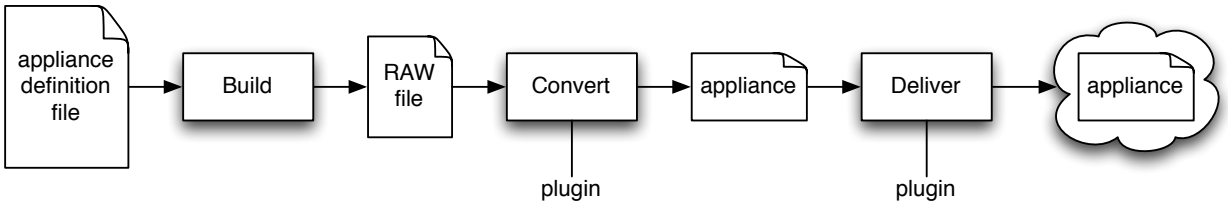

Figure 3.3: BoxGrinder's building process is a pipeline of 3 steps: building, conversion and delivery.

3.2.3 Supported platforms

BoxGrinder supports the following platforms:

- VirtualBox [40]
- VMWare [61]
- Amazon EC2 [16]
- ElasticHosts [3]
- SKALI Cloud [27]
- Open Hosting [38]
- Serverlove [34]

- CloudSigma [13]

The support for Eucalyptus in under development. There is no plan to support OVF [58] yet.

3.3 CirrAS

CirrAS is an effort to automatically deploy a clustered JBoss AS server in the Cloud [42]. CirrAS is a set of appliances to make JBoss AS 6 deployment in the cloud easier (see Figure 3.4).

Front-end appliance This appliance provides an Apache HTTP server and mod_cluster [45], a HTTP load balancer. Besides, an RHQ agent is installed on this appliance.

Back-end appliance This appliance provides JBoss AS 6. Besides, an RHQ agent is installed on this appliance.

Management appliance This instance provides a RHQ server for management (see Section 2.3). The agents that run on the frond-end and back-end appliances will automatically detect the management appliance and they will register, so there is nothing to configure.

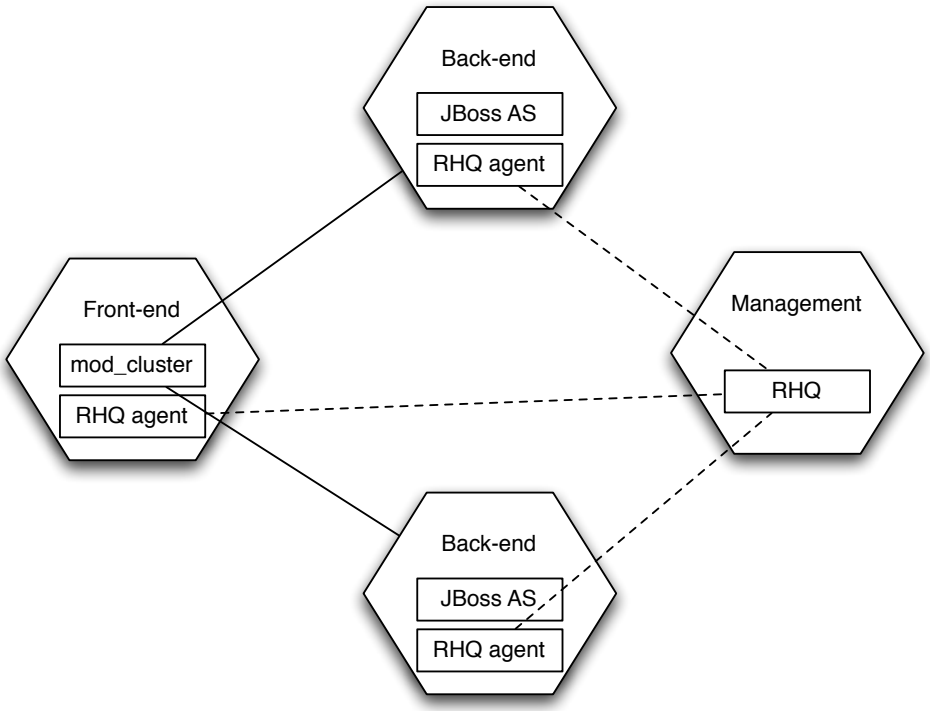

Figure 3.4: Example of a CirrAS deployment with one front-end, two back-ends, and one management instances.

CirrAS is not longer maintained. Instead, SteamCannon has been developed (see next Section).

3.4 SteamCannon

SteamCannon is a PaaS provider [49]. SteamCannon helps the user manage and deploy her platform. SteamCannon supports the JBoss AS platform out of the box.

SteamCannon as been developed as a proof of concept by employes of RedHat. SteamCannon will no longer be developed. Instead, RedHat plans to commercialize OpenShift [46]. Basically, OpenShift will provide the same functionalities as SteamCannon, except that OpenShift will be a hosted service (the user will not have to install an instance of OpenShift in her cloud).

SteamCannon is developed in Ruby. SteamCannon ships with Torquebox [51]. Torquebox is an extended JBoss AS version which aims to provide the application server abilities of JBoss AS for applications developed in Ruby and Ruby on Rails.

3.4.1 Usage

SteamCannon allows the user to create an environment from a platform. For each image of the platform that is set as scalable, the user can choose how many instances the environment should initially contain.

The user can then start or stop the environment. Starting or stopping an environment mean starting or stopping all the instances of this environment. If the platform contains an image set as scalable, the user can start more instances of this image within the environment. She can also stop one or more instances of the environment.

The user can also execute some operations over a service that is provided by an instance. The basic operations a service provides are start or stop the service, and see the logs of the service. A service may provide more operations.

Finally, the user can deploy an artifact on the environment. The artifact will be deployed on each service, if the service supports this artifact.

3.4.2 Architecture

SteamCannon provides an agent-server architecture. The SteamCannon server communicates with all the SteamCannon agents within a cloud. The SteamCannon agents are responsible for managing the services provided by the instances they are running on. The Figure 3.5 shows an example of a running environment.

At the moment, SteamCannon only runs on Amazon EC2 clouds. However, SteamCannon is based on the Deltacloud API [1]. So it would be possible to run SteamCannon on other cloud providers than Amazon EC2 that support the Deltacloud API, but that would require some work in order to adapt SteamCannon.

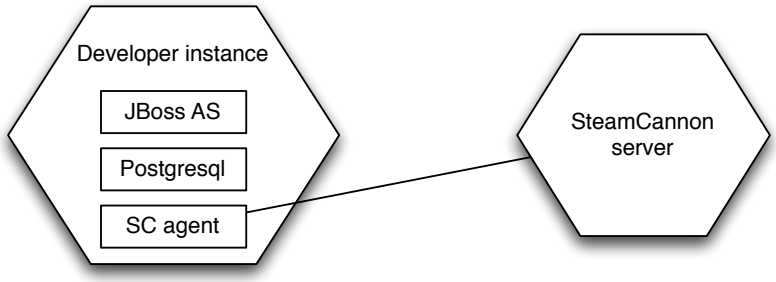

Figure 3.5: Example of the deployment of a JBoss AS Developer environment. This platform is provided by default by SteamCannon.

Chapter 4

SIP servlet

In this chapter, we will first introduce the SIP protocol.

Then, we will discuss the SIP Servlet specification. This specification allows the developers to build web applications that are able to handle SIP messages.

We will also discuss Mobicents Sip Servlets, which is an open-source implementation of the SIP Servlet specification.

Finally, we will briefly discuss SIPp, a SIP traffic generator.

Contents

4.1 The SIP protocol

SIP stands for Session Initiation Protocol. This protocol aims to establish a session between two end points. Once the session is established, several types of data can transit through the session. SIP is mostly used for controlling multimedia communication sessions such as voice calls over Internet Protocol (VoIP). SIP is specified by RFC 3661 [54].

4.1.1 Protocols stack

SIP stands in the session layer of the OSI model [8]. Therefore, SIP can be used over TCP or UDP. However, SIP implements an acknowledgment system [21], so UDP suits well.

Since SIP only deals with the session establishment, other protocols may be used in order to process the media data, such as RTSP. RTSP (for Real-Time Streaming Protocol) is a protocol designed to control streaming media servers [57].

Note that SIP is not a secure protocol. So SIP may be encapsulated in a protocol such as TLS in order to achieve encryption and authentication.

4.1.2 SIP messages

There are two sorts of SIP messages: SIP requests and SIP answers.

The request has the following form:

```
<request_name> <sip_url> <version>
{<param>: <value> CRLF}*
```

The `request_name` is one argument taken from the Table 4.1. The `sip_url` is discussed in Section 4.1.3. The `version` indicates the version of the protocol used by this agent.

Next, there are some mandatory and optional parameters. A parameter is composed of its name, a colon, its value and a carriage return. The mandatory parameters for all requests are `Via`, `Max-Forwards`, `From`, `To`, `Call-ID` and `CSeq` [21].

The answer is composed of a numerical code, as in HTML. The Table 4.2 lists the categories of those codes.

4.1.3 SIP agents

A SIP user agent (UA) is a network end-point which sends and receives SIP messages. The SIP UA can perform the role of an User Agent Client (UAC), which sends SIP requests, or an User Agent Server (UAS), which receives the requests and returns an answer. However, a SIP UA usually performs both roles in the same session.

Request name	Description
INVITE	Indicates a client is being invited to participate in a call session.
ACK	Confirms that the client has received a final response to an INVITE request.
BYE	Terminates a call and can be sent by either the caller or the callee.
CANCEL	Cancels any pending request.
OPTIONS	Queries the capabilities of servers.
REGISTER	Registers the address listed in the To header field with a SIP server.
PRACK	Provisional acknowledgment.
SUBSCRIBE	Subscribes for an Event of Notification from the Notifier.
NOTIFY	Notify the subscriber of a new Event.
PUBLISH	Publishes an event to the Server.
INFO	Sends mid-session information that does not modify the session state.
REFER	Asks recipient to issue SIP request (call transfer).
MESSAGE	Transports instant messages using SIP.
UPDATE	Modifies the state of a session without changing the state of the dialog.

Table 4.1: SIP requests.

Response	Code	Description
Provisional	1xx	Request received and being processed.
Success	2xx	The action was successfully received, understood and accepted.
Redirection	3xx	Further action needs to be taken (typically by sender) to complete the request.
Client error	4xx	The request contains bad syntax or cannot be fulfilled at the server.
Server error	5xx	The server failed to fulfill an apparently valid request.
Global failure	6xx	The request cannot be fulfilled at any server

Table 4.2: SIP responses.

A SIP proxy is an agent which stands in the path of two UA. The proxy is used only for the `INVITE` request and answer. The following `ACK` is then sent directly from one UA to another. The main purpose of the SIP proxy is to route the INVITE request between the UA's.

A SIP registrar is used in order to make a link between a `sip_url` and the IP address of the user. First, the user has to register to the registrar. Then, she obtains a `sip_url`, which has the form `username@registrar`. So, each time Alice wants to connect, she has to send a `REGISTER` request to her registrar. If Bob wants to call Alice, he will contact Alice's registrar in order to retrieve her IP address. He will then send a `INVITE`

request to her IP address. There exist some free registrars on the internet, such as [12].

4.1.4 Example

UAC and UAS

UAC and UAS are the roles that are the more often used by SIP agents. This simple example shows how an UAC and an UAS initiate a dialog and conclude it. This example is illustrated by the Figure 4.1

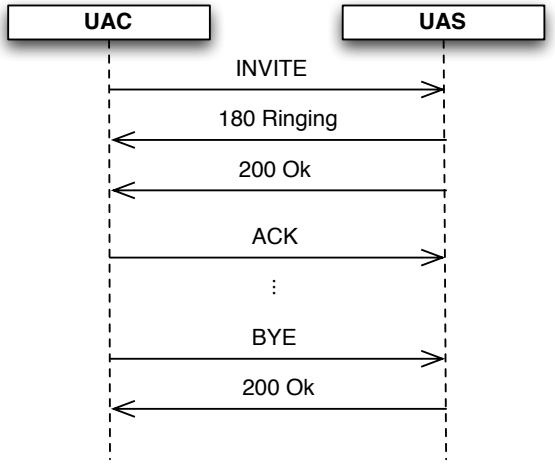

Figure 4.1: First, the UAC tries to open a SIP dialog with the UAS by sending the INVITE request. Then, the UAC closes the dialog by sending a BYE request.

Proxy

The Figure 4.2 shows an example of a proxy [54]. The SIP registers often act as proxy. First, a client (the UAS in this example) register with its registrar, so the registrar is aware of the IP address of the client (not shown here). When another client (the UAC in this example) wants to create a dialog with the UAS, it will send an INVITE request to the registrar, which it knows thanks to the SIP address of the UAS. Since the registrar knows the IP address of the UAS, it will forward the request to it.

B2BUA

The back to back user agent (B2BUA) is a common case for SIP application [54]. The B2BUA acts as an endpoint for two other agents and forwards requests and responses between those two agents in some way. As shown by the Figure 4.3, the requests and responses are not always symmetrical.

Unlike a proxy, a B2BUA is used in the situation where this agent must be aware of all the SIP messages of the dialog. Indeed, the role of a proxy is to forward some SIP messages of a dialog, while the role of a B2BUA is either to provide a service based on all the messages of the dialog or to alter some messages of the dialog.

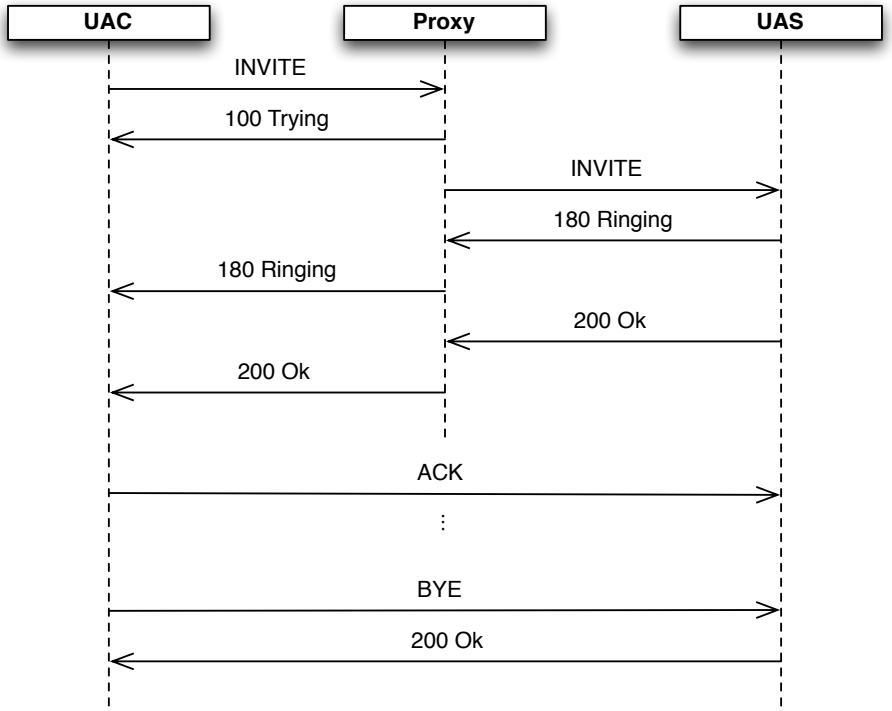

Figure 4.2: The proxy is only used until the ACK request.

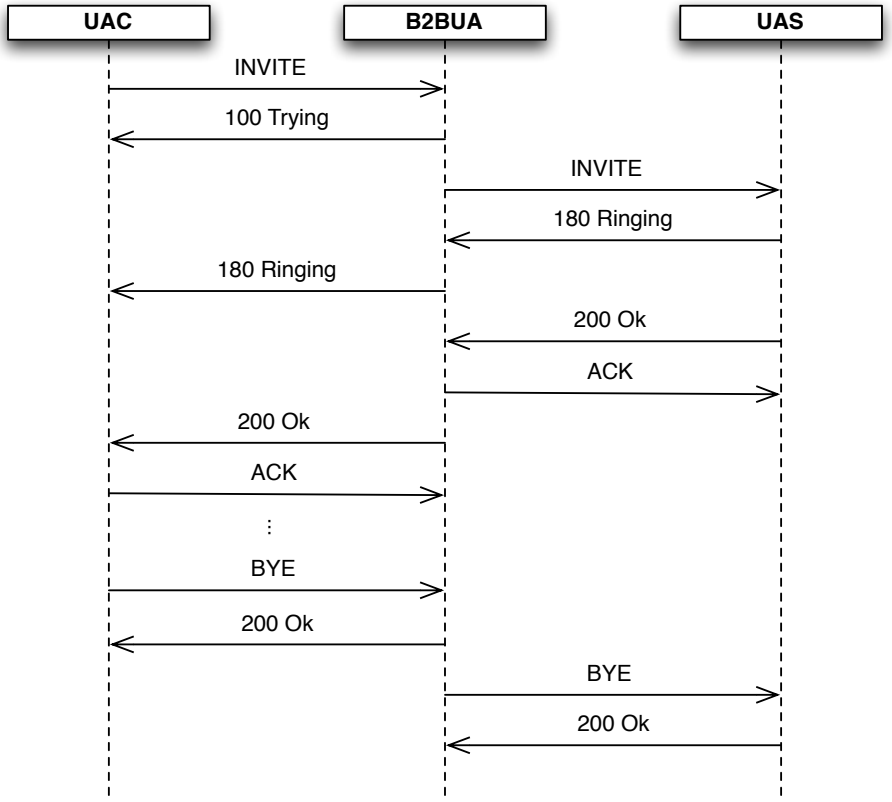

Figure 4.3: The B2BUA acts as an UAS for the UAC, and vise versa.

An example of an application which is built on a proxy is the call forwarding application. Thanks to this application, Bob can ask his calls to be transferred on Carol's phone. When Alice will try to reach Bob, the call forwarding application will transfer the call to Carol's phone. An example of application which is built on a B2BUA is the log application. The log application logs every message belonging to a dialog. It would not be possible to build this application on a proxy, since all the messages must transit by the application.

4.2 The SIP Servlet specification

SIP Servlet [31] is a specification part of Java EE. The specification defines SIP servlets, which are comparable to HTTP servlets. It is possible to build an web application which handles SIP requests, or which handle both SIP and HTTP requests. The latter is called a converged web application. The Figure 4.4 illustrates a converged web application, a web application and a SIP application.

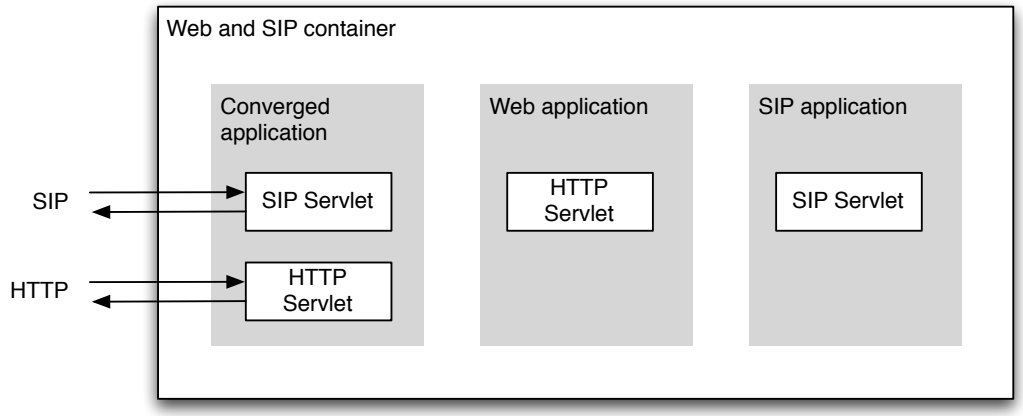

Figure 4.4: SIP Servlets are some kind of Servlets

4.2.1 SIP Servlet

A converged web application is composed of one or more HTTP Servlets and one or more SIP Servlets. SIP Servlets must be executed within a SIP Servlets Container, which implements the SIP Servlet specification.

However, there exist some differences between HTTP servlets and SIP servlets [9]. The Figure 4.5 illustrates those differences.

- HTTP servlets typically return HTML pages to clients, while SIP servlets typically connect SIP-enabled clients. SIP servlets can be seen as a link between two SIP devices.
- SIP is a peer-to-peer protocol and SIP servlets can originate SIP requests, while HTTP is a client-server protocol and HTTP servlets cannot originate requests.

- SIP servlets can generate multiple responses for one request.
- There are often several SIP servlets that handle the same SIP request, while a HTTP request is handled by an unique HTTP servlet.

(a) HTTP Servlet (b) SIP Servlet

Figure 4.5: SIP servlets handle a peer-to-peer protocol while HTTP servlets handle a client server protocol.

4.2.2 Sessions

SIP servlets are stateless. It means they do not store data across several SIP requests. The goal of a session is to associate SIP requests and data stored by the SIP Servlets container. There exist two sorts of sessions: SIP sessions and SIP application sessions.

SIP sessions store data of a single SIP servlet instance for a single SIP agent. A SIP session represents a point to point SIP relationships, and is roughly equivalent to a SIP dialog.

SIP application sessions, on the other hand, store data of all HTTP servlets instances and SIP servlets instances, for one or several clients. A SIP application session contains several SIP sessions and HTTP sessions. SIP applications sessions should have been called application sessions, because they are actually multi-protocol.

A session (SIP session or SIP application session) can be manually invalidated if it is not needed anymore. Otherwise, a timer will invalidate the session after a configured idle time.

