IMS MESSAGING GATEWAY IN THE CLOUD

Master’s Thesis, 10 March 2015

Pierre López Barbosa
IMS MESSAGING GATEWAY
IN THE CLOUD

THESIS

submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER SCIENCE
TRACK INFORMATION ARCHITECTURE

by

Pierre López Barbosa
born in Willemstad, Curaçao
IMS MESSAGING GATEWAY
IN THE CLOUD

Author: Pierre López Barbosa
Student id: 1396412
Email: me@piloba.com

Abstract
For mobile messaging service providers to endure in a competitive and dynamic market, it is vital to be flexible, which involves keeping up to date to technological developments, and to be cost effective. In order for service providers to provide highly available service, the use of Cloud computing technology is a well-suited solution. Cloud computing allows resources to be provided as general utilities, which users can lease and release in an on-demand fashion through the Internet. In this thesis an HTTP based mobile messaging solution is designed, implemented and described and deployed in the Amazon Cloud, with the main goal of determining to what extent the availability of mobile messaging services improves when deployed in the Amazon Cloud. Three testing environments are proposed for deploying mobile messaging services, i.e. using a proprietary server, Auto Scaling service and the Elastic Load Balancing service provided by Amazon. The architectures for all three scenarios are described and illustrated, and a description is provided of their implementation. Based on performance tests executed in all three scenarios the improvement in availability is determined. Also stress tests are executed in all scenarios with the purpose to compare the performance, i.e. the average response time required to process a subscriber’s request for service, of each scenario with each other. A comparison of the test results provides insight into the availability of the tested messaging service, and the relationship that exists between each scenario's performance.

Keywords: IMS, Cloud computing, SMS, HTTP, performance test, availability, auto scaling, load balancing

Graduation Committee
Chair: Prof. Dr. Ir. G.J.P.M. Houben, Faculty EEMCS, TUDelft
University supervisor: Dr. Ir. A.J.H. Hidders, Faculty EEMCS, TUDelft
Committee Member: Dr. Ir. A. Zaidman, Faculty EEMCS, TUDelft
Company supervisor: T. Martinez, ICTual Mobile Solutions
Acknowledgements

In this section I would like to mention and thank a number of people without whom this thesis would not have been written, and to whom I would like to express my deepest appreciation. First of all, I would like to express my sincere gratitude to my dear God, who gave me the strength and wisdom to be able to finish this thesis.

This thesis was executed at ICTual Mobile Solutions, formerly T-Sparx, in Curacao. I am really thankful that I had the opportunity to execute this thesis in my country of birth, and gain practical experience in the areas of telecommunications and Cloud computing. I would like to thank Shurbann Martes, Geraume Bor and Ted Martinez, who guided me during my thesis project and provided me with the valuable knowledge required to carry out this thesis project.

I would like to thank the members of the University. First of all, my daily supervisor, Jan Hidders, for his valuable feedback, scientific guidance and patience during this period. I would also like to thank my committee members Professor Geert-Jan Houben and Andy Zaidman for reviewing this document and serving on the defense committee.

I would also like to thank my other relatives, friends and colleagues for their faith in my perseverance and commitment, and for all their encouragement during this process. Finally, I would like to acknowledge all the persons that helped me by providing me with information to be able to finalize my thesis.

Last but not least, I thank my parents and sister for their continuous support and motivation throughout the entire finalization phase.

Pierre López Barbosa
10 March 2015
# Table of contents

**LIST OF FIGURES**                                                                                          ........................................... IV  
**LIST OF TABLES**                                                                                             ......................................... VI  

## CHAPTER 1 INTRODUCTION .................................................................................................................................. 1  
1.1 PROBLEM DESCRIPTION ................................................................................................................................... 1  
1.2 OBJECTIVE ..................................................................................................................................................... 6  
1.3 ASSUMPTIONS .................................................................................................................................................. 11  
1.4 RESEARCH STRATEGY ...................................................................................................................................... 12  
1.5 DELIVERABLES ............................................................................................................................................... 13  
1.6 RELEVANCY OF RESEARCH .............................................................................................................................. 14  
1.7 SCOPE OF RESEARCH ...................................................................................................................................... 14  
1.8 THESIS OUTLINE ............................................................................................................................................ 15  