4.2.3 The application router

A container may contain several converged applications. Each application may be interested in processing a SIP request received by the container. The applications process the SIP requests in a sequential order, a SIP request can be processed by only one application at a time. The Figure 4.6 illustrates a SIP request processed by several SIP applications.

When an initial request hits the container, the role of the application router is to determine the order in which the applications will process the request. After an application has processed a request, the request is sent back to the application router in order to determine the next application that will process this request. The Figure 4.7 illustrates the path of a SIP request.

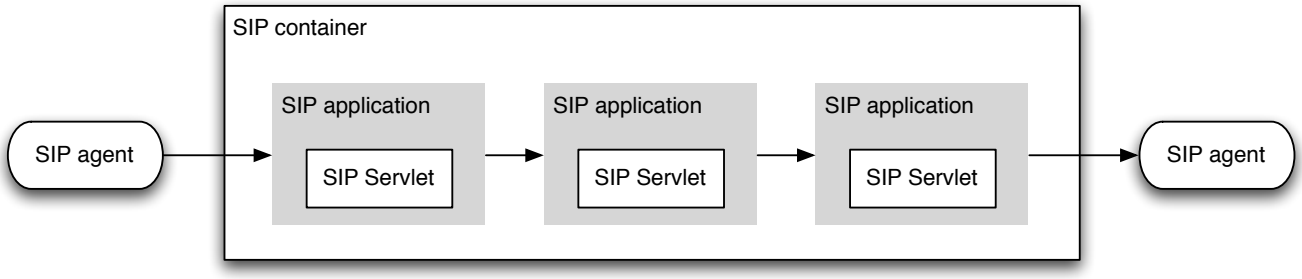

Figure 4.6: Several applications process a SIP request.

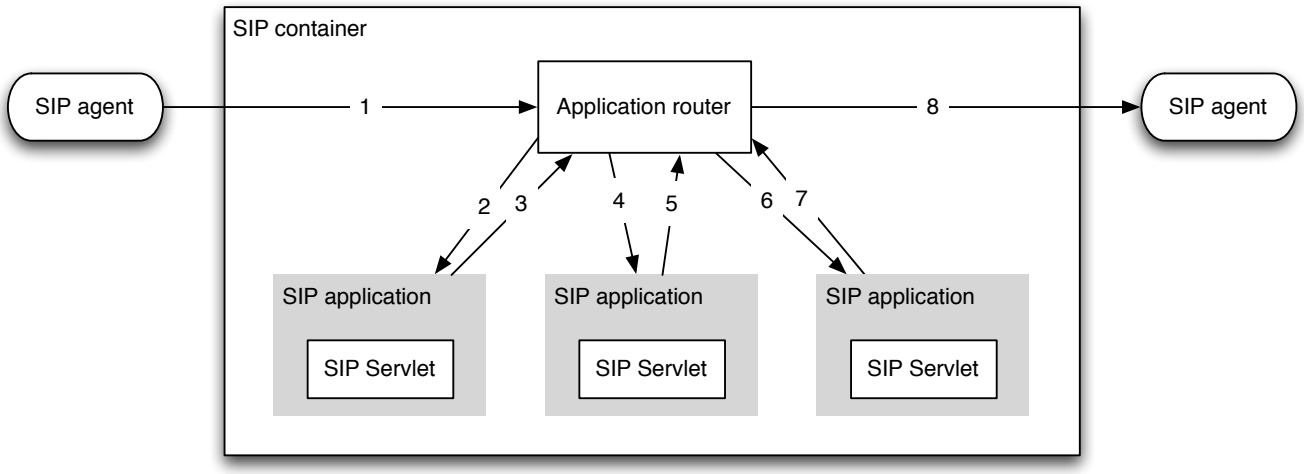

Figure 4.7: The application router forwards the SIP request among the applications.

When a request belonging to a dialog hits the container, the role of the application router is to ensure that the request follows the same path that the initial request of the dialog (or the reverse path if the request comes from the other end of the line).

When an application processes a SIP request, it can decide to do nothing, to relay the request by acting like a proxy or a B2BAU, or to consume the request by acting like an UAS. The decision of the application is sent back to the application router, so the application router can decide which is the next application that will process this request.

The application router needs to be configured in order to know which applications are interested by which SIP requests. When an application subscribes to the application router, it has to indicate in which region the application sits: originating region, neutral region, or terminating region. An application registers to the originating region if it wants to process SIP requests that are received by the container. On the contrary, it registers to the terminating region if it wants to process SIP requests that are sent by the container. Application that are registered in the neutral region will process the received and sent SIP requests.

4.2.4 Proxy and Back to back user agent

Proxy and back to back user agent are two very common scenarios. Thus, SIP Servlet includes some helper in order to build such applications.

4.2.5 Example of a converged application

This section will show an example of a converged application. This converged application is called *the call schedule on busy or no answer service* (CSBNA) [31]. This example aims to show how HTTP and SIP protocols can be used together. In order to keep the example simple, some messages are not shown here.

Let's say an user, Alice, wants to call another user, Bob. Let's say that the CSBNA service is installed on an application server which sits on the route to Bob. Finally, let's say that Bob is not available for the moment. The Figure 4.8 illustrates the messages that are sent. Here is the explanation of the messages that are sent:

1. When Alice wants to call Bob, she sends an INVITE SIP request to the CSBNA. Since Bob is not available, the CSBNA issues a SIP response which contains an URL. The URL points to a web page where Alice can ask Bob to call her back. The web page is handled by an HTTP servlet which is part of the CSBNA application.
2. Alice goes to the web page and thus sends an HTTP GET request to the CSBNA. The CSBNA issues an HTML page with a form that contains the SIP addresses of both Alice and Bob, and sends back the result to Alice.
3. Alice confirms and posts the form by sending a HTTP POST request to the CSBNA. The CSBNA sends a SIP SUBSCRIBE request to Bob and sends back an HTML confirmation page to ALICE.
4. A some point, Bob becomes available and sends a SIP NOTIFY request to the CSBNA.
5. Finally, the CSBNA initializes a call between Alice and Bob by sending a SIP INVITE request to both.

4.3 Mobicents Sip Servlets

Mobicents Sip Servlets is an implementation of the SIP Servlet specification on top of JBoss AS 5.1 [44].

4.3.1 The application router

Mobicents Sip Servlets ships with a default application router (DAR). The application router is a separate logical entity from the SIP Servlets container. The application router is responsible for application selection and does not implement application logic.

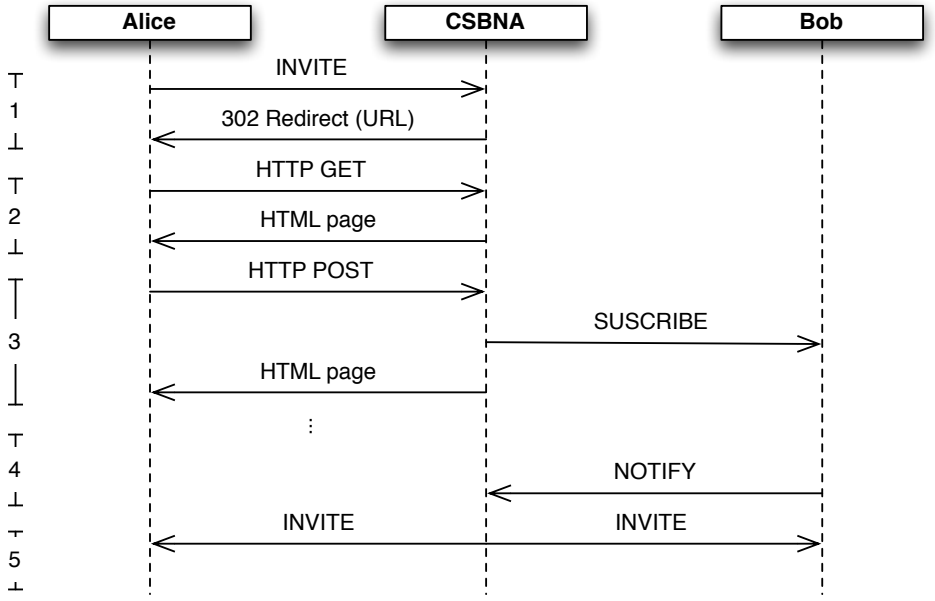

Figure 4.8: Example of the call schedule on busy or no answer service.

4.3.2 Clustering

Since Mobicents Sip Servlets is built on top of JBoss AS, it supports the clustering capabilities of JBoss AS. The Figure 4.9 illustrates a Mobicents Sip Servlets cluster.

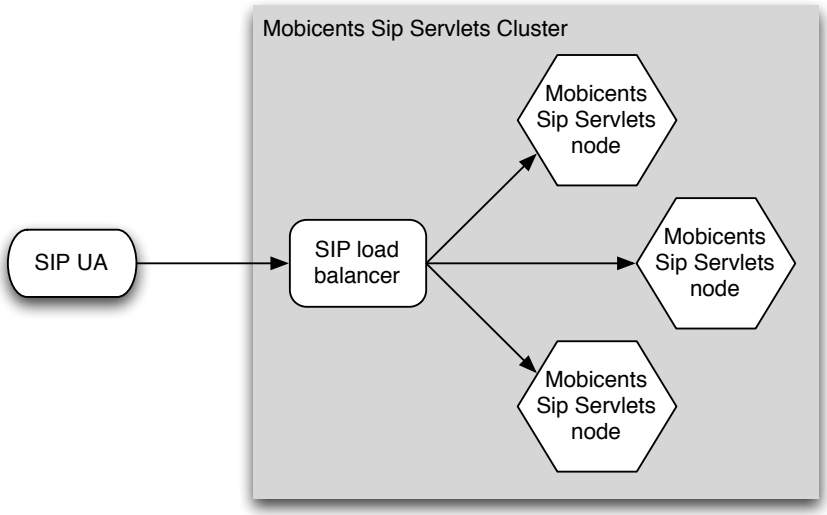

Figure 4.9: The SIP load balancer is used in order to distribute the messages from the SIP UA's among the Mobicents Sip Servlets nodes.

Load balancing

A load balancer is needed in order to balance the traffic among the nodes of the cluster. An IP load balancer can be used. However, it is recommended to use an HTTP and SIP load balancer, since this balancer should be able to provide sticky sessions.

The Section 4.3.3 discusses the Mobicents SIP load balancer.

Replication

In order to provide failover, some replication is needed. If the application is set as distributable, SIP sessions and SIP application sessions, like HTTP sessions, are replicated on all nodes. So, any node can serve any request to any agent.

As explained in Section 2.2.2, passivation is the process that consists in removing unused sessions from memory and store them in persistent storage. Mobicents Sip Servlets fully supports the passivation process for SIP sessions and SIP application sessions.

Failover

Mobicents Sip Servlets supports two failover modes: established SIP dialog failover and early SIP dialog failover [36]. As explained in Section 2.2.2, failover is the fact that a healthy node takes over a failed node.

In the established SIP dialog failover mode, failover can occur only on established calls (SIP dialogs which are in the CONFIRMED state as per RFC 3261 [54]). Calls that are in the process of being setup will not be failed over (SIP dialogs which are in the EARLY state as per RFC 3261 [54]). So the failover can occur in the states from 9 to 12 in the Figure 4.10.

In the early SIP dialog failover mode, failover can occur after an informational response (1xx) is received with a To Tag (SIP dialogs which are in the EARLY state as per RFC 3261 [54]). So the failover can occur in the states from 5 to 12 in the Figure 4.10.

The failover is actually done by the SIP load balancer. The nodes must register themselves to the SIP load balancer, and send it regularly a heartbeat. Thus, the SIP load balancer is aware of the alive nodes on the cluster. When a node does not send a heartbeat, all the messages that were belonging to it are routed to the other nodes. The failover uses the fact that any node can serve any request to any agent. In order to support failover, an application must be set as distributable, so its sessions are replicated on all nodes.

The early SIP dialog failover can introduce some overhead in terms of replicating transaction states. It is thus recommended to use the established SIP dialog failover mode.

4.3.3 SIP load balancer

Mobicents Sip Servlets provides a SIP load balancer. The Mobicents SIP load balancer acts as an entry-point for the cluster. The SIP load balancer can work together with an HTTP load balancer such as mod_jk, or the SIP load balancer can handle HTTP traffic as well.

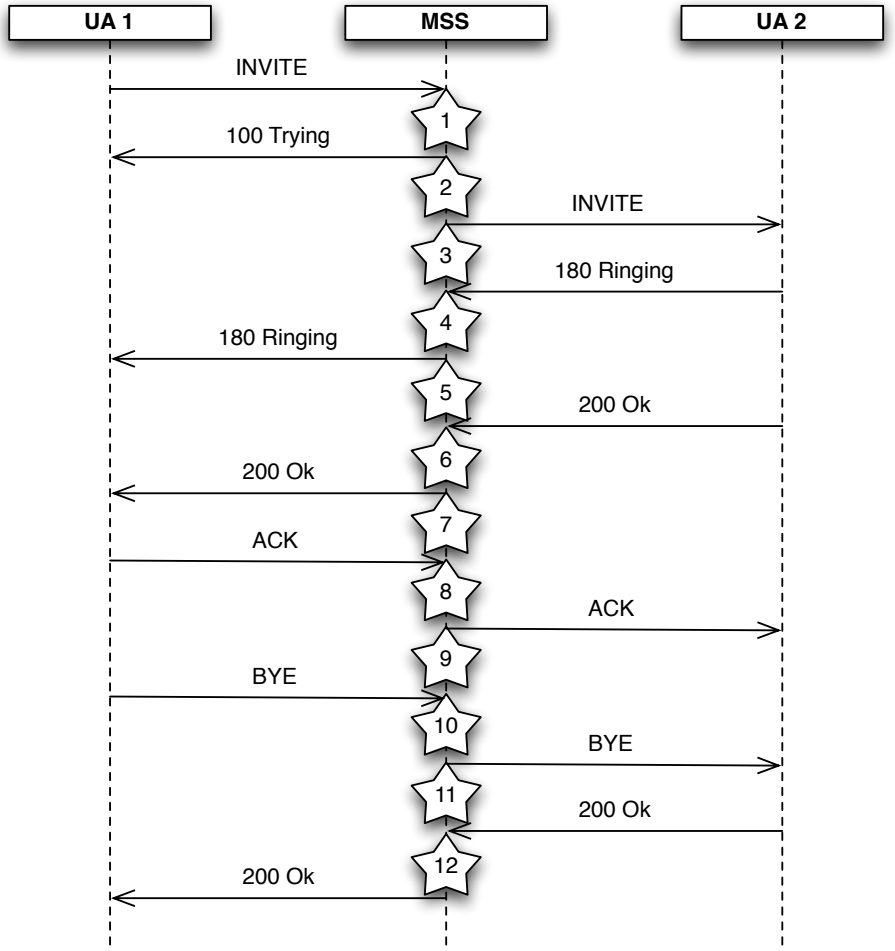

Figure 4.10: This figure represents Mobicents Sip Servlets acting as a B2BUA. The established SIP dialog failover can occur in steps from 9 to 12. The early SIP dialog failover can occur in steps from 5 to 12.

Topology

The SIP load balancer is aware of the status of the nodes by sending them regularly a heartbeat. The load balancer distributes the SIP messages among the alive nodes.

When a SIP message is processed by an application, it passes first by the SIP load balancer then by the Mobicents Sip Servlets instance. When the application send a SIP message, the message passes through the SIP load balancer. Hence, the load balancer appends itself to the Via header of each request. Thus, responses are sent to the SIP load balancer before they are sent to the originating SIP application.

The SIP load balancer uses two SIP ports: an external port and an internal port. The external port (5060 by default) is used for messages received from outside the cluster, and the internal port (5065 by default) is used for messages received from inside the cluster (see Figure 4.11). Those two ports are necessary because the routing decision is different depending on the source of the message.

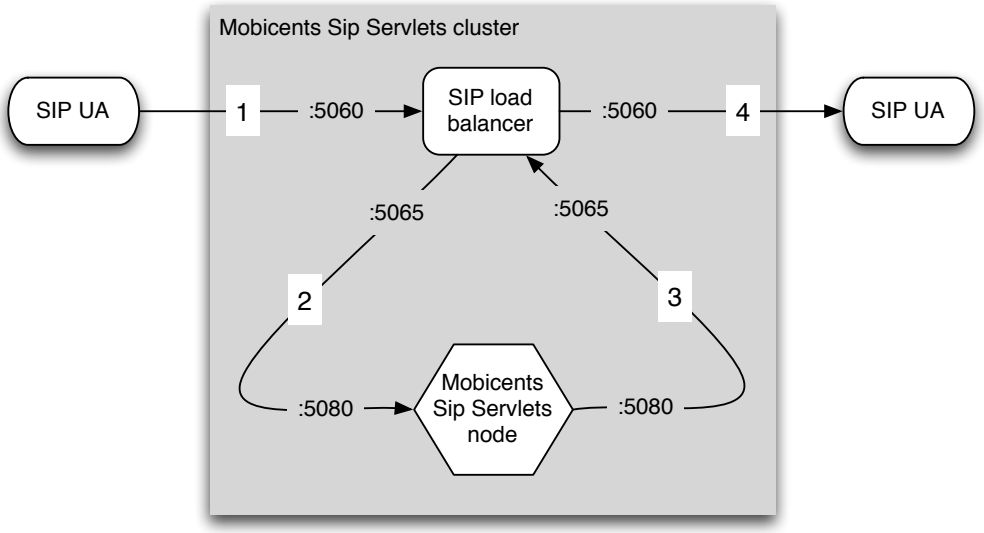

Figure 4.11: The SIP load balancer processes two times a SIP message: once when the message goes in the cluster, and once when the message goes out the cluster.

Sticky sessions

The SIP load balancer provides sticky SIP sessions. It means that all requests belonging to the same SIP dialog will be routed to the same node.

If the SIP load balancer is used with mod_jk and if an HTTP session is bound to a particular node, the SIP load balancer will automatically send the SIP messages from the same application session to the same node.

If the SIP load balancer is used to balance HTTP traffic as well, the HTTP messages and SIP messages that belong to the same application session are sent to the same node.

Routing algorithms

The SIP load balancer exposes an interface to allow users to build a customized routing algorithm. By default, the SIP load balancer proposes 4 different routing algorithms. The Table 4.3 shows the differences between these algorithms.

Call-ID affinity The routing decision of this algorithm is based on the Call-ID field of the SIP requests and responses. The load balancer keeps in memory a map associating the Call-ID value with the node ID. This algorithm behaves exactly like mod_jk for HTTP messages. Since this algorithms is stateful, it is not distributable.

Header consistent hash This algorithm achieves sticky sessions by hashing specific headers from SIP and HTTP messages and always linking the same hash to the same node. If new nodes are added to the cluster, the hashing space is redistributed and messages can be routed to another nodes. Since this algorithm is stateless, it can be used as a distributable algorithm.

Persistent consistent hash This algorithms is similar to the Header consistent hash algorithm, except that it keeps sessions affinity when a node is added or removed from the cluster. This algorithm is also distributable.

Cluster subdomain affinity This algorithm support grouping server nodes to act as a sub-cluster. Any call of a node that belongs to a cluster group will be preferentially failed over to a node from the same group. Otherwise, this algorithm behaves exactly like the Call-ID affinity algorithm.

	Call-ID affinity	Header consistent hash	Persistent consistent hash	Cluster subdomain affinity
Sticky sessions	✓	✓	✓	✓
Distributable		✓	✓	
New node consistency	✓		✓	✓
Sub-clusters				✓

Table 4.3: Capabilities of each algorithm provided with the SIP load balancer.

Using a single SIP load balancer can lead to a single point of failure. When a single SIP load balancer is not enough to process all the traffic, there should be multiple SIP load balancers. If multiple SIP load balancers use the same distributable algorithm, they will all make the same routing decision for the same SIP request. Therefore, a SIP request can be sent to any SIP load balancer. In this configuration, an IP load balancer is used in order to balance the traffic among the SIP load balancers, as illustrated by the Figure 4.12.

4.4 SIPp

SIPp is a SIP test tool and traffic generator [6].

SIPp ships with several scenarios such as an UAC and an UAS. However, it is possible to define new scenarios via XML files. The Figure 4.13 illustrates the UAC and UAS scenarios.

SIPp can be used to test many real SIP equipments like SIP proxies, B2BUAs, SIP media servers, SIP/x gateways, SIP PBX, ... It is also very useful to emulate thousands of user agents calling a SIP system.

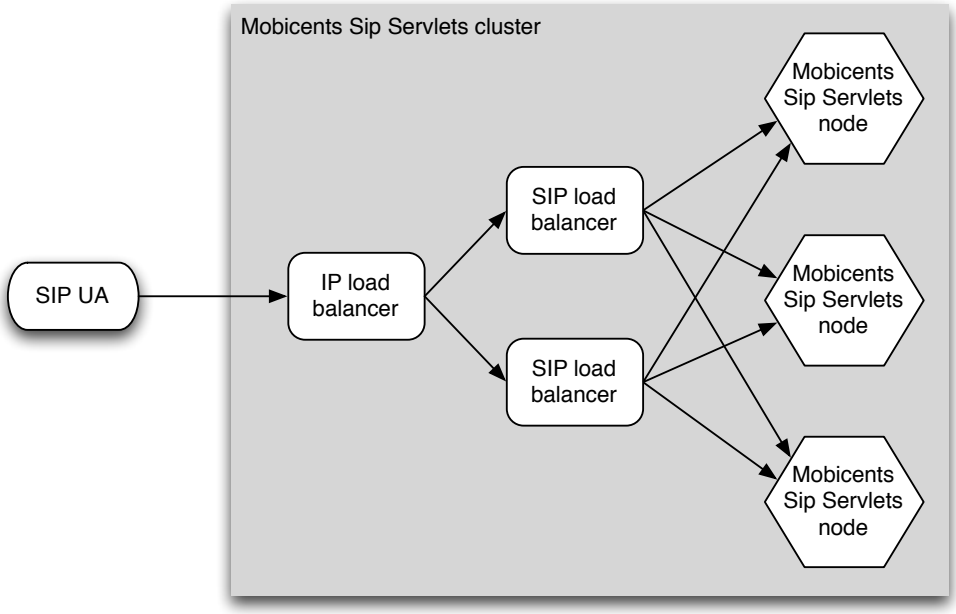

Figure 4.12: The IP load balancer distributes the traffic among the SIP load balancers. The SIP load balancers distributes the traffic among the Mobicents Sip Servlets nodes.

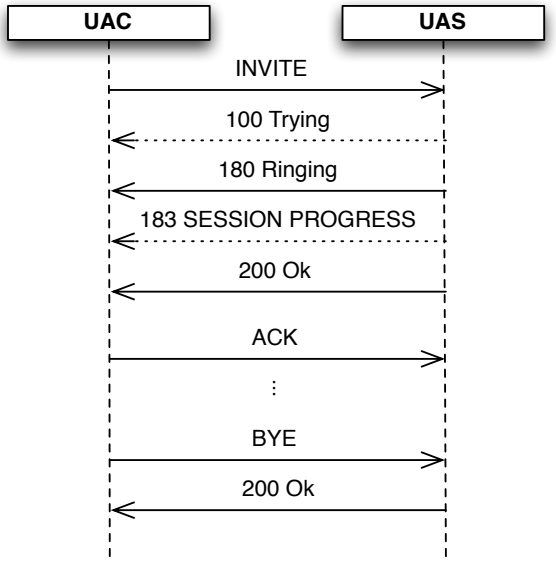

Figure 4.13: Two SIPp agents talking together. A SIPp agent is acting as UAC while the other is acting as UAS. The dashed lines indicate an optional message.

Part II

Analysis of the problem

Chapter 5

Mobicents Sip Servlets in a cluster

The subject of this thesis is focused on Mobicents Sip Servlets. This chapter will present the current situation of Mobicents Sip Servlets running in a cluster.

Then, we will present the problem that arise in a Mobicents Sip Servlets cluster and we will briefly evoke a solution.

Contents

5.1 Situation

In the context of a Mobicents Sip Servlets cluster, an user is defined as the person who uses Mobicents Sip Servlets. The user has developed some SIP applications or converged applications, and she wants those applications to run on top of Mobicents Sip Servlets. Later on, the term application will refer to either SIP application or converged application.

Typically, an application will send and receive SIP messages, HTML messages, and maybe some other messages as well. Later on, the term traffic will refer to the flow of messages that are processed by the application.

Mobicents Sip Servlets is used to build real time systems, such as VoIP softwares. The user wants her application to be able to handle a large amount of traffic in a very short time.

Some applications may face a huge amount of traffic. Those applications will require a consequent computation power in order to process all this traffic. In order to ensure a quick response time, a single machine may not be able to handle all the traffic by itself. In this case, the application can be deployed on a cluster.

Besides, telecommunications solutions often require a 5 9 availability (i.e. 99.999% of availability). Thus, a cluster is required since a single server would be a single point of failure.

As mentioned is the Section 2.2.2, a cluster is a set of machines on which the same application is deployed. A typical architecture of Mobicents Sip Servlets in a cluster is shown at Figure 5.1. A SIP load balancer (see Section 4.3.3) receives the traffic from outside and distributes it to the nodes of the cluster. A more sophisticated configuration would contain several SIP load balancers.

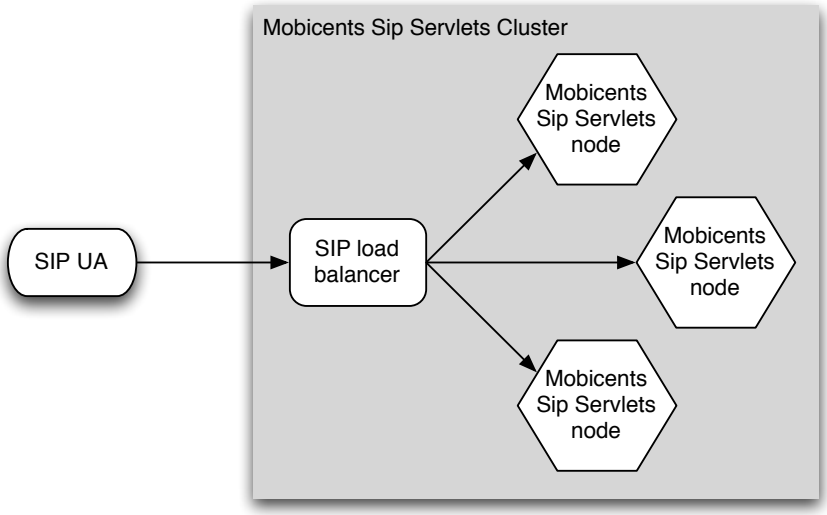

Figure 5.1: Mobicents Sip Servlets is usually executed in a cluster.

The JBoss AS cluster provides three properties: elasticity, high availability and failover. Since Mobicents Sip Servlets is built on top of JBoss, those properties are also

ensured by the former. Those properties ensure that it is possible to add or remove nodes at runtime in the cluster. The new nodes are automatically added to the cluster and are responsible for processing a part of the traffic. If a node fails or is removed, the traffic that was handled by this node is redirected to the others nodes of the cluster.

5.2 Problem

The problem that may occur is that the number of clients of the user's application may vary considerably in a short period of time. At some time in the day, there might be a lot of clients, and at some other time, there might be a few clients. As a result, some resources of the cluster in which the application is running may be unused, or on the contrary, the cluster may be overloaded by the amount of traffic it has to process.

A first solution would be to add or remove nodes in the cluster depending of the needs. In order to do so, one would have to monitor the status of each node of the cluster. The monitoring process would then decide if some nodes need to be added or removed, and would do so.

However, this solution has two drawbacks.

First, it is hard to predict the amount of traffic the application will meet. As a consequence, it is also hard to predict the amount of resources that should be added to or removed from the cluster. Such a prediction would be useful, as it is needed to have those resources available in order to put them in the cluster.

Secondly, the resources that are not currently used in the cluster are not costless. Indeed, the user need to own those resources in order to be able to add them quickly in the cluster.

In other words, the problem that a Mobicents Sip Servlets cluster is facing is the waste of unused resources (because they are reserved in case of higher load), or on the contrary, the overload of the cluster (because no other resource is available).

5.3 To the clouds

The solution that would address this problem is to take advantage of a cloud. Thus, we would run Mobicents Sip Servlets on a cloud.

As described in Chapter 1, a cloud does not have those drawbacks. Indeed, a cloud allows theoretically to have an infinite pool of resources available. This is the elasticity property of a cloud. Besides, if some resources are not needed anymore, it is easy to shut them down, and they become costless. This is the pay per use property of a cloud.

Chapter 6

Use cases

In order to analyze the problem and validate the solution, we have defined two use cases of Mobicents Sip Servlets. Those use cases are applications that may face a large and variating amount of traffic.

We have implemented those uses cases and published them to the Mobicents code repository. It is thus easy to deploy them.

This chapter presents the use cases. The Appendix A.5 explains the implementation of the use cases. The Appendix B.3 explains how to install and launch the use cases.

Contents

6.1 The proxy use case

The first use case is to use Mobicents Sip Servlets as a proxy. The role of a proxy is defined in Section 4.1.

We use two SIPp agents. One agent is acting as an UAS and receives messages from the proxy. The other agent is acting as an UAC and sends messages to the proxy. Thus, the proxy stands in the path of the two agents. The Figure 6.1 shows the path that is followed by the messages, and the Figure 6.2 shows the flow of messages that are sent between the agents.

Usually, the proxy is used only for the INVITE request and response. The ACK request and all the following messages should be sent from one agent to another. However, because of a limitation of SIPp which prevent it to send a message to different host during the same session, all the messages are sent though the proxy.

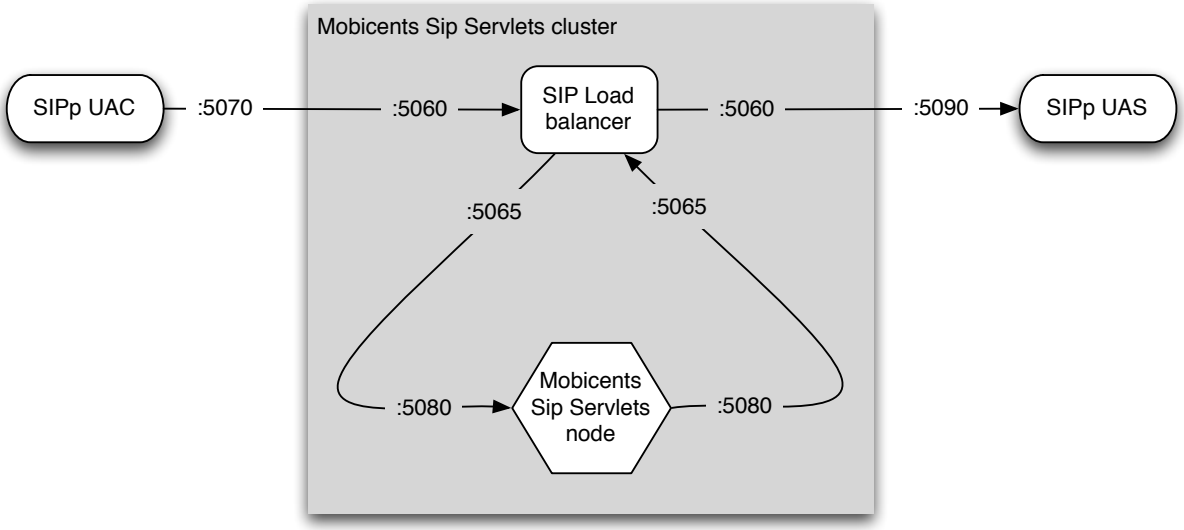

Figure 6.1: All the entities run on a separate machine. The ports on which are sent and received the messages are indicated for each entity.

6.2 The B2BUA use case

The second use case is to use Mobicents Sip Servlets as a back to back user agent (B2BUA). The role of a B2BUA is defined in Section 4.1.

The configuration is basically the same than in the proxy case. Two SIPp agents are used: one UAS and one UAC. The B2BUA stands in the path between the two agents. The Figure 6.1 shows the path that is followed by the messages, and the Figure 6.3 shows the flow of messages that are sent between the agents.

When the B2BUA receives a message from an UAC, it automatically forwards it (in a B2BUA way) to the same IP than the UAC, on port 5090. Thus, for this use case, the UAC and the UAS must run on the same host.

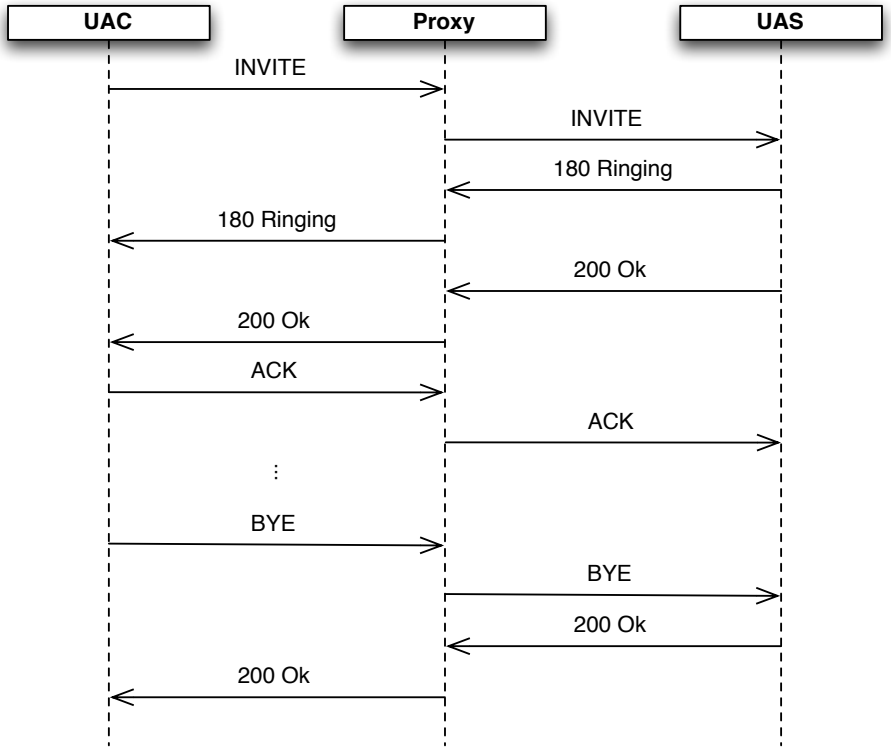

Figure 6.2: The proxy use case.

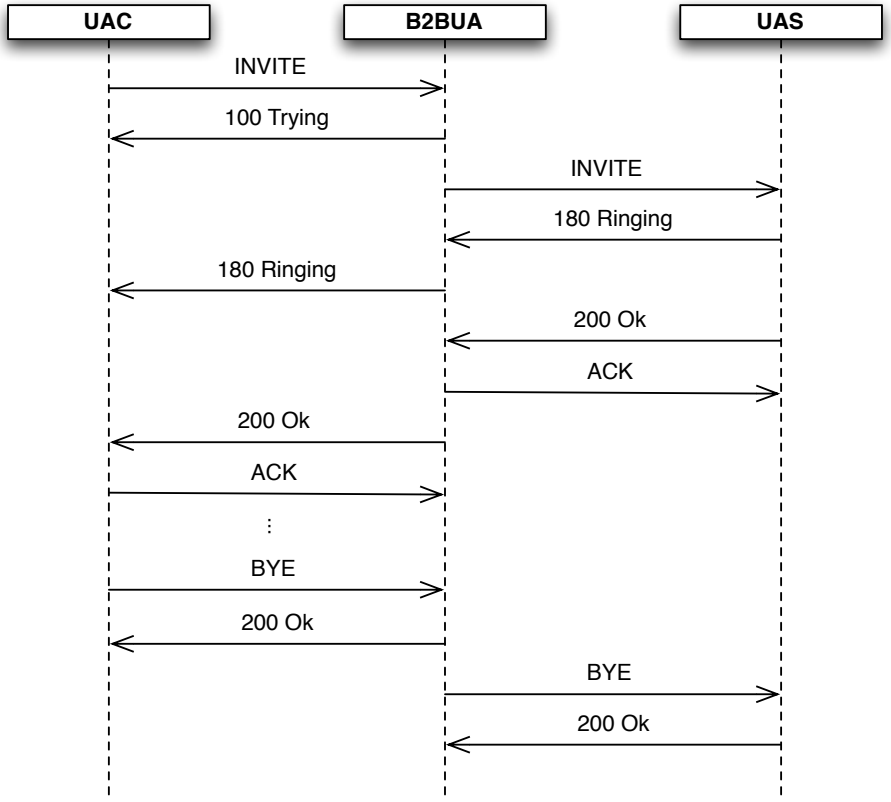

Figure 6.3: The B2BUA use case.

Part III

Design of a solution

Chapter 7

Mobicents Sip Servlets in a cloud

In this chapter, we will first define the requirements for migrating Mobicents Sip Servlets to a cloud platform. We will also define the cloud images that need to be provided.

Then, we will discuss some critics that can be made about our solution.

Finally, we will consider the financial aspect of running Mobicents Sip Servlets on a cloud platform.

Contents

7.1 Requirements

As discussed in Section 5.3, the solution of the problem requires that we migrate Mobicents Sip Servlets to a cloud platform.

The goal for the user is to be able to access in some way to some cloud images and to use them. The user shall be able to start a set of images that will compose a Mobicents Sip Servlets platform. The user shall be able to use this platform as a Mobicents Sip Servlets cluster. Finally, to achieve a full usage of the resources, the user shall be able to adjust the number of instances allocated to the Mobicents Sip Servlet platform depending on the load of this platform.

Thus, the cloud images are the building blocks of the solution. As a consequence, we will define the images that need to be provided in the next Section.

Once Mobicents Sip Servlets will run on a cloud, we will be able to solve the problem. *I.e.* we will have the resources we need in order to scale according to the load. Thanks to the pay per use property of the cloud, no resource will be wasted anymore.

We can consider two scaling mode.

Manual-scaling The user is responsible of the scaling. The scaling is made consecutively to a manual action of the user. This requirement will be fully achieved if Mobicents Sip Servlet runs on a cloud, as the user will be able to start or stop some Mobicents Sip Servlets instances.

Auto-scaling The environment scales automatically depending of the load. This requires the development of a management solution. The management solution will especially need to define the factors whereby the scaling is made.

As a conclusion, we will provide two things:

- A set of images that provides the Mobicents Sip Servlets platform. This is developed in the Chapter 8.
- A management system that provides auto-scaling. This is developed in the Chapter 9.

We would like our solution to be a proof of concept fully functional. Besides, we aim to be cloud provider independent as much as possible.

7.2 Images

In order to put Mobicents Sip Servlets in the clouds, we will define the role of the images that need to be provided. We will provide three images: a Mobicents Sip Servlets image, a Mobicents load balancer image and a management image. The Figure 7.1 illustrates those images.

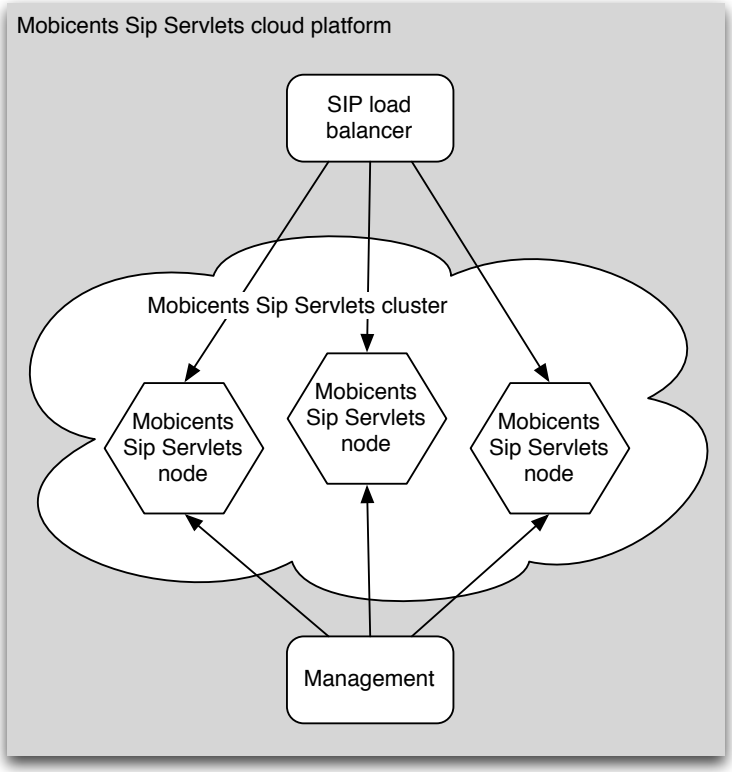

Figure 7.1: Although all the images are instanced in the cloud, only the Mobicents Sip Servlets instances are represented in a cloud. This is the only image which is scaled up and down, according to the load.

7.2.1 The Mobicents Sip Servlets image

The Mobicents Sip Servlets image will contain an instance of Mobicents Sip Servlets running in clustering mode. The JBoss AS clustering mode will allow several instances of this image to work as a cluster. Besides, thanks to the elasticity provided by the JBoss AS clustering mode, new instances will be automatically detected by the cluster and added therein. Existing instances will also be allowed to leave the cluster.

This image will also contain the applications of the user.

Finally, this image will need to be monitored. The monitoring will provide measurements of the auto-scaling factors. Thus, this image will contain an agent, which will communicate with an instance of the management image (see Section 7.2.3).

7.2.2 The Mobicents load balancer image

Since the Mobicents Sip Servlets image will work as a JBoss AS cluster, a SIP load balancer is needed in order to route the traffic among the running instances of the Mobicents Sip Servlets image.

Although it is possible to have multiple instances of the SIP load balancer running a distributable algorithm, we plan to initially have only one instance of the SIP load

balancer.

The Mobicents load balancer image will contain, as its name indicates, the SIP load balancer. The Mobicents Sip Servlets instances will need to know somehow the IP address of the Mobicents load balancer instance in order to register to the SIP load balancer.

7.2.3 The Management image

The management image is responsible for collecting measurements of the monitoring factors from the Mobicents Sip Servlets instances. According to those data, the management instance will be responsible for starting or stopping new instances of the Mobicents Sip Servlets image. The algorithm regulating this auto-scaling is discussed in Section 9.4.

7.3 Critics

The solution we have proposed is not perfect, and we will examine some drawbacks. First, we will criticize the architecture that we have proposed.

Secondly, we will take into account the properties of a cloud compared to the properties of a cluster. A cluster will very likely be composed of a small number of nodes. Although the cluster is elastic, it is rare that a node is added to or removed from the cluster. On the contrary, a cloud can be composed of a very large number of nodes. We can consider cloud of more than 100 nodes for example. Besides, nodes can be added or removed very frequently.