## CHAPTER 2 SHORT MESSAGE SERVICE MESSAGING SOLUTION ...................................................................................... 17  
2.1 INTRODUCTION TO SHORT MESSAGE SERVICE AND HYPERTEXT TRANSFER PROTOCOL ..................................... 17  
  2.1.1 SHORT MESSAGE PEER-TO-PEER PROTOCOL .............................................................................................. 17  
  2.1.2 HYPERTEXT TRANSFER PROTOCOL ........................................................................................................ 19  
2.2 CURRENT MESSAGING SOLUTION ARCHITECTURE AND INFRASTRUCTURE ARCHITECTURE ................................ 20  
  2.2.1 THE GSM STANDARD .................................................................................................................................. 20  
  2.2.2 JIVER .......................................................................................................................................................... 23  
  2.2.3 MESSAGING SOLUTION ARCHITECTURE ................................................................................................. 27  
  2.2.4 MESSAGING SOLUTION INFRASTRUCTURE ARCHITECTURE .................................................................... 29  
2.3 PROBLEMS FACED WITH CURRENT MESSAGING INFRASTRUCTURE ARCHITECTURE ...................................... 30  
2.4 POSSIBLE SOLUTIONS AND CHALLENGES ....................................................................................................... 31  

## CHAPTER 3 IP MULTIMEDIA SUBSYSTEM ................................................................................................................ 33  
3.1 IMS DEFINITION .................................................................................................................................................. 33  
3.2 IMS HISTORY ...................................................................................................................................................... 35  
3.3 IMS STANDARDS ................................................................................................................................................ 36  
  3.3.1 SESSION CONTROL PROTOCOL ................................................................................................................ 36  
  3.3.2 DIAMETER: THE AUTHENTICATION, AUTHORIZATION AND ACCOUNTING (AAA) PROTOCOL .................. 36  
  3.3.3 REAL-TIME TRANSPORT PROTOCOL AND RTP CONTROL PROTOCOL .................................................. 36  
  3.3.4 H.248 MEDIA CONTROL PROTOCOLS ..................................................................................................... 36  
  3.3.5 IPv6 ............................................................................................................................................................ 37  
3.4 IMS ARCHITECTURE AND FEATURES ............................................................................................................... 37  
  3.4.1 ACCESS NETWORK ....................................................................................................................................... 39  
  3.4.2 THE HOME SUBSCRIBER SERVER (HSS) AND SUBSCRIPTION LOCATOR FUNCTION (SLF) ......................... 39  
  3.4.3 IDENTIFICATION IN IMS ............................................................................................................................ 40  
  3.4.4 THE CALL/SESSION CONTROL FUNCTION COMPONENTS ....................................................................... 41  
  3.4.5 THE APPLICATION SERVER (AS) ............................................................................................................... 42  
  3.4.6 THE MEDIA RESOURCE FUNCTION ......................................................................................................... 42  
  3.4.7 THE BREAKOUT GATEWAY CONTROL FUNCTION (BGCF) ...................................................................... 42  
  3.4.8 IMS APPLICATION LAYER GATEWAY (IMS-ALG) AND TRANSITION GATEWAY (TrGW) ......................... 42  
  3.4.9 THE PSTN/CS GATEWAY ......................................................................................................................... 43  
3.5 FUNDAMENTALS FOR GETTING THE IMS SERVICE ............................................................................................ 43  
3.6 IMS SECURITY ..................................................................................................................................................... 45  
  3.6.1 ACCESS SECURITY ....................................................................................................................................... 45  
  3.6.2 NETWORK SECURITY ................................................................................................................................... 45  
  3.6.3 IMS SECURITY ARCHITECTURE .................................................................................................................. 47  
3.7 IMS SECURITY CHALLENGES .............................................................................................................................. 47  
  3.7.1 IMS SECURITY CHALLENGES FOR NETWORK PROVIDERS ...................................................................... 48  
  3.7.2 IMS SECURITY CHALLENGES FOR NETWORK USERS .............................................................................. 49
List of figures

Figure 1.1 Current messaging solution based on SMPP and HTTP protocols .............................................. 3
Figure 1.2 Abstract model of messaging solution ......................................................................................... 12
Figure 2.1 SMPP interface between SMSC and ESME ................................................................................ 19
Figure 2.2 GSM architecture with essential components ........................................................................... 21
Figure 2.3 GPRS system architecture ....................................................................................................... 22
Figure 2.4 Jiver Application Suite .............................................................................................................. 24
Figure 2.5 Jiver Application Suite more detailed ....................................................................................... 25
Figure 2.6 Jiver Portal ProCharge Supervisor role ..................................................................................... 25
Figure 2.7 RadPas Context diagram ........................................................................................................ 27
Figure 2.8 SMS gateway enabling communication between SMSCs ......................................................... 28
Figure 2.9 SMS Gateway context diagram ................................................................................................ 28
Figure 2.10 Typical Infrastructure architecture of production environment .............................................. 30
Figure 2.11 Abstract view of mobile device with RadPas communication ............................................... 31
Figure 3.1 Layered architecture of IMS ..................................................................................................... 37
Figure 3.2 IMS architecture consisting of majority of components .......................................................... 38
Figure 3.3 IMS architecture highlighting the components needed for basic messaging solution ........... 38
Figure 3.4 IMS architecture with basic components extracted from the complete architecture............. 39
Figure 3.5 PSTN/CS gateway interfacing a circuit-switched network ......................................................... 43
Figure 3.6 High-level view of prerequisites to get the IMS service .......................................................... 44
Figure 3.7 IPsec ESP in transport mode .................................................................................................... 46
Figure 3.8 IPsec ESP in tunnel mode ........................................................................................................ 46
Figure 3.9 IMS Security architecture ....................................................................................................... 47
Figure 4.1 Cloud actors .............................................................................................................................. 52
Figure 4.2 Type I Hypervisor .................................................................................................................... 53
Figure 4.3 Type II Hypervisor .................................................................................................................. 53
Figure 5.1 IMS messaging solution architecture ....................................................................................... 68
Figure 5.2 Context diagram IMS messaging solution .............................................................................. 68
Figure 5.3 OpenNebula architecture ....................................................................................................... 71
Figure 6.1 Context diagram HTTP messaging solution simulation ............................................................ 75
Figure 6.2 BB_ON request from client to MessagingGateway with the first corresponding forwarding route .................................................................................................................. 75
Figure 6.3 BB_ON request from client to MessagingGateway with the second corresponding forwarding route .................................................................................................................. 76
Figure 6.4 Model View Controller architecture ......................................................................................... 78
Figure 6.5 High level view of the requesting processing workflow in Spring Web MVC ........................... 78
Figure 6.6 Proprietary server scenario infrastructure .............................................................................. 88
Figure 6.7 Amazon AWS infrastructure ................................................................................................... 88
Figure 6.8 Health Check configuration ..................................................................................................... 93
Figure 6.9 Amazon AWS Auto Scaling infrastructure ............................................................................ 94
Figure 6.10 Amazon Elastic Load Balancing ............................................................................................ 95
Figure 6.11 HTTP Messaging solution using Amazon Elastic Load Balancing infrastructure .......... 97
Figure 6.12 HTTP messaging solution on the Amazon AWS Platform using a dedicated Server ....... 98