7.3.1 Single points of failure

There exist two single points of failure in our solution. Indeed, we have planned to have one instance of the Mobicents load balancer image and one instance of the management image. Thus, both of those instances are a single point of failure.

However, the SIP load balancer can support stateless algorithms that are distributable (see Section 4.3.3). Thus, it is easy to extend our solution in order to consider multiple instances of the Mobicents load balancer. One could also require that the management system should be responsible for monitoring and scaling the number of instances of the Mobicents load balancer.

We have not defined the management system yet. Similarly to the Mobicents load balancer, one could require that the management system is distributable. In this case, the management system would be responsible for monitoring and scaling its own instances.

In order to stay in a proof of concept spirit, those two solutions will not be developed in this work.

7.3.2 The SIP load balancer in a cloud

Let's consider that the Mobicents load balancer utilizes a distributable algorithm and let's consider that our solution allows to have multiple instances of the Mobicents load balancer. In this situation, the Mobicents load balancer would be able to support a very high load, and this would suit well to a cloud environment.

One question remains: should the new node consistency feature be used or not? The new node consistency feature ensures that when a node is added to or removed from the cluster, the existing sessions will still be routed to the same nodes. If this feature is not used, sessions affinity may be redistributed among the alive nodes. There is a drawback is both cases.

If the new node consistency feature is used, a node may still be overloaded. Indeed, new nodes would be created by the management system in order to lighten the overloaded nodes. Nevertheless, since the overloaded nodes still have to manage the same clients, it would still be overloaded.

If the new node consistency feature is not used, the sticky sessions property of the SIP load balancer could be weakened. Indeed, in a cloud environment, nodes might be added or removed very frequently. If nodes were added or removed more frequently than a SIP request of a dialog was sent, each request would be handled by a different node. Thus, the situation would be the same if the sticky sessions feature was not used.

Thus, the user should choose herself if she enables this feature or not, depending on the characteristics of her applications and her cloud.

7.3.3 The Mobicents Sip Servlets cluster in a cloud

Basically, our solution consists in taking the Mobicents Sip Servlet cluster and putting it in a cloud. In a Mobicents Sip Servlets cluster, sessions are replicated among all nodes of the cluster. This replication leads to a maximal viable number of nodes in the cloud. If too many nodes are added, each node will have to replicate too many data, and eventually the whole cluster will fail.

There are three solutions to resolve this issue. The two first solutions are possible solutions that can be deployed. The third solution is an ideal solution but it is not possible in the current situation.

Avoid distributable application

The simplest solution consists of avoiding the distributable applications. If there is not any distributable application, there is not any data that need to be replicated, and thus the problem does not exist.

Actually, in this solution the problem is just moved elsewhere. If an application is not distributable, it cannot store any data within the sessions. Thus, the data will be stored in the database tier instead. So, the database tier should provide some sort of replication in order to provide a high degree of reliability and scalability. However,

such databases exist and are available [20].

Sub-clusters

One can imagine two levels of scalability. The level one would be, as we have seen, to increase or decrease the number of Mobicents Sip Servlets instances. The level two would be to increase or decrease the number of Mobicents Sip Servlets clusters. Thus we would have several sub-clusters working independently, and an IP load balancer would balance the traffic among the sub-clusters (see Figure 7.2).

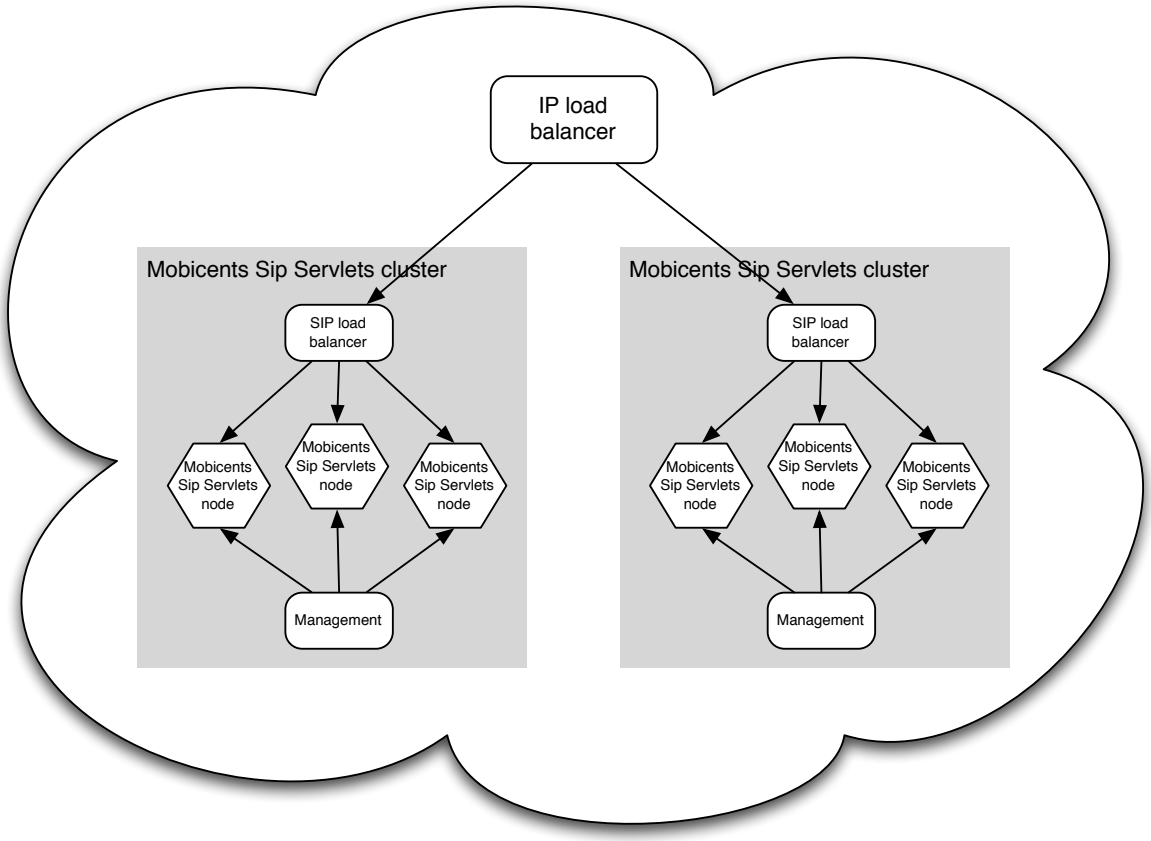

Figure 7.2: A larger cluster can be split in two smaller clusters with an IP load balancer in front of them.

Another problem that arises from this solution is that all the SIP messages concerning the same SIP application session must be handled by the same sub-cluster. However, that would be mostly the case. Let's consider the B2BUA use case (see Section 6.2). When the UAC send the first INVITE request to Mobicents Sip Servlets, one sub-cluster will be chosen randomly among all the sub-clusters to serve this request. One Mobicents Sip Servlets instance within this sub-cluster will issue the INVITE request for the UAS. This request will be sent by the SIP load balancer of the sub-cluster in which the Mobicents Sip Servlets instance sits. So, the UAS will address its response to the SIP load balancer of this sub-cluster, and not to the top-level IP load balancer.

Replicate on some nodes only

Finally, the ideal solution would be to replicate the sessions on some nodes only. Unlike the sub-clusters solution, we would still have one solely cluster. In this cluster, only some nodes would be responsible for replicating some sessions. For determining which node should be responsible for each session, one could think of a solution similar to a DHT ring [33].

The SIP load balancer should also be aware of the responsibility of each node in order to failover a node that is responsible for the sessions of the failed node.

7.4 Budget

We have chosen to develop our platform on Amazon EC2. This choice will be detailed in the Sections 8.1 and 9.2.

Since Amazon EC2 is not free, we have analyzed the cost to develop the solution on this platform. First, we have estimated the budget that will be needed for this work. Secondly, we have estimated how much a provider would have to pay according to the number of calls he would process.

7.4.1 Cost for this work

We have made the following estimation for the budget needed for this work. We plan to work during 10 weeks (or 3 months). During those 10 weeks, we plan to work about 40h per week.

There will be 2 Mobicents Sip Servlets instance, 1 Mobicents load balancer instance, 1 management instance, and 1 meta instance used to build the images. So there will be 5 instances in total. The instances will be launched in the US East region, where the cost for a *small* instance with Linux usage is $0.085 per hour The cost for the instances would be:

$$5 \text{ instances} * 40 \text{ hours} * 10 \text{ weeks} * \frac{\$0.085}{hour} = \$170$$

We plan to launch our tests within the availability zone in order to minimize the data transfer cost. The only data that should be transferred will be SSH traffic between us and the instances. This traffic should be inferior to 1GB for both in and out. The price is $0.1 per GB for in traffic, and $0 for the first GB of out traffic. Thus, the total cost for the traffic would be:

$$1 \text{ GB} * \frac{\$0.1}{\text{GB}} + 1 \text{ GB} * \frac{\$0}{\text{GB}} = \$0.1$$

Finally, we have to consider the Elastic Block Store (EBS) cost. This cost is $0.1 per GB-month and $0.1 per 1 million I/O requests. Each instance is 5GB, except the Mobicents Sip Servlets instance which is 10GB. The number of I/O requests that will

be executed is hard to predict. Let's assume that we will do 100 million I/O requests. The total cost for the storage would be:

$$30\ \text{GB} * 3\ \text{months} * \$0.1 + 100\ \text{million I/O requests} * \frac{\$0.1}{1\ \text{million I/O requests}} = \$19$$

Thus, the total expected cost for using Amazon Web Services would be $189.1. That corresponds to €133.51. A budget of €200 for Amazon Web Services has been allocated for this thesis.

7.4.2 Cost per call

The goal of this section is to evaluate how much it would cost per call to provide a full Mobicents Sip Servlets platform in the cloud. Thus, we will provide a function which takes the expected number of calls per hour as an argument.

First, we define some variables:

instance_cost This is the cost to run an instance during one hour. This cost depends of the instance type. For example, a *small* instance costs $0.085/h and a *high memory double extra large* instance costs $1.00/h.

call_per_instance This is the maximum number of calls an instance is expected to process during one hour. If this value is exceeded, a new instance shall be started.

data_in This is the amount of traffic which is expected to goes in the cloud for one call. The unit is GB/call.

data_out This is the amount of traffic which is expected to goes out the cloud for one call. The unit is GB/call.

Then, we define the constants that represent some Amazon EC2 charges.

storage_cost This is the cost to store 1 GB for one hour.

data_in_cost This is the cost for receiving 1 GB of incoming traffic.

data_out_cost This is the cost for sending 1 GB of outgoing traffic.

The *cost* function return the cost per call and per hour in order to process c calls.

$$
\begin{array}{lll}
cost(c) & = & \frac{1}{c}\Big(\\
 & + & (5+5+10)GB* \text{ storage\_cost} \\
 & + & 2* \text{ instance\_cost} \\
 & + & c* \text{ data\_in } * \text{ data\_in\_cost} \\
 & + & c* \text{ data\_out } * \text{ data\_out\_cost} \\
 & + & ceiling(\frac{c}{\text{call\_per\_instance}})* \text{ instance\_cost } \Big)
\end{array}
$$

Each image is 5GB, except the Mobicents Sip Servlets image which is 10GB.

The load balancer and management images always run.

Incoming traffic.

Outgoing traffic.

Mobicents Sip Serlets instances.

As an example, let's consider the values in Table 7.1. The output of the *cost* function is shown by the Figure 7.3. We can conclude that the cost per call is almost not affected by the number of instances that will be needed in order to process the calls.

Variable	Value
instance_cost	$1
call_per_instance	36 000
data_in	1KB
data_out	1KB

Table 7.1: Example of values

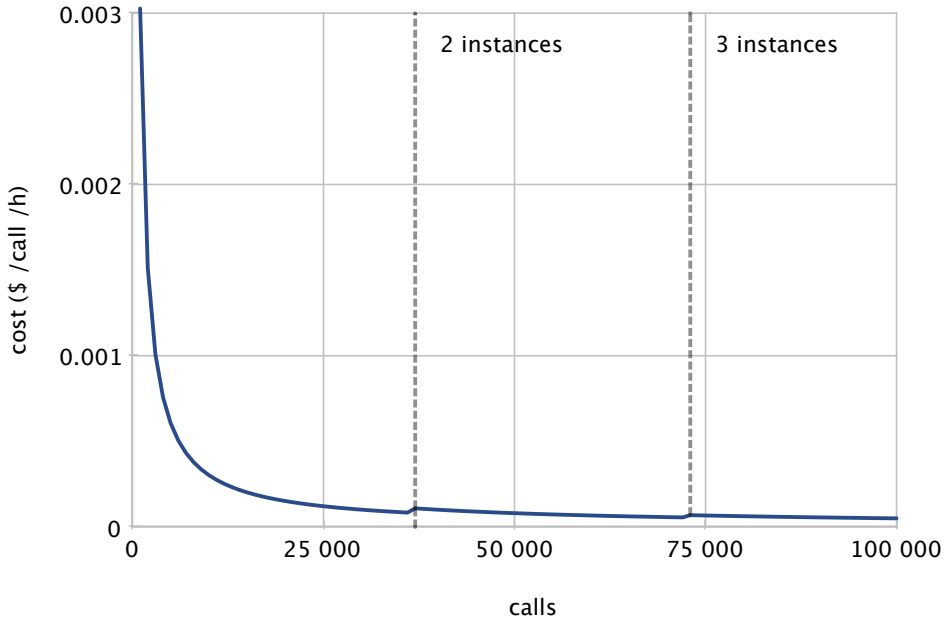

Figure 7.3: Cost for running the Mobicents Sip Servlets platform in a cloud for the value of the Table 7.1. The cost per call is almost not affected by the number of instances that are running.

Chapter 8

Migration to the cloud

The goal of this chapter is to build the images that are needed to run Mobicents Sip Servlets in the cloud. This chapter will focus only on the Mobicents Sip Servlets and Mobicents load balancer images, as described in Sections 7.2.1 and 7.2.2. The build process of the management image is described in the next Chapter.

Contents

8.1 Tools

In order to build the images, two tools are used: Cantiere (see Section 3.1) and BoxGrinder (see Section 3.2). The packages built by Cantiere are described in the first section of this chapter. The next section describe the appliances built by BoxGrinder.

We have developed a project called CirrAS-M. This project contains the resources needed to build the appliances, and a script that build them. The project has been called CirrAS-M because it is based on CirrAS (see Section 3.3). CirrAS-M is an effort to automatically deploy a clustered Mobicents Sip Servlets AS in the cloud. The Appendix A.1 explains the implementation of CirrAS-M.

The appliances that are built with BoxGrinder can only be used on the platforms supported by it (see Section 3.2.3). However, BoxGrinder is built with plugins. So, if another platform support is needed, it would be only necessary to build a plugin for BoxGrinder.

We have chosen to test our solution on Amazon EC2, since Amazon EC2 is supported by BoxGrinder and it is the most famous platform.

8.2 Packages

Two RPM packages are built with Cantiere: the Mobicents Sip Servlets package and the Mobicents load balancer package. All the resources needed to built those packages are included in the CirrAS-M project.

8.2.1 Mobicents Sip Servlets

This package aims to provide Mobicents Sip Servlets as a service. This package contains:

- Mobicents Sip Servlets.
- A configuration file for Mobicents Sip Servlets.
- A script to manage Mobicents Sip Servlets as a service.

During the start operation, the service init script will especially do the following:

- Read the path of the Mobicents Sip Servlets home in the configuration file.
- Read the SIP load balancer's IP from the configuration file, and write that IP in the configuration files of Mobicents Sip Servlets (see Section 8.5.3).
- Start the *all* JBoss AS server of Mobicents Sip Servlets. This launches JBoss AS in the clustering mode.
- Get the private IP of the host and bound JBoss AS to that IP.

8.2.2 Mobicents load balancer

This package aims to provide Mobicents load balancer as a service. This package contains:

- The SIP load balancer provided by Mobicents Sip Servlets.
- A configuration file for the SIP load balancer.
- A script to manage Mobicents load balancer as a service.

During the start operation, the service init script will especially do the following:

- Read the path of the Mobicents load balancer home in the configuration file.
- Get the private IP of the host and bound SIP load balancer to that IP.
- Start the SIP load balancer.

8.3 Appliances

The two appliances corresponding to the Mobicents Sip Servlets and Mobicents load balancer images are built with BoxGrinder. All the resources needed to built those appliances are included in the CirrAS-M project.

The Mobicents Sip Servlets appliance contains the Mobicents Sip Servlets package, and the Mobicents load balancer appliance contains the Mobicents load balancer package. The two appliances are built on top of Fedora 14. The Figure 8.1 illustrates the two appliances.

8.4 Building the appliances

Building the appliances should be done by the developers of Mobicents. In fact, proposing images ready to be deployed on a cloud is another mean of distributing an application. It is comparable to packaging the application, for example in the deb format for the debian-based distributions of GNU/Linux. The appliances can be proposed for downloading, or uploaded on a service such as Amazon Community AMI's (see Section 1.2.1).

8.4.1 Building process

The Section B.1 of the A step-by-step tutorial explains how to build the appliances.

The easiest way to build an appliance with BoxGrinder is by using the BoxGrinder meta appliance. So the first step is to launch an instance of that appliance.

The second step is to check-out the CirrAS-M project from the Mobicents SVN, and to configure BoxGrinder.

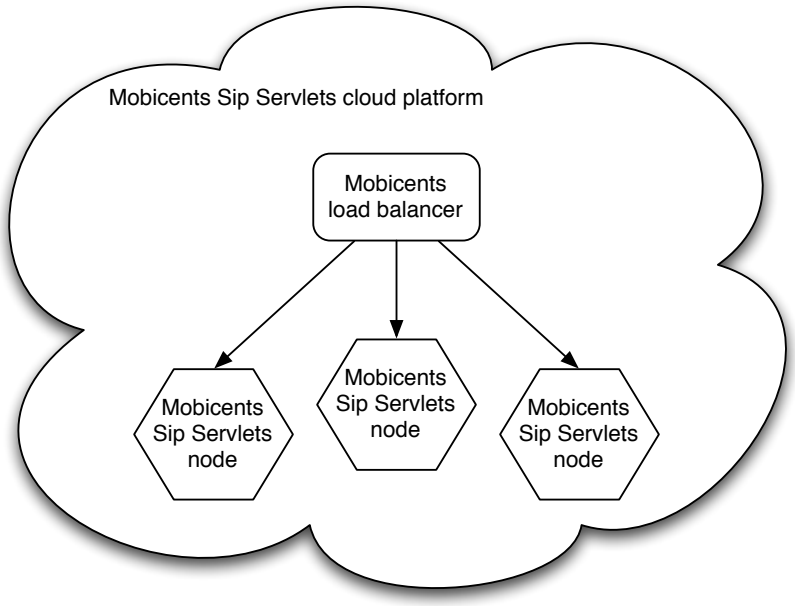

Figure 8.1: We want to provide a Mobicents Sip Servlets cluster in a cloud.

The CirrAS-M project provides a script that build the appliances. The last step is to execute this script. This script does the following:

- Update the system by using `yum`.
- Update the CirrAS-M SVN local copy.
- Clone or update a copy of Cantiere from its git repository.
- Copy the content of the `spec` and `src` folders to Cantiere.
- Build the RPM packages.
- Create a local RPM repository with the previously built packages.
- Build the appliances. The appliances will be delivered to Amazon EBS.

8.4.2 Known bug

During the process, a bug might appear. When an appliance is built by BoxGrinder, BoxGrinder will try to upload this appliance as a new EBS volume to Amazon EC2. In order to do so, BoxGrinder will first request the creation of a new EBS volume. Then, BoxGrinder will attach this new volume to the instance it is running on. It will then copy the appliance to the attached volume, and finally detach the volume and register it as a new AMI.

The bug appears because BoxGrinder does not handle properly the process of attaching an EBS volume. BoxGrinder might think that the volume as been attached

while it has not. Thus, BoxGrinder will try to copy the appliance to the volume and an error will be triggered.

We have reported this bug as BGBUILD-193 [32] and it should be fixed in BoxGrinder 0.9.2 (not yet available during our implementation). A simple workaround exists: the building process just need to be restarted. Note that the instance may also need to be rebooted. Indeed, the attached volume is not properly unmounted from the instance, and the instance may run short of mounting points.

8.5 Using the appliances

In opposition to building the appliances, using them is obviously done by the end-user. The goal is to keep this step as quick as possible, so it will be easier for the user. Basically, the user will have to personalize the appliances she will get from the developers, and then she will be able to use them.

8.5.1 Using the Mobicents load balancer appliance

The load balancer is ready to be used out of the box and does not need any configuration.

8.5.2 Using the Mobicents Sip Servlets appliance

Three quick steps are required in order to configure the Mobicents Sip Servlets appliance:

- Set-up the IP of the Mobicents load balancer.
- Upload the user's applications.
- Edit the DAR configuration.

The Section B.2.2 of the A step-by-step tutorial explains how to process those steps.

When the Mobicents Sip Servlets instance is configured, it is recommended to create an new image from the configured appliance. The creation of this image can be easily processed by the user, as explained in the same Section B.2.2 of the A step-by-step tutorial. From this point, the user should use only the configured image to start new instances. Thus, she would not have to configure again the appliance.

8.5.3 Using an elastic IP

It is possible and recommended to use an elastic IP for the Mobicents load balancer instance. The advantage of doing so is that it will not be needed to reconfigure the Mobicents Sip Servlets appliance if a new instance of the Mobicents load balancer should be used.