Figure 6.13 HTTP messaging solution in the Cloud using Auto Scaling ................................................. 98
Figure 6.14 HTTP messaging solution in the Cloud using Elastic Load Balancing ............................ 99
Figure 7.1: Thread group in JMeter ......................................................................................................... 105
Figure 7.2: HTTP request configuration in JMeter ................................................................................... 105
Figure 7.3: Random Variables in JMeter ................................................................................................ 106
Figure 7.4: JMeter test plan proprietary server ......................................................................................... 110
Figure 7.5: Proprietary server scenario test results graph ......................................................................... 112
Figure 7.6: JMeter test plan Auto Scaling scenario .................................................................................. 113
List of tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Security group rules</td>
</tr>
<tr>
<td>6.2</td>
<td>Hardware specification m3.xlarge EC2 instance</td>
</tr>
<tr>
<td>6.3</td>
<td>Hardware specification m1.xlarge EC2 instance</td>
</tr>
<tr>
<td>6.4</td>
<td>Hardware specification m3.xlarge EC2 instance</td>
</tr>
<tr>
<td>6.5</td>
<td>Hardware specification c3.xlarge EC2 instance</td>
</tr>
<tr>
<td>6.6</td>
<td>Hardware specification m2.2xlarge EC2 instance</td>
</tr>
<tr>
<td>6.7</td>
<td>Hardware specification i2.xlarge EC2 instance</td>
</tr>
<tr>
<td>7.1</td>
<td>Performance test methodology</td>
</tr>
<tr>
<td>7.2</td>
<td>Test plan thread number</td>
</tr>
<tr>
<td>7.3</td>
<td>Heap space allocation</td>
</tr>
<tr>
<td>7.4</td>
<td>Proprietary instance test plan results reaching maximum number of open ports available</td>
</tr>
<tr>
<td>7.5</td>
<td>Proprietary server scenario total processed requests overview</td>
</tr>
<tr>
<td>7.6</td>
<td>Proprietary instance test plan results reaching maximum allocated memory</td>
</tr>
<tr>
<td>7.7</td>
<td>Proprietary server scenario test results sample</td>
</tr>
<tr>
<td>7.8</td>
<td>Auto Scaling test plan results reaching maximum allocated memory</td>
</tr>
<tr>
<td>7.9</td>
<td>Auto Scaling scenario test results sample</td>
</tr>
<tr>
<td>7.10</td>
<td>Amazon Elastic Load Balancer test results</td>
</tr>
<tr>
<td>7.11</td>
<td>Elastic Load Balancer scenario test results sample</td>
</tr>
<tr>
<td>7.12</td>
<td>Custom load balancer test results</td>
</tr>
<tr>
<td>7.13</td>
<td>Custom load balancer scenario test results sample</td>
</tr>
<tr>
<td>7.14</td>
<td>Test results for all scenarios</td>
</tr>
<tr>
<td>7.15</td>
<td>All test results response time proprietary vs. auto-scaling</td>
</tr>
<tr>
<td>7.16</td>
<td>Correct test results response time proprietary vs. auto-scaling</td>
</tr>
<tr>
<td>7.17</td>
<td>All test results response time proprietary vs. Amazon ELB load balancing</td>
</tr>
<tr>
<td>7.18</td>
<td>Correct test results response time proprietary vs. Amazon ELB load balancing</td>
</tr>
<tr>
<td>7.19</td>
<td>All test results response time proprietary vs. Apache2 load balancing</td>
</tr>
<tr>
<td>7.20</td>
<td>Correct test results response time proprietary vs. Apache2 load balancing</td>
</tr>
<tr>
<td>7.21</td>
<td>Test results response time auto-scaling vs. Amazon Elastic Load Balancing</td>
</tr>
<tr>
<td>7.22</td>
<td>Used test results response time auto-scaling vs. Amazon Elastic Load Balancing</td>
</tr>
<tr>
<td>7.23</td>
<td>Test results response time auto-scaling vs. Apache2 load balancing</td>
</tr>
<tr>
<td>7.24</td>
<td>Used test results response time auto-scaling vs. Apache2 load balancing</td>
</tr>
<tr>
<td>7.25</td>
<td>Test results response time Amazon Elastic Load Blancer vs. Apache2 load balancer</td>
</tr>
<tr>
<td>7.26</td>
<td>Used results response time Amazon Elastic Load Blancier vs. Apache2 load balancer</td>
</tr>
<tr>
<td>7.27</td>
<td>Cost indication On-Demand EC2 instance</td>
</tr>
<tr>
<td>7.28</td>
<td>Cost indication Dedicated On-Demand instance</td>
</tr>
<tr>
<td>7.29</td>
<td>Cost indication Dedicated Reserved instance 1 year</td>
</tr>
<tr>
<td>7.30</td>
<td>Cost indication Dedicated Reserved instance 3 years</td>
</tr>
<tr>
<td>7.31</td>
<td>Cost indication Auto Scaling</td>
</tr>
<tr>
<td>7.32</td>
<td>Cost indication Elastic Load Balancing</td>
</tr>
<tr>
<td>7.33</td>
<td>Cost indication custom load balancer</td>
</tr>
<tr>
<td>7.34</td>
<td>Amazon cost comparison</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In this Master thesis the results are presented of research done for the partial fulfillment of the requirements for the degree Master of Science in Computer Science, track Information Architecture. The thesis concerns the implementation of a mobile messaging solution based on a high availability services platform for telecom operators. The proposed mobile messaging solution in this thesis involved initially the design and implementation of a messaging solution based on the IP Multimedia Subsystem architectural framework, which is subsequently deployed in the Cloud. However, the focus of this thesis changed after carried out research, resulting in the design and implementation of a messaging solution based on the Short Message Service (SMS) protocol using the Hypertext Transfer Protocol. Research done for both solutions are discussed and elaborated in this thesis.

This project was carried out in collaboration with T-Sparx Wireless Data Solutions, a company located on the island of Curaçao that offers telecom operators/providers, and their customers messaging, integration and service provisioning solutions. T-Sparx is specialized in the development of mobile messaging solutions for telecom operators or telecom providers and their subscribers. It has developed an application platform named “Jiver” to implement various telecom services for its customers. One key component of the Jiver Platform is the Short Message Service (SMS) Gateway, which offers message routing between mobile devices and from mobile devices to applications using Short Message Service as transport and it operates on a TCP/IP network. T-Sparx is constantly doing research about new technologies to optimize and enhance their services package with innovative services. Hence, a generic architecture has been defined, the so called Messaging Gateway, which defines the same architecture as the SMS Gateway, but with the goal of providing a protocol independent gateway offering the same type of services as the SMS gateway. These architectures are elaborated in Chapter 2.

1.1 Problem description

Since the emergence of mobile communication people have considered the mobile device as a key communication technique between parties that share the same communication technology. Initial mobile communication was used mostly for voice communication, but is shifting from voice to text messaging. In the 1980’s text messaging was introduced through Short Message Service (SMS) technology. With this introduction it became possible for mobile service subscribers to exchange text messages. It also formed the basis for subscriber-to-application and application-to-application communication using text. It has been more than 30 years and SMS is still one of the key mobile text messaging platforms. According to Accenture (2012), the majority of Internet users are using a mobile phone to connect to the Internet, instead of a computer, with smartphones emerging as the most popular Internet access medium. These consumers are shifting from feature phones to smartphones, and are increasingly using their smartphones for Internet access, making mobile Internet one of the most widely used services. Instant messaging services, like WhatsApp Messenger, are becoming a popular (if not the most popular) communication form between mobile subscribers. Hence, the number of messages being sent using Short Message Service by subscribers has been decreasing in recent years. A major shift is taking place in the habits of mobile communication users. The everyday communication habits of these users have changed dramatically, where their demand for Internet services at any time everywhere using mobile devices is increasing considerably. Web 2.0 and social
networking applications, like Facebook, Skype, Google etc. are becoming very popular and also the center of most people’s lives. Telecom consumers want to access these applications and be available online both at home and on their mobile phones everywhere and at any time. T-Sparx has been observing this change in subscriber’s utilization of mobile communication and is interested in extending its current mobile messaging solution to be able to develop and provide additional third generation services, such as conference calls, multimedia conferences, presence, instant messaging, digital television etc.

In addition T-Sparx is experiencing considerable growth in demand for particular services it is offering and is expanding operations to all major telecom operators in the Caribbean. In order to anticipate and prepare for the upcoming growth curve, T-Sparx is investing in research and development of its messaging solution. T-Sparx wants to grow its business to provide mobile messaging solutions to customers all over the Caribbean and is looking for a scalable, reliable, low cost solution, guaranteeing high availability of services with the expected performance requirements to its growing user base.

As mentioned in the chapter introduction T-Sparx has developed an application platform named “Jiver” to make the implementation of numerous telecom services possible. Jiver is a suite of applications developed by T-Sparx to provide services to customers, including services such as message routing, activation and deactivation of services like GPRS, BlackBerry and 3G, billing services like voucher recharge and payment charging. The Jiver platform is elaborated in section 2.2.2. In Figure 1.1 a part of the Jiver platform is illustrated. The area inside the dotted line involves services provided by T-Sparx, and those outside the dotted area include components and/or services owned and/or provided by customers of T-Sparx or other stakeholders.