In order to use an elastic IP, the SIP load balancer's IP property must be set to the public DNS of the elastic IP. Because the command that get the IP from the DNS server is executed within the cloud, the result will be the internal IP of the host which this IP is attached to. So the internet traffic between the load balancer and the Mobicents Sip Servlets instances will not be charged.

8.6 Conclusion

The goal has been achieved: it is possible to propose images with Mobicents Sip Servlets that are ready to run on a cloud. The user only needs to upload its applications to the appliance, and she will be ready to use Mobicents Sip Servlets in a cloud.

The cloud provider independence of the solution proposed depends of the BoxGrinder plugins. The solution has been tested on Amazon EC2. During the development of this work, BoxGrinder has announced to support more cloud provider, such as ElasticHosts. We can expect BoxGrinder to support even more cloud providers in a near future.

Chapter 9

Management of the cloud

We will first discuss the requirements a management software should support, and choose a management software based on these requirements.

We will then describe the modifications we have made to this management software. We will especially detail the algorithm that is responsible for providing auto-scaling.

Finally, we will explain how this solution has been deployed.

Contents

9.1 Requirements

The Section 7.1 defines the basis of the management system. We would like to have an appliance which will be responsible for the management. An instance of this management appliance will communicate with all the Mobicents Sip Servlets instances. Thus, an agent that will handle this communication may be added to the Mobicents Sip Servlets appliance (see Figure 9.1).

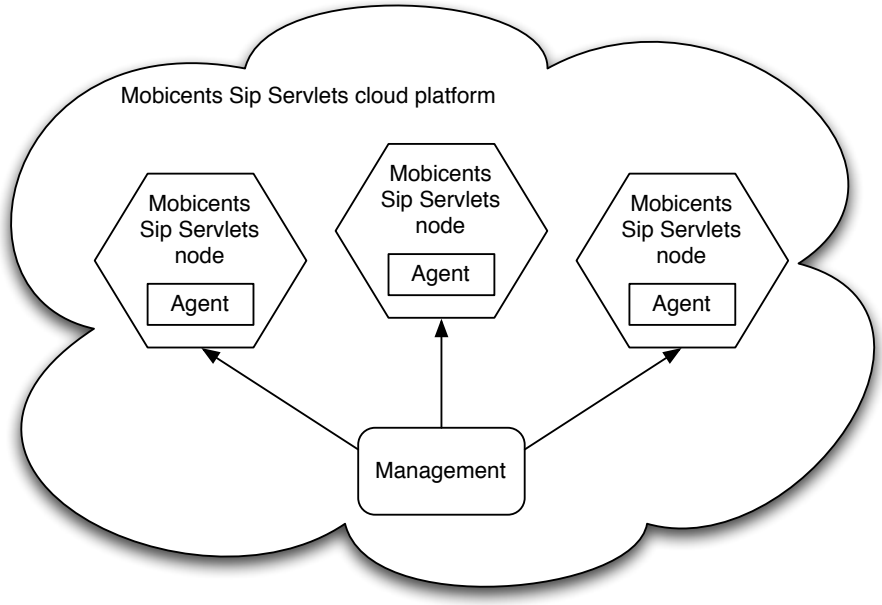

Figure 9.1: The management instance communicates with the Mobicents Sip Servlets instances.

The management instance shall be aware of the state of each Mobicents Sip Servlets instance. (We will define what the state is.) The management instance shall be able to start a new instance of Mobicents Sip Servlets, or stop an existing instance. Besides, the management instance shall provide auto-scaling.

Auto-scaling is the fact that, depending of the states of all Mobicents Sip Servlets instances, some instances may be started or stopped. The goal to achieve is to have all the instances running to be in a state such as they are not idle, *i.e.* they are processing a substantial part of the traffic. Also, none of the instances running should be in state such as they are overloaded, *i.e.* they are not lacking some resources to process the traffic.

Auto-scaling is a process that can be decomposed in three steps (see Figure 9.2):

Monitoring is the action of retrieving the state from an (Mobicents Sip Servlets) instance. This step is done by the agent.

Deciding is the ability to decide if an instance should be started or stopped. This step is done by the management system.

Scaling is the ability to start or stop instances. This step is done by the management system.

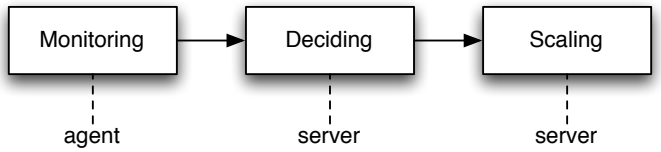

Figure 9.2: Auto-scaling is composed of three steps: monitoring, deciding and scaling.

We must also define the state of an Mobicents Sip Servlets instance. This state is defined by a set of metrics. Those metrics are the following:

- The uptime of the instance.
- The CPU load.
- The memory usage.
- The Java heap memory usage of the Java virtual machine Mobicents Sip Servlets is running on.
- The number of active SIP sessions.
- The number of active SIP application sessions.

As a critic, we notice that the management system is a single point of failure. Indeed, if there are too many Mobicents Sip Servlets instances, the management system may not be able to manage all these instances and will fail.

One could require that the management system is distributable. It would mean that there can be several instances of the management image, and those instances would somehow work together. In order to stay in a proof of concept spirit, we will not require such a thing.

9.2 Management software

In this section, we will evaluate two solutions which the management system could be based on.

9.2.1 Managing with RHQ

The chapter 6 of the user guide of Mobicents Sip Servlets [36] describes a software providing enterprise monitoring and management. This software is actually a RHQ plugin (see Section 2.3) developed for Mobicents Sip Servlets. RHQ is a system management for a JBoss AS cluster.

The Mobicents Sip Servlets RHQ plugin has some capabilities needed for our management system. First, RHQ use the same architecture that we need: an agent runs on each node of the JBoss AS cluster. Each agent reports to a server, which is the RHQ server in this case. Secondly, all the informations that we need are monitored by the plugin. So we are able to access them easily and the monitoring phase would be already implemented.

The first idea has been to develop a plugin for RHQ in order to run it in the cloud. Since the monitoring is already provided, only the deciding process and the scaling process would have to be implemented. However, this idea has had to be abandoned because RHQ does not support JBoss AS 5.1, which Mobicents Sip Servlets is based on [53].

9.2.2 Managing with SteamCannon

Another possibility is to use SteamCannon (see Section 3.4) for managing our Mobicents Sip Servlets cloud. SteamCannon is a system management for an Amazon EC2 cloud.

SteamCannon also has some capabilities needed for our management system. First, SteamCannon use the same architecture than RHQ: a SteamCannon agent runs on each instance that is monitored. Each agent reports to a SteamCannon server. Secondly, SteamCannon can monitor JBoss AS instances. However, we are also interested in some informations that are specific to Mobicents Sip Servlets. So we would have to modify the monitoring in order to add more informations. SteamCannon also has the capability to start and stop instances within the cloud that is supervised. Thus, the scaling phase is already implemented.

For this benefits, we have chosen to use SteamCannon as management software.

9.3 Implementation

This section explains, based on the architecture of SteamCannon, the components that our solution should provides. This is illustrated by the Figure 9.3.

On the node side, two components need to be modified.

The Mobicents Sip Servlets container shall be modified so it could provide the monitoring informations discussed in Section 9.1. This will be done by implementing a service MBean called Monitor. This MBean will provide a `monitor` method, which will return a String containing all the monitoring informations. A MBean is a special kind of EBJ which is used for management (see Section 2.1.3). The Appendix A.2 explains the implementation of the Monitor.

The SteamCannon agent shall be modified in order to support Mobicents Sip Servlets as a service (as defined in the Section 3.4). This service will support the following operation: start, stop, status and show the logs.

On the server side, the SteamCannon server shall be modified so it could support

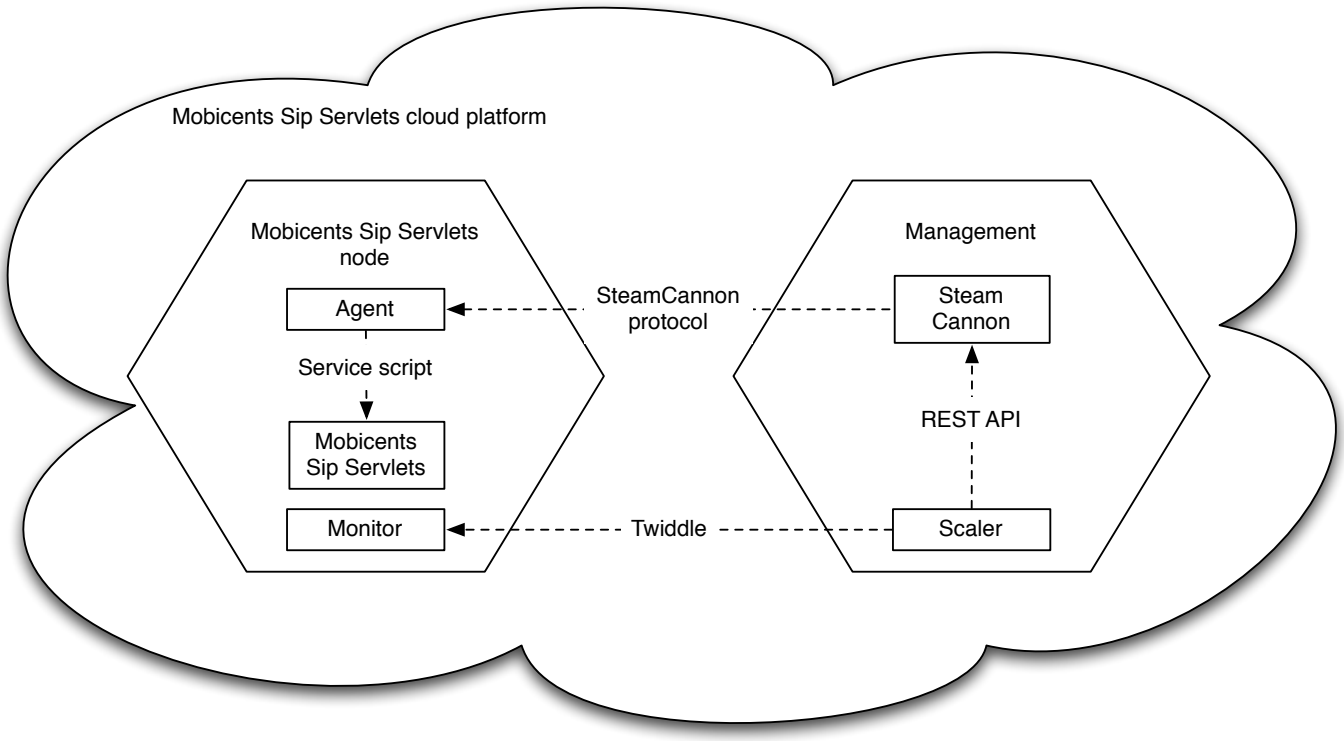

Figure 9.3: The SteamCannon agent and the SteamCannon server shall the Mobicents Sip Servlets. The cloud monitor shall provide the the monitoring step, and the scaler shall provide the deciding step.

the Mobicents Sip Servlets service. Besides, a Mobicents Sip Servlets platform shall be added to the server, so it will be possible to start and manage such an environment. The Appendix A.4 shows the modifications made to the SteamCannon agent and the SteamCannon server.

Finally, a client for the SteamCannon server shall be implemented. The client will be responsible of the deciding and scaling steps. We will call this client the scaler. The scaler will use the features provided by the SteamCannon server via the server's REST API. It will use especially the server in order to know the instances that are running, and be able to start or stop some instances. When the scaler knows an instance, it will trigger its `monitor` method directly by using Twiddle [52].

Twiddle is a command execution tool that allows the user to send a command to a JBoss AS instance. The command is handled by the JMX server of JBoss. JMX is a technology that is used to implement management interfaces for Java applications [25]. In short, the Monitor MBean is registered to the JMX server. When Twiddle sends a command to the JMX server, the JMX server retrieves the MBean and forwards the command to it.

The Appendix A.3 explains the implementation of the scaler.

9.4 Auto-scaling algorithm

In order to provide auto-scaling, the scaler must use an algorithm which will indicate when start or stop an instance. The Listing 9.1 gives the pseudo-code of the algorithm while this section explains how the algorithm works.

First, we will define two thresholds: the minimum threshold and the maximum threshold. Each threshold will contain a value for each metric that is monitored, except the uptime metric.

Periodically, the scaler will get the list of the instances of each environment it has to monitor from the SteamCannon server. It will then trigger the `monitor` method and get some values for each metric monitored for each instance. The parameter `MONITOR_INTERVAL` allows the user to set the duration of the periods.

Three sets are created: `belows`, `averages` and `aboves`. The instances for which *all* the metrics are below the minim threshold are put in the `belows` set. The instances for which *at least one* of the metrics is above the maximum threshold are put in the `aboves` set. The others (for which the metrics are between the minimum and maximum thresholds) are put in the `averages` set.

If there is not any instance in both the `belows` and the `aboves` sets, the system is stable and no action is taken. If there are as many instances in the `belows` as in the `aboves` set, the system is not well distributed, and no action is taken neither. If there are more instances in the `belows` set than in the `aboves` set, one instance is stopped. On the other hand, if there are more instances in the `aboves` set than in the `belows` set, one new instance is started.

A problem may appear during the execution of this algorithm. This problem is stated as following: if one instance is started because there was more above instances than below, the new instance may handle a part of the traffic such that there is now more below instances than above. In consequence, an instance will be stopped. This may cause the situation where there are more above instance than below again. We would then be in a loop, where an instance is started then stopped several times in a row.

There is two ways to avoid this problem. The first way is to configure the thresholds properly so this situation never happen. It is obviously impossible to prove that the thresholds are configured properly.

The second way is to set the rate of growing. When an instance is started or stopped, the algorithm shall not start or stop another instance before the amount of time indicated by the rate is elapsed. This will ensure at least a minimum lifetime for an instance. The parameter `PERIODS_OF_INACTION` allows the user to indicate the number of periods of inaction after a successful action.

Listing 9.1 Deciding step of the auto-scaling algorithm.

```
const minThreshold
const maxThreshold
last_action_clock ← 0

loop
  wait MONITOR_INTERVAL seconds
  last_action_clock ← last_action_clock + 1
  instances ← get running instances from SteamCannon
  belows ← ∅
  averages ← ∅
  aboves ← ∅

  for all instance in instances do
    monitoring ← monitor(instance)
    if monitoring < minThreshold then
      belows ← belows ∪ {instance}
    else if monitoring > maxThreshold then
      aboves ← aboves ∪ {instance}
    else
      averages ← averages ∪ {instance}
    end if
  end for

  if |belows| = |aboves| then
    do nothing
  else if last_action_clock < PERIODS_OF_INACTION then
    do nothing
  else if |belows| > |aboves| then
    stop an instance
    last_action_clock ← 0
  else if |belows| < |aboves| then
    start an instance
    last_action_clock ← 0
  end if
end loop
```

9.5 Deployment

This section will explain which modifications need to be applied to the solution described in Chapter 8 in order to include our management system.

9.5.1 Packages

The Mobicents Sip Servlets package from the CirrAS-M project is modified in order to include the Monitor MBean.

The SteamCannon agent is packaged into steamcannon-agent. Note that we package the implementation of the SteamCannon agent that contains our improvements. The SteamCannon project [49] already provides the spec files needed by Cantiere to build this package [50]. However, the SteamCannon project build the steamcannon-agent package dependencies dynamically, by using Rumpler [5].

Rumpler is a tool that builds the dependencies spec files for a specified version of another spec file.

In order to be able to build the steamcannon-agent package, we have imported all the spec files for the dependencies of steamcannon-agent built by Rumpler. All those packages have been included in the CirrAS-M project.

9.5.2 Appliances

The Mobicents Sip Servlets appliance shall be modified in order to include the steamcannon-agent package. The appliance definition also indicates which service the agent will be responsible of. In this case, the agent will be responsible of a service called `MobicentsSipServlets`.

We have tried to build an appliance for SteamCannon server as well. However, the appliance definition provided by SteamCannon [50] is obsolete. SteamCannon depends of Torquebox, and Torquebox has been updated such as it is not possible anymore to build the SteamCannon appliance.

The solution is to patch the 0.2.0 release of the SteamCannon appliance. This appliance is available in the Amazon community AMI's.

9.5.3 Building the appliances

Since our improvements have been made available in the CirrAS-project, the Mobicents Sip Servlets and Mobicents load balancer appliances provided by this project are ready to be used.

The 0.2.0 SteamCannon appliance need to patched. This is covered by the Section B.1.2 of the A step-by-step tutorial. As in the Section 8.4, this process should be executed by the developers of Mobicents Sip Servlets.

9.5.4 Using the appliances

First, the user will need to define a platform that the management system will use.

The management system provides two functionalities. Fist, the user can use SteamCannon to manually controls her Mobicents Sip Servlets environment. Secondly, she can install the scaler component in order to achieve auto-scaling.

As in the Section 8.5, those steps should be done by the user. The three steps are explained in the Section B.2.3 of the A step-by-step tutorial

Add a platform

The user needs to define a platform. The Mobicents Sip Servlets platform should contain only the Mobicents Sip Servlets appliance, since it is the only appliance that scales up or down. It is recommended that the user uses the appliance she has customized as explained in Section 8.5.2.

Control the environment

The benefits of using SteamCannon is that the user can also use the features it provides. SteamCannon delivers those features though a web interface.

Thus, the user can create a Mobicents Sip Servlets environment from the platform definition. She can choose how many Mobicents Sip Servlets instances the environment will initially contain. She can then start or stop instances within this environment.

Each Mobicents Sip Servlets instance provides the Mobicents Sip Servlets service. Thanks to this service, the user can start, stop and get the status of the Mobicents Sip Servlets container that is running on an instance. She can also see the logs of the container.

Auto-scaling

The user can get auto-scaling by installing the scaler. The scaler can be installed on any host, even outside the cloud, since it communicates with SteamCannon via the latter's REST API. However, we recommend to instal it on the management appliance, in order to keep all the management softwares in the same place.

The scaler depends on Twiddle for communicating with the instances. Since SteamCannon runs on top of Torquebox, which runs on top of JBoss AS 6, the SteamCannon appliance already owns a Twiddle instance. Unfortunately, the Twiddle instance of JBoss AS 6 is not compatible with JBoss AS 5.1. Thus, it is necessary to install Twiddle for JBoss AS 5.1.

The scaler can be installed by downloading it from the Mobicents code repository. The user can customize some parameters of the scaler, such as the thresholds and the rate of growing (see Section 9.4).

Once it is started, the scaler will automatically apply auto-scaling to all the environments started within SteamCannon.

9.6 Security analysis

In this section, we will analyze the security of the overall solution. The Figure 9.4 shows the connections that we will consider.

SteamCannon communicates with the SteamCannon agents through a SSL connection. Thus, this connection is secure.

The scaler communicates with SteamCannon through its REST API and with the Mobicents Sip Servlets instances through Twiddle. Those two connections are not encrypted, thus an attacker could eavesdrop those connections. Besides, the REST API of SteamCannon requires the credentials of the user in order to respond to the requests. So the attacker could retrieve those credentials.

However, the scaler is designed to run within the cloud. The cloud environment can be considered as a secure LAN and it should not be possible for an attacker to enter it.

Finally, SteamCannon provides a web interface which allows the user to control her environments. This web interface does not provide a SSL connection so the data are transferred in cleartext. Besides, this web interface requires the user to enter her Amazon EC2 credentials such as her access key and her secret access key. So an attacker could retrieve both the Amazon EC2 credentials and the SteamCannon credentials of the user. Thus, this connection is not secure and should be used on secured environments only.

9.7 Conclusion

SteamCannon has been modified in order to support the Mobicents Sip Servlets platform. It is now possible to manage this platform with SteamCannon. It is possible to start or stop some instances.

Besides, a component called the monitor has been implemented. The role of the monitor is to measure some metrics on the Mobicents Sip Servlets instances.

Finally, a component called the scaler has been implemented. The scaler provides the auto-scaling algorithm. Thanks to the scaler, it is possible to automatically scale the Mobicents Sip Servlets instances according to the values returned by the monitor.

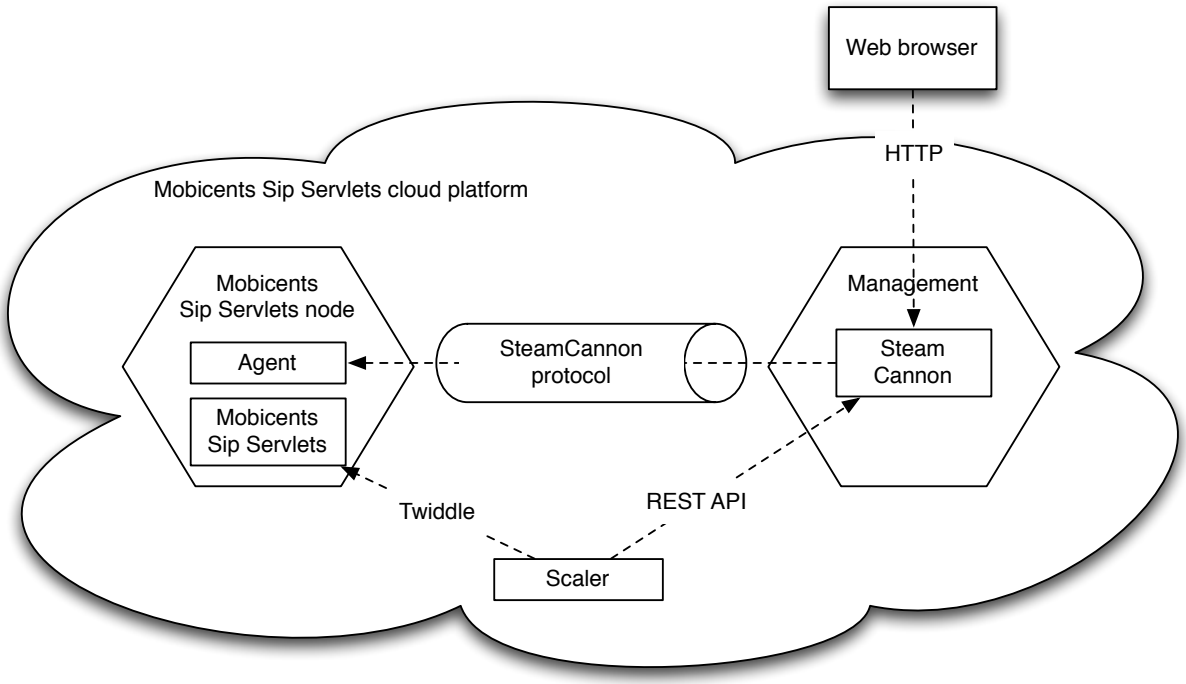

Figure 9.4: SteamCannon uses a SSL connection to communicate with the agents. The other connections are not encrypted.

Chapter 10

Evaluation of the solution

In this chapter, we will evaluate the solution. In order to do so, we will deploy the two use cases on a cloud, and we will observe the management system scaling up and down the number of instances.

Contents

10.1 Procedure

The goal is to test the management system among the two use cases. So we will run two tests: one for each use case.

We will configure a Mobicents Sip Servlets platform providing the use case to test. This platform will initially contain one Mobicents Sip Servlets instance, one Mobicents load balancer instance and one management instance. Another instance, called the client instance, will be used in order to act as the clients and servers of the use cases. This instance will contains two SIPp agents: one UAC and one UAS. (See Figure 10.1).

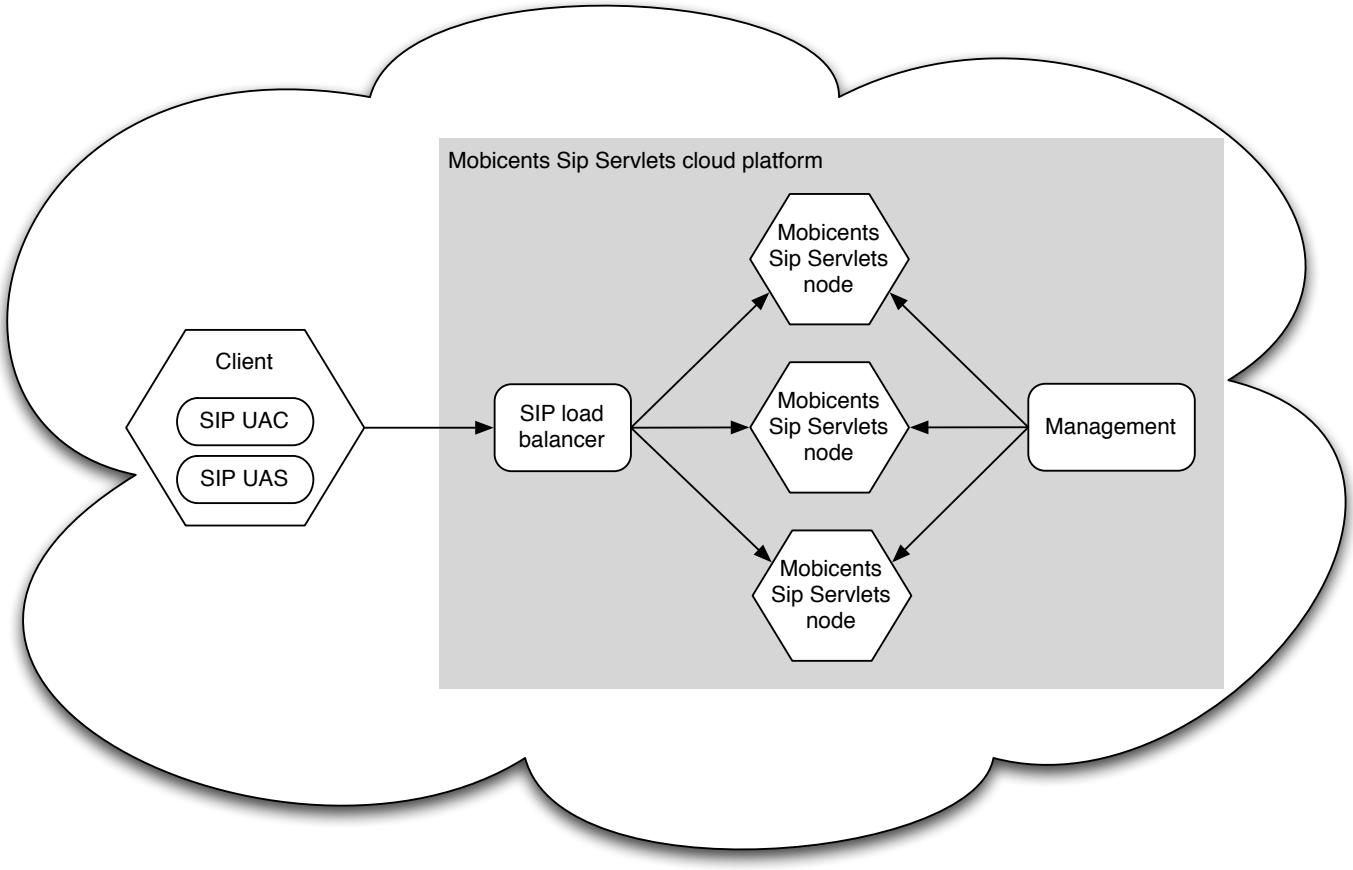

Figure 10.1: The client instance contains two SIPp agents that are used to generate SIP traffic.

The SIP load balancer will be configured with the header consistent hash algorithm. This algorithm has been chosen because it is stateless and it redistributes the clients among the nodes of the cluster when a node is added or removed.

We will run the tests on Amazon EC2. Each instance will run on a *small* instance (1.7GB memory, 1 EC2 compute unit), except the client instance which will run on a *medium* instance (1.7GB memory, 5 EC2 compute units). The client instance needs a larger computing power because it will generate a high call rate.

The role of the client instance is to generate traffic in order to make the management

system launch some instances. There will be two phases. In the first phase, the client instance will increase the traffic. The expected result is that the management system will start some instances of the Mobicents Sip Servlets image. In the second phase, the client instance will decrease the traffic. The expected result is that the management system will stop some instances (see Figure 10.2).

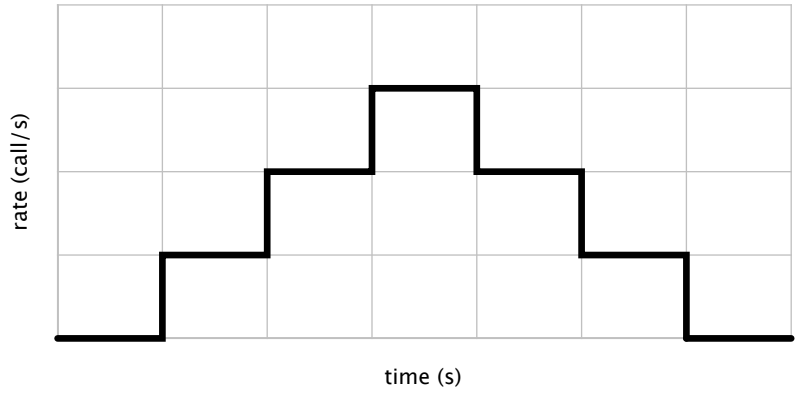

Figure 10.2: The call rate will first increase then decrease.

10.2 Experiment

The client instance has been configured to deliver first a rate of 5, 10 and 15 calls per second during the 21 first minutes (7 minutes for each rate). Then, it will deliver a rate of 10 and 5 calls per second and stop. The management system has been configured to measure the metrics every 15 seconds. The period of inaction has been set to 20, which corresponds roughly to 5 minutes. The thresholds have been configured as shown in Table 10.1. Also, the maximal number of instances has been set to 3.

	Minimum threshold	Maximum threshold
CPU load	0.5	1.5
Memory usage	1	1
JVM memory usage	1	1
SIP sessions	50	200
SIP applications sessions	25	100

Table 10.1: An instance will be stopped if all of its metrics are below the minimum threshold. An instance will be started if one metric of one instance is above the maximum threshold.

The results are shown in the Figure 10.3 for the proxy use case and in the Figure 10.4 for the B2BUA use case.

The first graph shows the rate of the calls that have been received by the UAS. The expected rate is the rate that the UAC has been requested to send, and the real rate is the rate actually received by the UAS.

The second graph shows the number of running Mobicents Sip Servlets instances. A Mobicents Sip Servlets instance is considered as running when Mobicents Sip Servlets is actually ready to process the traffic. When the management system decides to start an instance, our experiments have shown that it takes about 3 minutes to the instance to be considered as running. So, when the graph shows an increase of the number of instances, the management system has actually taken that decision some time before. On the other hand, when the management system decides to stop an instance, this is shown immediately on the graph.

The others graphs show the metrics that have been measured on each instance.

10.3 Analyze

For both tests, the memory and JVM memory metrics have not been used in the thresholds. We have noticed that those metrics increase but never decrease. The problem to use those metrics for the maximum threshold is that if the threshold is reached once, it will always be and the scaler will always create new instances. The metrics are also useless for the minimum threshold since they never decrease. The small amount of available memory on *small* instances and the way the memory is handled by the operating system could explain why the memory is rapidly saturated and never decrease.

10.3.1 Proxy use case

For the proxy use case test, we notice that the real rate is very close to the expected rate. At around 2'45", all the metrics have been monitored as 0. A bug might have produced this result.

We also notice that the number of SIP sessions is always the same that the number of SIP application sessions. This was excepted, as each SIP application session should contain one and only one SIP session.

Since the maximum SIP application session threshold is lower than the maximum SIP session threshold, only the maximum SIP application session threshold has been hit. The SIP application session metric is the only metric responsible for the increase of the number of instances.

Finally, the number of instances is proportional to the call rate, except that there is a small time shift. This time shift is explained by the fact that it takes some time for an instance to start.

10.3.2 B2BUA use case

For the B2BUA use case test, we can observe that a congestion occurs when the rate is expected to grow up to 15 calls per second. Different factors show this congestion. First, the rate actually achieved is lower than the expected one. Then, the instances are

running at a high CPU load. Finally, the number of SIP sessions and SIP application sessions suddenly increase. It has not been identified if this congestion is due to the new instance that is started, to the rate increase or to another factor. Nevertheless, this test allows us to consider the behavior of the management system according to the metrics measured.

SIPp allows us to automatically increase or decrease the traffic at some point. However, SIPp does not allow us to decrease the traffic after having increased it. (The rate increase must be constant during all the execution.) Thus, the UAC is actually modeled by two SIPp scripts that are executed consecutively. Because the first script is stopped, many calls that were stuck in some state are dropped. That would explain the small increase of the rate and the small decrease of the number of SIP sessions and SIP applications sessions around the 21st minute.

We also notice that the number of SIP sessions is always twice the number of SIP application sessions. This was expected, as each SIP application sessions should contain two SIP sessions: one for the UAC and one for the UAS.

The maximum CPU load threshold, the maximum SIP sessions threshold and the maximum SIP application sessions threshold have been hit and are responsible for the increase of the number of instances.

As in the proxy use case test, the number of instances is proportional to the call rate, except that there is a small time shift.

10.4 Conclusion

The management system has been able to respond correctly according to the metrics. The management system is likewise good at scaling up than at scaling down.

The main flaw of this system is that it is slow. This slowness is caused by the inaction period of the auto-scaling algorithm. However, the inaction period should not be decreased, as it takes a few minutes to an instance to start. Our experiences have shown that JBoss AS takes about 2.5 minutes to start and thus can be considered as the bootle neck. Running on more powerful machines or using a more recent version of JBoss AS could increase this startup time.

The two use cases have been exposed to the same call rate and have responded differently. The values of the metrics are really dependent of the application. Thus, it is necessary to analyze the behavior of an application in order to determine the thresholds that will suit well for this application.

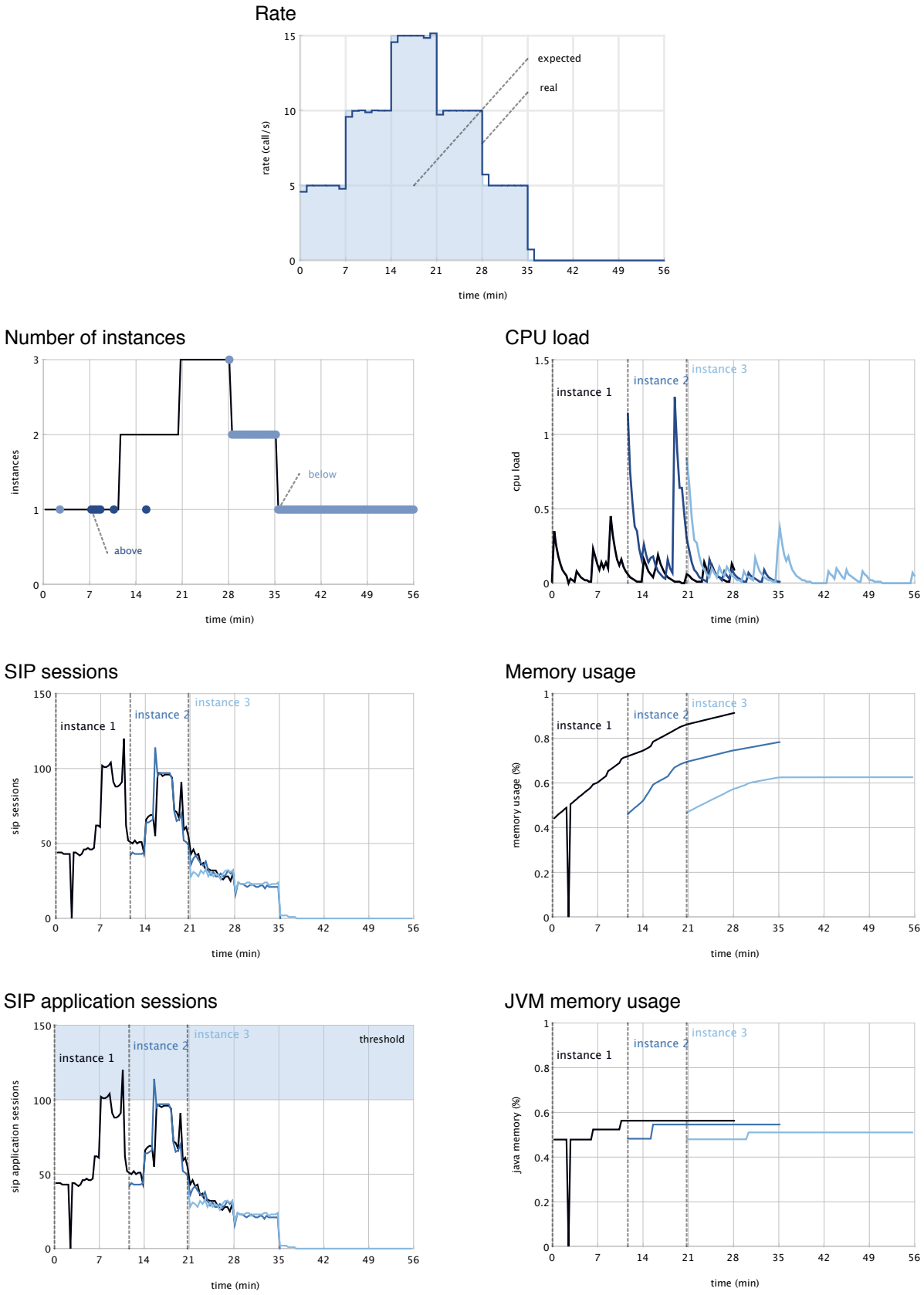

Figure 10.3: Results for the proxy use case test.

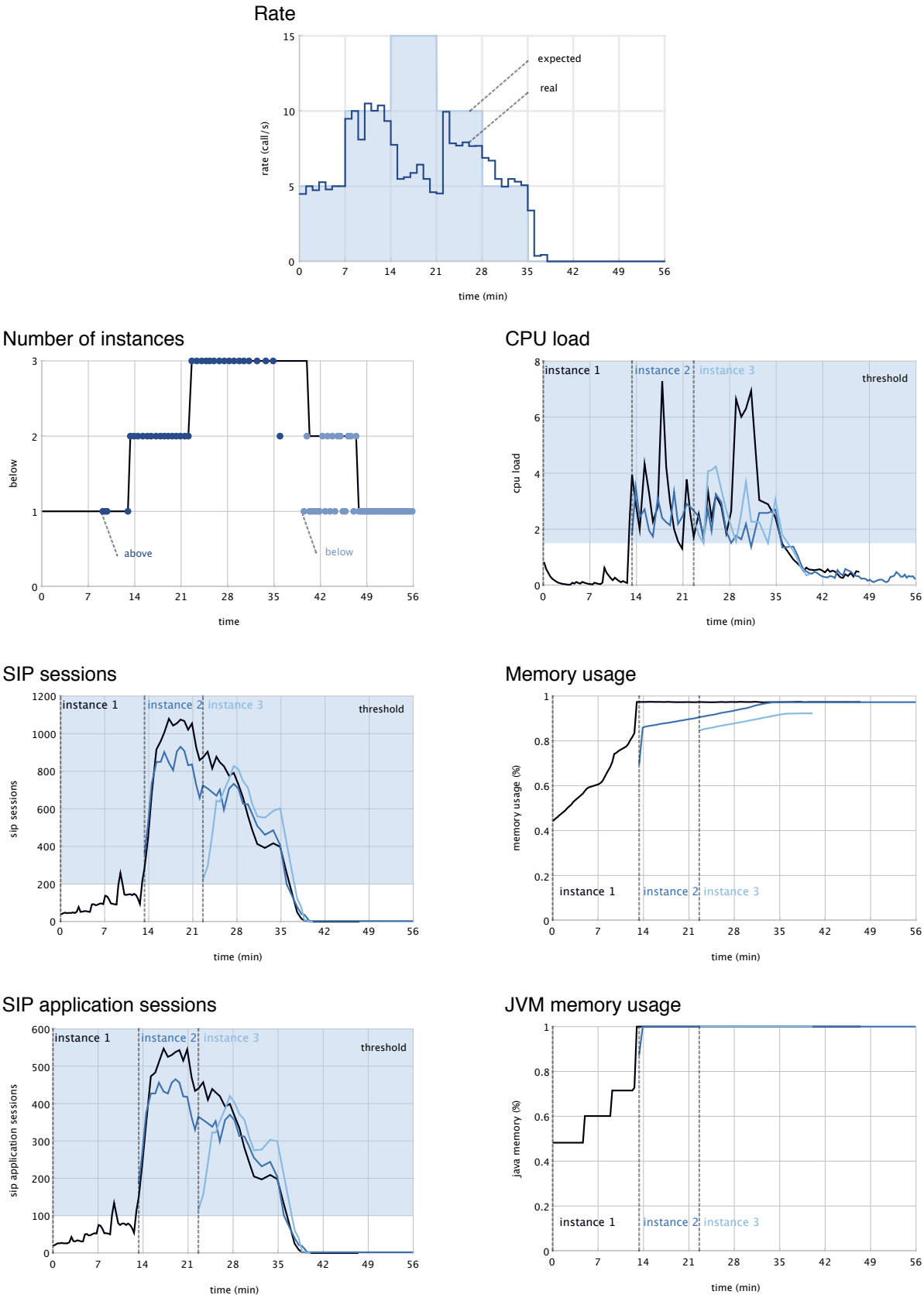

Figure 10.4: Results for the B2BUA use case test.

Part IV

Outcome

Chapter 11

Conclusion

We have provided cloud images for the Mobicents Sip Servlets platform. Those images uses the JBoss AS clustering capabilities in the cloud. Thus, it is possible to deploy in a cloud the same Mobicents Sip Servlets platform that would have been deployed in a cluster. We have seen that this design is not optimal since it leads to a high degree of replication. However, it can be used as it for small clouds.

We have also implemented a management system that supports auto-scaling. This management system is fully functional and can be used to manage clouds. However, the tests have shown that the parameters must be carefully set. Those parameters depend on the specific resources that are needed for each application. Thus it is the role of the user to adjust those parameters according to her applications.

From this work, we can deduce some requirements that an application server should fulfill in order to be deployed on a cloud. First, the application server should support a clustering capability that would allow it to run on several instances. The clustering capability must provide the elasticity, high availability and failover properties. Nevertheless, it is important to design this clustering in a way that would not involve a session replication on each node. Secondly, the application server should provide a load balancer aware of the responsibilities of each node in order to route the request to the corresponding node. In order to be scalable, the load balancer must use a stateless and distributable algorithm.

We have committed the code related to this work to Mobicents. Mobicents is an open-source project and we have taken part in the development of Mobicents throughout the elaboration of this work. We have participated to the community weekly meetings and we have received feedbacks from the community. The solution we have developed is already fully integrated to the Mobicents project, and will continue to be developed by the community. Indeed, some members of the community have already tried our solution, while other have tried to build the cloud images for Eucalyptus. Besides, the Appendix B has been already integrated to the documentation of Mobicents [18]. We will continue to provide support for our work during the weekly community meetings and we will keep a strong link with the community.

Chapter 12

Further work

In this section, we will describe some improvements that could be added to our solution.

Contents

12.1 Mobicents RPM packages

In the process of migrating Mobicents Sip Servlets on a cloud, we have built RPM packages for Mobicents Sip Servlet and the SIP load balancer. Those packages are installed on the appliances.

One could want to use those packages in another context than the appliances. Indeed, those packages could also be used as another mean of distribution for Mobicents Sip Servlets. They could be integrated in famous distributions repositories such as the repositories of Fedora or RHEL.

However, that would require a slight adaptation of the packages. The packages definitions respect the spec file standard and they can be installed on any distribution that supports RPM packages. Nevertheless, the packages are designed to be installed once during the appliance building process, and not to be updated later. Thus, those packages cannot be updated via the yum utility.

So, it is necessary to complete the spec definitions of the packages in order to support the update operation. If this is done, the packages can be put in any RPM repository for any use.

12.2 Mobicents appliances

This work has shown how to build appliances for a Mobicents Sip Servlets platform. It could be interesting to distribute those appliances in a place such as the Amazon Community AMI's so the users can just download them and use them.

12.3 Better SteamCannon integration

SteamCannon has been used for our management system. However, a better integration of the features provided by SteamCannon is possible.

By default, SteamCannon supports a JBoss AS platform. SteamCannon allows the user to edit, through its interface, some parameters of JBoss AS such as the credentials of the admin user and the multicast IP. For now, it is not possible to edit the same parameters for the Mobicents Sip Servlets platform. Since it is already available for JBoss AS, it should be easy to integrate them for Mobicents Sip Servlets as well. Besides, some parameters specific to Mobicents Sip Servlets should be editable through SteamCannon as well. Those parameters are the IP of the SIP load balancer and the DAR configuration.

SteamCannon also supports the artifacts. In the case of a JBoss AS platform, an artifact is an user's application. The Mobicents Sip Servlets platform for SteamCannon does not support the artifacts yet. SteamCannon should support the artifacts for Mobicents Sip Servlets, so it would be possible for the user to upload her applications through SteamCannon.

If both the parameters edition and the artifacts were supported by SteamCannon for the Mobicents Sip Servlets platform, the user would not have to customize its instance of Mobicents as explained in Section 8.5. Indeed, all those operations would be possible through SteamCannon.

Moreover, for now the scaler uses Twiddle to retrieve the monitoring informations. However, thanks to our improvement, the SteamCannon agent support this operation as well. The SteamCannon server could be modified in order to allow clients to retrieve the monitoring informations through the REST API of SteamCannon. So, the scaler would not be required to have an instance of Twiddle anymore.

12.4 High availability

In our solution, we only have one instance of the Mobicents load balancer image and one instance of the management image. In order to provide high availability, it is necessary to avoid those single points of failure.

As we have stated before (see Section 7.3.1), it would be easy to add more instances of the Mobicents load balancer, since it supports distributable algorithms. SteamCannon does not support multiple instances, so more work would be required in order to distribute this system. Nevertheless, it would be interesting to distribute the management system as well.

Moreover, the management system should also manage the Mobicents load balancer instances and the managements instances in order to provide auto-scaling for those two.

12.5 JBoss for the clouds

As we have stated in the Section 7.3.3, JBoss AS clustering is not well adapted for a cloud environment because the sessions are replicated on all nodes of the cloud. An improvement would be to replicate the sessions on some nodes only.

12.6 Auto-scaling algorithm

The auto-scaling algorithm (Section 9.4) can be improved in several ways:

- For now, the memory and the JVM memory metrics are useless (see Section 10.3). We can improve those metrics in two different ways. First, the scaler can take those metrics into account only once, so it will not start more than one new instance for a given instance if those metrics are above the threshold. Secondly, we can improve the way those metrics are implemented in order to reflect the real memory usage of Mobicents Sip Servlets.
- For now, when an instance must be stopped, the instance which is stopped is chosen by chance. We can optimize this choice. For example, we could require to

stop the instance that is the less used.

- The avoid loop solution could be improved in order to allow multiple instances to start or stop more rapidly. For now, there is an inaction period between each action. This inaction period imposes a maximal rate of instance that can be started or stopped. It should be possible to start several instances more rapidly, in the case where the environment is facing a traffic that only one instance more could not help to absorb.

- The metrics that are measured includes the uptime metric. However, this metric is not used. This metric could be used in order to avoid the loop problem.

- For now, the bandwidth is not monitored. However, the bandwidth is an interesting metric. Indeed, a powerful instance may be limited by its bandwidth. Thus it would be useful to monitor the bandwidth and to add this metric as a criteria in the thresholds.

Part V

Appendix

Appendix A

Code

This appendix aims to detail the code that has been produced during the elaboration of this thesis. All the code has been committed on the Mobicents code repository [4], under the folder trunk/cloud/sip-servlets/. The code is also presents on the CD provided with this document, in the code/ folder.

This folder contains 5 sub-projects: examples, CirrAS-M, monitor, scaler and SteamCannon. This appendix will detail each of those sub-projects.

Contents

A.1 CirrAS-M

The CirrAS-M project is used in order to build the images defined in Section 7.2. The Section B.1.1 explains how to build the appliances.

The specs/ folder contains the definitions of the RPM packages that are built, and the src/ folder contains the files that are used in the package process. The appliances/ folder contains the definitions of the appliances. Finally, the `build-cirrasM.sh` script is used to automatically build the packages and the appliances.

The Listing A.1 shows the tree of the CirrAS-M project.

Listing A.1 CirrAS-M's tree

```
.
|-- README
|-- appliances
|   |-- mobicents-base.appl
|   |-- mobicents-load-balancer.appl
|   |-- mobicents-sip-servlets.appl
|-- build-cirrasM.sh
|-- config
|-- specs
|   |-- mobicents-load-balancer.spec
|   |-- mobicents-sip-servlets.spec
|   |-- steamcannon-agent-dependencies-1fcee5c.spec
|   |-- steamcannon-agent.spec
|   |-- steamcannon-rubygem-addressable-22.spec
|   |-- steamcannon-rubygem-bundler-10.spec
|   |-- steamcannon-rubygem-daemons-11.spec
|   |-- steamcannon-rubygem-data_objects-010.spec
|   |-- steamcannon-rubygem-dm-core-10.spec
|   |-- steamcannon-rubygem-dm-do-adapter-10.spec
|   |-- steamcannon-rubygem-dm-is-tree-10.spec
|   |-- steamcannon-rubygem-dm-migrations-10.spec
|   |-- steamcannon-rubygem-dm-sqlite-adapter-10.spec
|   |-- steamcannon-rubygem-do_sqlite3-010.spec
|   |-- steamcannon-rubygem-eventmachine-012.spec
|   |-- steamcannon-rubygem-extlib-09.spec
|   |-- steamcannon-rubygem-json-14.spec
|   |-- steamcannon-rubygem-mime-types-116.spec
|   |-- steamcannon-rubygem-open4-10.spec
|   |-- steamcannon-rubygem-rack-10.spec
|   |-- steamcannon-rubygem-rack-test-05.spec
|   |-- steamcannon-rubygem-rake-08.spec
|   |-- steamcannon-rubygem-rcov-09.spec
|   |-- steamcannon-rubygem-rest-client-16.spec
|   |-- steamcannon-rubygem-rspec-13.spec
|   |-- steamcannon-rubygem-sinatra-10.spec
|   |-- steamcannon-rubygem-steamcannon-thin-12.spec
|-- src
    |-- javasysmon-0.3.3.jar
    |-- mobicents-load-balancer.init
    |-- mobicents-sip-servlets.init
    |-- monitor.jar
    |-- steamcannon-agent.init
```

A.2 Monitor

Monitor is a MBean. It provides a method that returns measurements for each metric defined in Section 9.1.

The uptime and the available memory are given by the JavaSysMon library [28]. The available JVM memory is given by the methods `maxMemory` and `totalMemory` of the `Runtime` class [11]. The CPU load is given by the method `getSystemLoadAverage` of the `OperatingSystemMXBean` class [10].

The Listing A.2 shows the tree of the monitor.

Listing A.2 Monitor's tree

```
.
|-- README
|-- build
|-- ejbModule
|   |-- META-INF
|   |   |-- MANIFEST.MF
|   |   |-- jboss-service.xml
|   |-- org
|       |-- mobicents
|           |-- cloud
|               |-- monitor
|                   |-- CloudMonitor.java
|                   |-- CloudMonitorMBean.java
|-- javasysmon-0.3.3.jar
```

A.3 Scaler

The scaler provides the deciding module of the management system described in Section 9.1. The scaler uses the REST API provided by SteamCannon in order to get informations on the running environments and instances. It triggers the monitor of the instances by using Twiddle [52]. Finally, it sends the commands to start or stop an instance to the REST API of SteamCannon.

The scaler uses the `commons-cli-1.2` library in order to process the arguments of the command line, and the `jdom` library in order to process the XML files that are returned by the REST API of SteamCannon.

The Listing A.3 shows the tree of the scaler.

A.4 SteamCannon

SteamCannon has been modified in order to support a Mobicents Sip Servlets platform.

The Listing A.4 shows the diff between the official SteamCannon server component and our improvement. The Listing A.5 shows the diff between the official SteamCannon client component and our improvement.

Listing A.3 Scaler's tree

```
.
|-- bin
|-- build
|   |-- build_jar.sh
|   |-- manifest.txt
|-- exe
|   |-- scaler.sh
|-- lib
|   |-- commons-cli-1.2
|   |-- jdom
|-- src
    |-- org
        |-- mobicents
            |-- cloud
                |-- scaler
                |   |-- client
                |   |   |-- HTTPClient.java
                |   |   |-- Logger.java
                |   |   |-- Parameters.java
                |   |   |-- PostParameter.java
                |   |   |-- Scaler.java
                |   |   |-- Test.java
                |   |-- model
                |       |-- Diary.java
                |       |-- DiaryEntry.java
                |       |-- Environment.java
                |       |-- Image.java
                |       |-- Instance.java
                |       |-- InstanceMetrics.java
                |       |-- Metrics.java
                |       |-- Threshold.java
                |-- xml
                    |-- RelFilter.java
                    |-- XMLUtils.java
```

Listing A.4 SteamCannon server's diff

```
 .gitignore                                          |    1 +
 app/controllers/instances_controller.rb             |    2 +-
 .../agent_services/mobicents_sip_servlets.rb        |   55 ++++++++++++++++++++
 app/models/instance.rb                              |    2 +-
 app/views/instances/_instance.xml.haml              |    2 +-
 db/fixtures/services/jboss_services.yml             |    9 +++-
 lib/tasks/load_platforms.rake                       |    2 +-
 lib/tasks/load_services.rake                        |   15 +++++-
 8 files changed, 83 insertions(+), 5 deletions(-)
```

Listing A.5 SteamCannon client's diff

```
.gitignore                                    |    1 +
Gemfile                                       |    2 +-
Gemfile.lock                                  |   14 ++++--
config/thin/dev.yaml                          |   15 +++
.../commands/check-status-command.rb          |   80 ++++++++++++++++
.../commands/configure-command.rb             |   95 ++++++++++++++++++++
.../commands/monitor-command.rb               |   50 ++++++++++
.../mobicents-sip-servlets-service.rb         |   74 ++++++++++++++++
8 files changed, 323 insertions(+), 8 deletions(-)
```

A.5 Examples

Examples contains the implementation of the uses cases discussed in Chapter 6. The Appendix B.3 explains how to deploy those examples on the Mobicents Sip Servlets cloud platform.

On one hand, the apps/ folder contains two applications which correspond to the two use cases. Each use case has been implemented as a SIP application. The apps/wars/ folder contains the packaged applications, and a script called configure-DAR.sh. This script can be used in order to configure the application router of the Mobicents Sip Servlets instance where are installed the applications.

On the other hand, the agents/ folder contains a script called build-agents.sh. This script can be used in order to generate two SIPp agents. The SIPp agents will be used in order to test the SIP applications.

The Listing A.6 shows the tree of the examples.

Listing A.6 Example's tree

```
.
|-- agents
|   |-- build-agents.sh
|   |-- scenarios
|       |-- uac-proxy.xml
|       |-- uas-proxy.xml
|-- apps
|   |-- b2bua-service
|   |   |-- WebContent
|   |   |   |-- META-INF
|   |   |   |   |-- MANIFEST.MF
|   |   |   |-- WEB-INF
|   |   |-- build
|   |   |-- src
|   |       |-- org
|   |           |-- mobicents
|   |               |-- cloud
|   |                   |-- servlet
|   |                       |-- sip
|   |                           |-- example
|   |                               |-- B2BUAServlet.java
|   |                               |-- package-info.java
|   |-- proxy-service
|   |   |-- WebContent
|   |   |   |-- META-INF
|   |   |   |   |-- MANIFEST.MF
|   |   |   |-- WEB-INF
|   |   |-- build
|   |   |-- src
|   |       |-- org
|   |           |-- mobicents
|   |               |-- cloud
|   |                   |-- servlet
|   |                       |-- sip
|   |                           |-- ProxyServlet.java
|   |                           |-- package-info.java
|   |-- wars
|       |-- b2bua-service.war
|       |-- configure-DAR.sh
|       |-- local.sh
|       |-- proxy-service.war
|-- read_me.txt
```

Appendix B

A step-by-step tutorial

This tutorial explains first how to build the appliances for the cloud. Three appliances are built: one for Mobicents Sip Servlets, one for the SIP load balancer and one for the management.

Then, this tutorial explains how to use and customize the appliances. The Mobicents Sip Servlets appliance and the SIP load balancer appliance will be used to build a Mobicents Sip Servlets cluster. The management appliance will be used to monitor the instances and to provide auto-scaling.

Finally, this tutorial explains how to deploy two use case applications to the cloud.

Contents

B.1 Build the appliances

B.1.1 Build the Mobicents Sip Servlets and Mobicents Load Balancer appliances

1. Start a new instance of the BoxGrinder meta appliance on Amazon EC2. The latest AMI id can be found here: http://boxgrinder.org/download/boxgrinder-build-meta-appliance/. The AMI `ami-888d7fe1` has been used for this tutorial.

2. Login to the instance of the BoxGrinder meta appliance you have just created. The user is `ec2-user` and has a full sudo access (no password required). Install some dependencies needed later:

```
sudo yum update -y && sudo yum install -y svn
```

3. Check-out the cirrasM project from SVN:

```
svn checkout https://mobicents.googlecode.com/svn/trunk/cloud/sip-
    servlets/cirrasM cirrasM
cd cirrasM/
```

4. Edit BoxGrinder's config file according to your credentials and move it to /root/.boxgrinder/config.

```
vi config
sudo mkdir /root/.boxgrinder
sudo mv config /root/.boxgrinder/
```

5. Build the appliances by launching the script. By default, this script build EBS backed images.

```
./build-cirrasM.sh
```

   This step may take a long time (*i.e.* more than one hour). You can see the log in the `log` folder.

B.1.2 Build the management appliance

1. Start a new instance of SteamCannon 0.2.0. The community AMI is `ami-6ea25307`. The user is `root` and has a full sudo access (no password required).

2. Patch the SteamCannon software by issuing the following commands:

```
yum update -y && yum install -y svn
cd /opt/
sudo rm -r steamcannon
sudo -u jboss-as6 svn checkout https://mobicents.googlecode.com/svn/
    trunk/cloud/sip-servlets/steamcannon
cd
```

3. Reinitialize SteamCannon

```
cd /opt/steamcannon/
/etc/init.d/jboss-as stop
/opt/jruby/bin/jruby -S rake db:drop:all
./bin/appliance_initialization.sh
/etc/init.d/jboss-as start;
cd
```

B.2 Use the appliances

B.2.1 Mobicents load balancer appliance

This appliance is ready to be used out of the box. The user is `ec2-user` and has a full sudo access (no password required). The logs are available in the file /server/mobicents-load-balancer/console.log. You can manage Mobicents load balancer by using the following command:

```
sudo service mobicents-load-balancer (start | stop | status | restart | help)
```

B.2.2 Mobicents Sip Servlets appliance

1. Launch a new instance of the Mobicents Sip Servlets appliance. The user is `ec2-user` and has a full sudo access (no password needed).

2. Set-up the IP of the load balancer. Open the file /etc/sysconfig/mobicents-sip-servlets and edit the `MOBICENTS_LB_IP` property.

   ```
   sudo vi /etc/sysconfig/mobicents-sip-servlets
   ```

   If you want to use an elastic IP for the SIP load balancer, you can set the `MOBICENTS_LB_IP` property to the following

   ```
   MOBICENTS_LB_IP=‘dig +short <public_dns>‘
   ```

   Where `<public_dns>` is the public DNS of your elastic IP. For example, if your elastic IP is `50.16.205.218`, your public DNS will be something like `ec2-50-16-205-218.compute-1.amazonaws.com`.

3. Consider installing the use cases if you want to use them (Section B.3.1), and jump to the step 6.

   Otherwise, upload your applications to the container. From your computer:

   ```
   scp -i <your_keyfile.pem> <your_application.war> ec2-user@<
       your_instance_ip>:/home/ec2-user
   ```

   From your instance:

```
sudo mv /home/ec2-user/*.war /server/mobicents-sip-servlets/server/all/
    deploy/
```

4. Edit the dar configuration file.

```
sudo vi /server/mobicents-sip-servlets/server/all/conf/dars/mobicents-
    dar.properties
```

5. Restart the Mobicents Sip Servlets container in order to apply the changes. (Do not forget it!)

```
sudo service mobicents-sip-servlets restart
```

6. Once you have customized your appliance, create a new image of this appliance in the Amazon EC2 console (See Figure B.1). You can now start or stop as many instances of this new image as you wish.

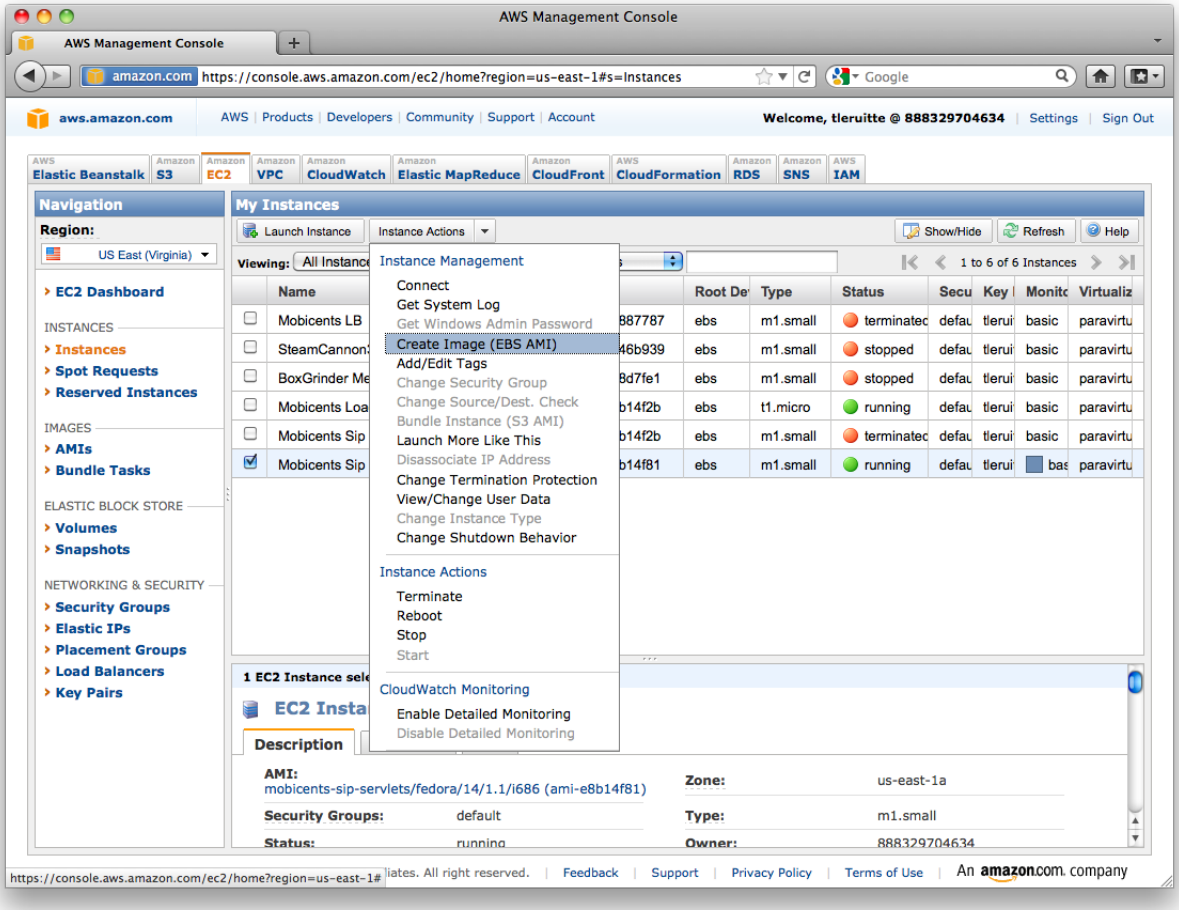

Figure B.1: Create an EBS image from the running instance.

You can manage Mobicents Sip Servlets by using the following command:

```
sudo service mobicents-sip-servlets (start | stop | status | restart | help)
```

B.2.3 Management appliance

Add a platform

1. Register a new environment, containing your Mobicents Sip Servlets appliance created before (See Subsection B.2.2, Step 6). First, copy this content to a file named `mobicents-sip-servlets.yml` and edit the `cloud_id` to the id of your AMI. If you do not have a 64 bits image, just put the same id in both sections.

```
---
platforms:
  - name: Mobicents Sip Servlets Platform
    platform_versions:
      - version_number: 1
        images:
          - name: Mobicents Sip Servlets
            description: Mobicents Sip Servlets 1.5
            uid: mss-1
            can_scale_out: true
            services:
              - mobicents_sip_servlets
            cloud_images:
              - cloud: ec2
                region: us-east-1
                architecture: i386
                cloud_id: <your_ami>
              - cloud: ec2
                region: us-east-1
                architecture: x86_64
                cloud_id: <your_ami>
```

2. Then, register this environment in SteamCannon.

```
cd /opt/steamcannon/
export RAILS_ENV=production
export FILE=/root/mobicents-sip-servlets.yml
/opt/jruby/bin/jruby -S rake app:platforms:load_from_yaml
cd
```

If you want to remove a platform, you can reinitialize the database of SteamCannon:

```
cd /opt/steamcannon
/etc/init.d/jboss-as stop
/opt/jruby/bin/jruby -S rake db:drop:all
./bin/appliance_initialization.sh
/etc/init.d/jboss-as start
cd
```

Control the environment

1. Ensure that the port 8080 is open for all traffic in the security group within which your SteamCannon instance is running.

2. By default, SteamCannon start new instances in a new security group called *steamcannon*. Ensure that all the TCP, UDP and ICMP traffic is allowed between the *steamcannon* security group and the security group within your Mobicents load balancer is running (in both directions).

3. Open a web browser on http://⟨your_instance's_public_ip⟩:8080 and register an account (Figure B.2). Make sure to use Firefox when you use the web interface. (A bug in SteamCannon prevents it from working in Safari.)

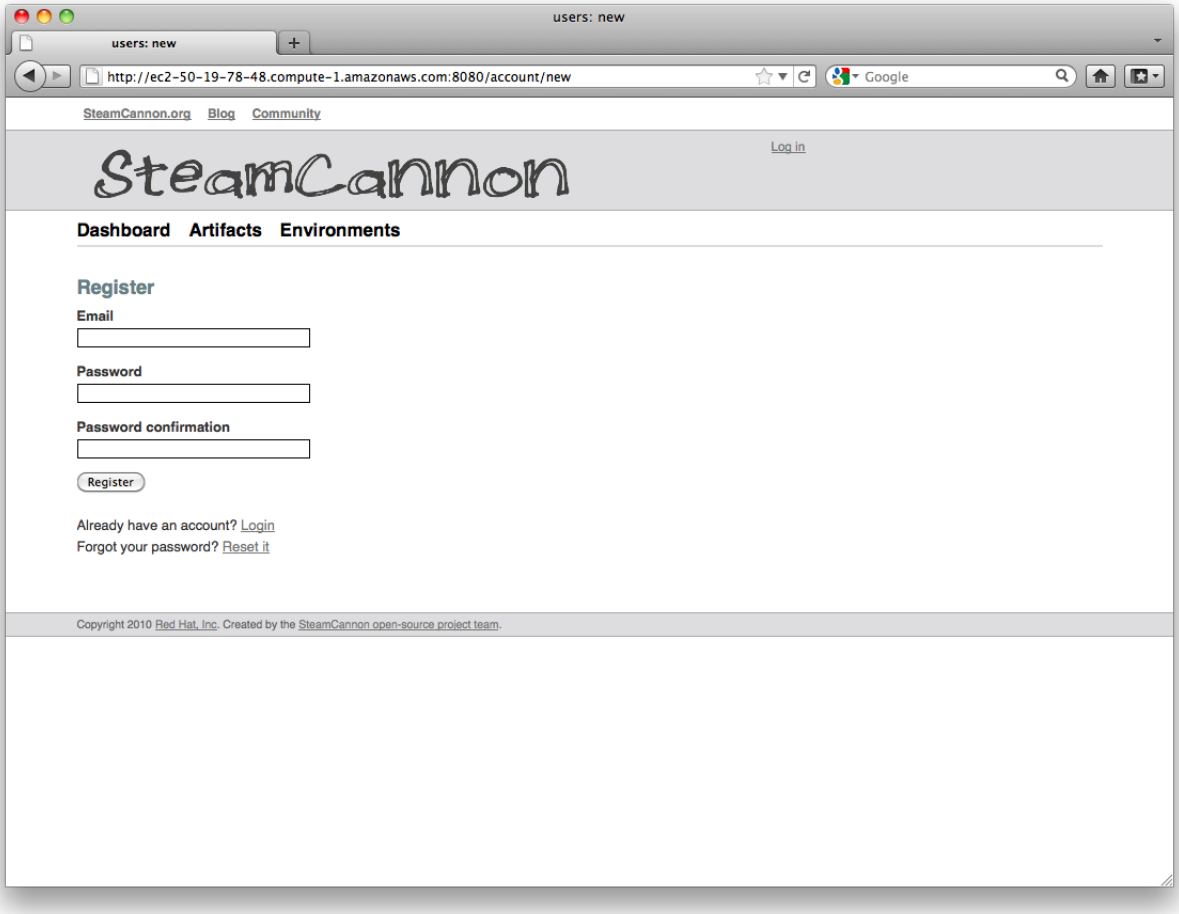

Figure B.2: Register an account on SteamCannon.

4. Create a new environment of the platform you have created in the previous section (Figure B.3). You can start and stop Mobicents Sip Servlets instances within this environment.

Auto-scaling

1. You need a working copy of Twiddle, with the same version than the one distributed with JBoss AS 5.1. The simplest way to get it is to get a fresh copy of Mobicents Sip Servlets.

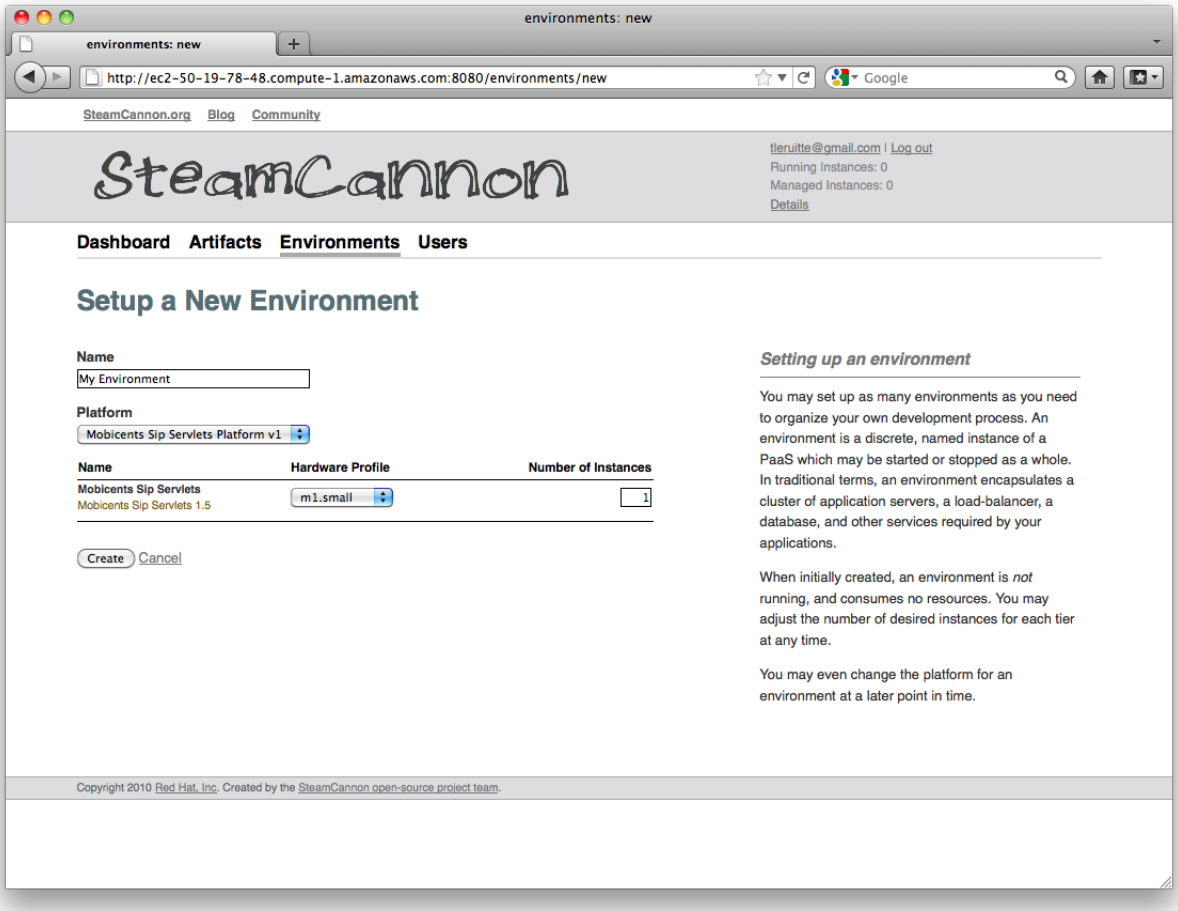

Figure B.3: Create an environment.

```
yum install -y unzip
wget http://sourceforge.net/projects/mobicents/files/Mobicents%20Sip%20
    Servlets/Mobicents%20Sip%20Servlets%201.5.0.FINAL/mss-1.5.0.FINAL-
    jboss-jdk6-5.1.0.GA-1012212016-LGPL.zip/download
unzip mss-1.5.0.FINAL-jboss-jdk6-5.1.0.GA-1012212016-LGPL.zip
sudo rm mss-1.5.0.FINAL-jboss-jdk6-5.1.0.GA-1012212016-LGPL.zip
```

2. Install the scaler.

```
sudo yum install -y svn
svn checkout https://mobicents.googlecode.com/svn/trunk/cloud/sip-
    servlets/scaler
```

3. *(Optional.)* Personalize the monitoring. Two files are worth checking. Those are scaler/src/org/mobicents/cloud/scaler/client/Parameters.java and scaler/src/org/mobicents/cloud/scaler/model/Threshold.java.

4. Build the scaler executable.

```
sudo yum install -y java-1.6.0-openjdk-devel
cd scaler/build/
```

```
./build_jar.sh
cd
```

5. Make sure an environment is running (Section B.2.3).

6. Start the monitoring process.

```
cd scaler/exe/
./scaler.sh -s localhost -u <your_steamcannon_user> -p <
    your_steamcannon_password>
```

B.3 Deploy the use cases

B.3.1 Install the use cases

On the Mobicents Sip Servlet appliance, follow these steps:

1. Check-out the use cases from the SVN.

```
sudo yum -y install svn
svn checkout https://mobicents.googlecode.com/svn/trunk/cloud/sip-
    servlets/examples/apps/wars
cd wars
```

2. Move the applications to the container.

```
sudo mv *.war /server/mobicents-sip-servlets/server/all/deploy/
```

3. Edit the default application router for the use case you want to use.

```
sudo ./configure-DAR.sh [proxy|b2bua]
```

B.3.2 Launch the SIPp agents

You can install the agents on any host. However, the UAC and the UAS must be on the same host for the B2BUA application. Follow these steps to install and launch the agents:

1. Check-out the agents from the SVN.

```
svn checkout https://mobicents.googlecode.com/svn/trunk/cloud/sip-
    servlets/examples/agents
cd agents
```

2. Generate the agents for the application you have installed (Section B.3.1).

```
./build-agents.sh [proxy|b2bua]
```

3. Launch the UAS.

```
./sipp-server.sh
```

4. From another console, launch the UAC. You can modify the rate at which the messages are sent by using the keys '+' and '-'.

```
./sipp-client.sh
```

Bibliography

[1] *Delatacloud API*. http://incubator.apache.org/deltacloud/.

[2] *EC2 Disabled*. http://ec2disabled.com/.

[3] *Elastichosts*. http://www.elastichosts.com/.

[4] *Mobicents SVN*. http://code.google.com/p/mobicents/source/browse/.

[5] *Rumpler*. https://github.com/torquebox/rumpler/.

[6] *SIPp*. http://sipp.sourceforge.net/.

[7] *YAML*. http://www.yaml.org/.

[8] *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. Standard ISO/IEC 7498-1:1994, International organization for standardization, 1994. http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html.

[9] *The sip servlet tutorial*. Documentation, Sun Microsystems, Inc., January 2009. http://download.oracle.com/docs/cd/E19355-01/820-3007/820-3007.pdf.

[10] *OperatingSystemMXBean class*. Java API, Oracle, 2011. http://download.oracle.com/javase/6/docs/api/java/lang/management/OperatingSystemMXBean.html.

[11] *Runtime class*. Java API, Oracle, 2011. http://download.oracle.com/javase/6/docs/api/java/lang/Runtime.html.

[12] *SIP2SIP*, January 2011. http://wiki.sip2sip.info/.

[13] CLOUDSIGMA AG: *CloudSigma*. http://cloudsigma.com/.

[14] Amazon.com: *Amazon Simple Storage Service (Amazon EBS)*. http://aws.amazon.com/ebs/.

[15] Amazon.com: *Amazon Simple Storage Service (Amazon S3)*. http://aws.amazon.com/s3/.

[16] Amazon.com: *Amazon Web Services*. http://aws.amazon.com/.

[17] Galder Zamarreno Brian Stansberry and Paul Ferraro: *JBoss AS 5.1 Clustering Guide*. Documentation, Red Hat, Inc., September 2009. http://docs.jboss.org/jbossclustering/cluster_guide/5.1/html-single/index.html.

[18] Mobicents Community: *Mobicents in the cloud*. Documentation, Red Hat, 2011. https://docs.jboss.org/author/display/MOBICENTS/Mobicents+in+the+cloud.

[19] Bart Czernicki: *IaaS, PaaS and SaaS Terms Clearly Explained and Defined*. Blog post, Silverlight Hack, February 2011. http://silverlighthack.com/post/2011/02/27/IaaS-PaaS-and-SaaS-Terms-Explained-and-Defined.aspx.

[20] Thibault Dory: *Study and comparison of elastic cloud databases: Myth or reality?* Master's thesis, Louvain School of Engineering, 2011.

[21] Quentin Dugauthier: *SIP : Le protocole d'initialisation de session*. May 2005.

[22] Eucalyptus Systems, Inc.: *Eucalyptus*. http://www.eucalyptus.com/.

[23] Ian Evans: *Your First Cup: An Introduction to the Java EE Platform*. Documentation, Oracle, November 2010. http://download.oracle.com/javaee/6/firstcup/doc/.

[24] Eric Foster-Johnson, Stuart Ellis, and Ben Cotton: *RPM Guide*. Documentation, Red Hat, Inc., 2010. http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/.

[25] Daniel Fuchs: *What is JMX?* Blog post, Oracle, September 2006. http://blogs.oracle.com/jmxetc/entry/what_is_jmx.

[26] Google: *AppEngine*. http://code.google.com/appengine/.

[27] SKALI GROUP: *SKALI Cloud*. http://www.skalicloud.com/.

[28] Jez Humble: *JavaSysMon*. https://github.com/jezhumble/javasysmon/wiki.

[29] Eric Jendrock, Jennifer Ball, Debbie Carson, Ian Evans, Scott Fordin, and Kim Haase: *The Java EE 5 Tutorial*. Documentation, Oracle, September 2010. http://download.oracle.com/javaee/5/tutorial/doc/.

[30] Charles M. Kozierok: *The TCP/IP Guide: DNS Name Server Load Balancing*. Technical report, 2010. http://www.tcpipguide.com/free/t_DNSNameServerLoadBalancing.htm.

[31] Mihir Kulkarni and Yannis Cosmadopoulos: *Sip servlet 1.1*. Specification JSR 289, Oracle, August 2008. http://jcp.org/aboutJava/communityprocess/final/jsr289/index.html.

[32] Thibault Leruitte: *BGBUILD-193*. Bug report, BoxGrinder Build, 2011. https://issues.jboss.org/browse/BGBUILD-193.

[33] Anthony Liegeois: *Security validation of a peer-to-peer system*. Master's thesis, Louvain School of Engineering, 2011.

[34] Melbourne Server Hosting Ltd: *Serverlove*. http://www.serverlove.com/.

[35] Peter Mell and Tim Grance: *The NIST Definition of Cloud Computing*. National Institute of Standards and Technology, 53(6):50, 2009. http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc.

[36] Mobicents team: *Mobicents Sip Servlets user guide*. Documentation, Red Hat, Inc., 2011. http://hudson.jboss.org/hudson/view/Mobicents/job/MobicentsBooks/lastSuccessfulBuild/artifact/sip-servlets/index.html.

[37] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov: *Eucalyptus: A Technical Report on an Elastic Utility Computing Archietcture Linking Your Programs to Useful Systems*. UCSB Computer Science Technical Report, (2008-10), 2008.

[38] Open Hosting Inc: *Open Hosting*. http://www.openhosting.com/.

[39] Oracle: *Java Naming and Directory Interface (JNDI)*. http://www.oracle.com/technetwork/java/jndi/index.html.

[40] Oracle: *VirtualBox*. http://www.virtualbox.org/.

[41] Red Hat, Inc.: *Cantiere*. http://www.jboss.org/stormgrind/projects/cantiere.html.

[42] Red Hat, Inc.: *CirrAS*. http://www.jboss.org/stormgrind/projects/cirras.

[43] Red Hat, Inc.: *JBoss AS*. http://www.jboss.org/jbossas/.

[44] Red Hat, Inc.: *Mobicents Sip Servlets*. http://www.mobicents.org/products_sip_servlets.html.

[45] Red Hat, Inc.: *mod_cluster*. http://www.jboss.org/mod_cluster/.

[46] Red Hat, Inc.: *Openshift*. http://openshift.redhat.com/app/.

[47] Red Hat, Inc.: *Red Hat*. http://www.redhat.com/.

[48] Red Hat, Inc.: *RHQ*. http://rhq-project.org.

[49] Red Hat, Inc.: *Steamcannon*. http://steamcannon.org/.

[50] Red Hat, Inc.: *steamcannon-appliances code repository*. https://github.com/steamcannon/steamcannon-appliances.

[51] Red Hat, Inc.: *Torquebox*. http://torquebox.org/.

[52] Red Hat, Inc.: *Twiddle*. http://community.jboss.org/wiki/Twiddle.

[53] RHQ: *Is it possible to monitor JBoss AS 5.1?* Faq, Red Hat, Inc., 2011. http://rhq-project.org/display/JOPR2/FAQ#FAQ-IsitpossibletomonitorJBossAS5.1%3F.

[54] J. Rosenberg, H. Schulzrinne, Columbia U., G. Camarillo, Ericsson, A. Johnston, WorldCom, J. Peterson, Neustar, R. Sparks, dynamicsoft, M. Handley, ICIR, E. Schooler, and AT&T: *Sip: Session initiation protocol*. Proposed standard RFC 3261, The Internet Engineering Task Force, June 2002. http://tools.ietf.org/html/rfc3261.

[55] Peter Van Roy: *The clouds of revolution*. Université catholique de Louvain, 2010.

[56] Dmitriy Samovskiy: *Security Groups - Most Underappreciated Feature of Amazon EC2*. Blog post, Dmitriy Samovskiy's Blog, September 2009. http://www.somic.org/2009/09/21/security-groups-most-underappreciated-feature-of-amazon-ec2/.

[57] H. Schulzrinne, Columbia U., A. Rao, Netscape, R. Lanphier, and RealNetworks: *Real Time Streaming Protocol (RTSP)*. Proposed standard RFC2326, The Internet Engineering Task Force, April 1998. http://tools.ietf.org/html/rfc2326.

[58] System Virtualization, Partitioning, and Clustering Working Group: *Open Virtualization Format Specification*. Specification DSP0243, Distributed Management Task Force, January 2010. http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_1.1.0.pdf.

[59] The Amazon Web Services team: *Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region*. Technical report, Amazon, 2011. http://aws.amazon.com/message/65648/.

[60] The Apache Software Foundation: *Apache Tomcat*. http://tomcat.apache.org/.

[61] VMware, Inc.: *VMWare*. http://www.vmware.com/.

[62] Simon Wardley, Etienne Goyer, and Nick Barcet: *Ubuntu enterprise cloud architecture*. Technical white paper, Canonical, August 2009. http://i.dell.com/sites/content/business/solutions/cloud-computing/en/Documents/ubuntu-enterprise-cloud-architecture.pdf.

[63] Wikipedia: *Application server*, 2011. http://en.wikipedia.org/wiki/Application_server.

[64] Wikipedia: *Virtual machine*, 2011. http://en.wikipedia.org/wiki/Virtual_machine.

Acknowledgments

I would like to thank both my promoters, Peter Van Roy and Sabri Skhiri, for having proposed this thesis and for their support and advices during the whole academic year.

I would like to thank the Mobicents community, and more generally the whole JBoss AS community. I especially would like to thank Jean Deruelle, who introduced me to the Mobicents community and who gave me great feedbacks and guidance during this work. I also would like to thank Marek Goldmann (aka mgoldmann on #boxgrinder) for its great support of BoxGrinder, and bbrowning, tcrawley and bobmcw (on #steam-cannon) for their precious help with SteamCannon.

I would like to thank Boriss Mejias and Nam-Luc Tran for their help, advices and review during the whole academic year. I also would like to thank Thibault Dory and Julian Janssens for their final review.

Finally, I am grateful to Alexandra Collignon for her moral support.