T-Sparx provides messaging services including the SMS Messaging Gateway and other messaging services such as ProCharge, BillingGateway, ProServ and RadPas. These services are implemented based on the Short Message Peer-to-Peer (SMPP) and Hypertext Transfer Protocol (HTTP) protocols. The Short Message Peer-to-Peer protocol is an open, industry standard message-transfer protocol that enables the transfer of short message data between a Message Center, such as a Short Message Service Centre (SMSC) as can be seen in figure 1.1, and a SMS application system, called the External Short Message Entity (ESME). An SMSC is a network element in the mobile telephone network with the purpose of storing, forwarding, converting and routing SMS messages. ESMEs are non-mobile entities that submit messages to, or receive messages from an SMSC. The Hypertext Transfer Protocol (HTTP) is a request/response protocol, where a client sends a request to a server in the form of a request method, the so-called Uniform Resource Identifier (URI) and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible content over a connection with a server. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta-information, and possible entity-body content. The SMPP, HTTP protocols and the current messaging solution provided by T-Sparx are elaborated, illustrated and evaluated in Part 1 of this thesis. Other components in figure 1.1 include RIM, HLR and the Prepaid platform, which are normally owned by customers of T-Sparx or other stakeholders. The arrows in the figure are bi-directional, meaning that both requests and responses are sent between components.
For the sake of a concrete example, a real-world scenario is generalized as follows. Mobile client II sends an SMS with the text “ATE ON” to a short code number 5555. This SMS message is sent from Mobile client II to the telecom operator’s SMSC. The SMSC forwards the SMS message based on the short code number to T-Sparx’ SMS Messaging Gateway component. If the SMS Messaging Gateway component is down for some reason, or unavailable, the SMSC will store the SMS message and will deliver it to the SMS Messaging Gateway when it is available. The SMS Messaging Gateway receives the SMS message, analyzes its contents, and based on the text message’s content, in this case “ATE ON”, the type of request sent by the mobile client is defined. Based on the type of request the SMS Messaging Gateway may then send a response to the SMSC to be forwarded to Mobile client II, or forward the request to other Jiver components, i.e. BillingGateway or ProCharge, which subsequently forward the request to other components, such as RIM, RadPas, ProServ, HLR, or the Prepaid Platform. These components are described in section 2.2.2.

Due to the considerable growth in demand for particular services at particular peak points, the number of requests for a service is enormous for available proprietary servers to manage. These servers cannot provide the required results to customers in a timely manner, forming queues of requests. This leads to dissatisfied customers waiting for the expected results. The investment in own infrastructure to enable highly-available services to a mass-market may be huge as this includes investing in own infrastructure, investing in maintenance of this infrastructure, leading to slower time to market of new services for small to medium-sized businesses. Currently there exists a lack of an environment that makes the low-cost creation of highly available third generation messaging services possible.

The concept of Next Generation Networks (NGN) has been introduced as a result of the mentioned increasing demand of mobile subscribers for new Internet-mobile-fixed converged services that can be accessed by any end-user device. The growing demand for this type of services triggers telecommunication service providers to start caring about NGN services. The Next Generation Network is characterized by providing innovative telecommunication services to users and making use of multiple broadband transport technologies (Chiang & Chang, 2010), such as cable, DSL, power line, satellite, and wireless. As discussed in Lopez Barbosa (2011) a NGN is a packet-switched network where the packet switching and transport elements are both physically and logically separated from the service/call control intelligence. Transport elements include routers, switches, gateways etc. and service/call intelligence is used to support all types of services over the packet-based transport
network. These types of services include everything from basic voice telephony services to data, video, multimedia, advanced broadband, and management applications. In Next Generation Networks all the telecom infrastructures are IP-based. The International Telecommunication Union - Telecommunication Standardization Sector (ITU-T) defines the Next Generation Network in (ITU, 2004) as a packet-based network able to provide telecommunication services and able to make use of multiple broadband, QoS-enabled transport technologies and in which service-related functions are independent from underlying transport related technologies. It enables unfettered access for users to networks and to competing service providers and/or services of their choice. It supports generalized mobility, which will allow consistent and ubiquitous provision of services to users. The International Telecommunication Union (ITU) is the specialized agency of the United Nations (UN) in the field of telecommunications. The ITU-T is a permanent organ of ITU, responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis (ITU, 2004).

The Third Generation Partnership project (3GPP) standardized the IP (Internet Protocol) Multimedia Subsystem (IMS) architectural framework and described it as the emerging industry standard of choice for the Next Generation Network (Alcatel-Lucent, 2005). The 3GPP is a collaboration agreement between a number of telecommunication standardization bodies, as part of their standardization work for supporting the evolution of the system to a third generation cellular system, and to support the evolution of radio technology, with an emphasis on new higher bandwidth radio access methods (Lopez Barbosa, 2011). According to Camarillo & Garcia-Martin (2008), IMS is considered the key element in the 3G architecture that makes it possible to provide cellular access everywhere and at any time to all the services provided by the Internet. The IP Multimedia Subsystem architectural framework defines the standard interfaces, based on Internet protocols, to be used by service developers, enabling operators to take advantage of a powerful multi-vendor service creation industry, avoiding sticking to a single vendor to obtain new services. The idea of IMS is to offer users Internet services globally at any time using cellular technologies, as discussed in Lopez Barbosa (2011). Using IMS a standardized IP-based architecture is realized which makes convergence possible of fixed and mobile communication devices, multiple network types, and multimedia applications. According to Dialogic Corporation (2009), IMS makes the combination of voice, text, pictures and video in a seamless call session possible. Subscribers are offered this way significant ease-of-use, and service providers can drive branding through a common interface, while considerably reducing operating costs. The IMS architecture consists of three main layers, namely:

1. The “Media/Transport plane” refers to a wide range of different access technologies. Within the IP transport layer users acquire network connectivity through e.g. DSL, Ethernet, Wireless LAN, GPRS or UMTS (Universal Mobile Telecommunication Systems) to acquire network connectivity.
2. After connecting to the IMS network, users can access multimedia services. This is the “Control/Signaling plane”, which consists of the IMS core components, including the Home Subscriber Server (HSS), which is the master user database in the IMS network that supports the IMS network entities that are actually handling the calls/sessions, by providing user- and subscription-related information, and the Call/Session Control function (CSCF), which is a SIP server that is used to process SIP signaling in the IMS. Its main function is providing session control for terminals. SIP requests are processed and routed to the destination in this plane.
3. The “Service/Application plane” consists of different application servers, which provide users with a wide variety of IMS services, such as push-to-talk, presence and instant messaging, and collaborative multimedia conferencing. The real drivers of IMS are combinational services, which combine multiple sessions per call using different types of media, i.e. voice, text, video and pictures.
The separation of the network into these three layers allows true convergence of devices, networks, services and applications. Convergence of devices involves the management and enablement by the network of mobile and fixed devices, such as computers, PDAs, phones and televisions. With convergence of networks is meant that the legacy PSTN, the wireless access network and the broadband networks are treated as a single manageable entity by the corresponding IMS abstraction layer. This way, network services can be offered independently of the access and transport medium. Convergence of network services implies the necessary network functionality that enables subscriber-level applications. Such network services involve tools that make the access possible of user profiles, authentication and billing, location services, and media control services via open, standards-based Application Programming Interfaces (APIs). Lastly, convergence of IMS applications implies that applications can make use of the mentioned common network services, and subscribers can access these applications using different types of networks and devices. Also these IMS applications may reside either within the service provider’s own domain or anywhere in the network. A more comprehensive description of 3GPP and IMS is found in chapter 3.

To allow the low-cost provision of highly available messaging services, Cloud computing is considered as the solution in this project. Cloud computing is the new paradigm for the provision of computing infrastructure, shifting the location of computing infrastructure to the network to reduce the costs associated with the management of hardware and software resources. Buyya et al. (2009) defines a cloud as a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers. Clouds can be considered as a considerable pool of virtualized computing resources that can be reconfigured dynamically to adjust to variable loads, allowing for an optimal resource utilization. As this pool of resources is used according to a pay-per-use model, costs are reduced, as no up-front capital investments in infrastructure are required. A more comprehensive description of Cloud computing can be found in chapter 4.

As IMS is considered the emerging industry standard of choice for the Next Generation Network, this architectural framework and Cloud computing are proposed as the solution for the realization of the environment that makes the low-cost creation of highly available third generation messaging services possible. To make this possible research must be done in both the areas of IP Multimedia Subsystem and Cloud computing.

In a real-world scenario the SMS messaging service currently provided by T-Sparx, using SMPP and HTTP as described in previous paragraphs, is deployed on a proprietary server owned by T-Sparx or by a Telco operator. Mobile subscribers send requests to a short code number that are subsequently received by the server and processed. At certain peak hours the number of requests being received by a server may exceed the processing capacity of the proprietary server. Hence, large queues of requests are formed, consequently overloading the server, causing the server to go down. Typically, a cluster may be created and using a cluster resource manager, such as Corosync Pacemaker, to detect server failures and recovering nodes by starting up a second server. This may result in the loss of a number of requests sent by subscribers between the point of failure and the point of successful recovery of the failed node. This problem of investing in proprietary servers just to ensure that capacity is in place to satisfy peak demands, and the problem of losing a number of requests sent by subscribers when failures occur, can be solved by using Cloud computing technology. Messaging services implemented using HTTP and SMPP may be deployed using virtual machines in the cloud.

Using Cloud computing additional server instances can be automatically launched to ensure the high availability of messaging services without down time. As IMS is considered the emerging industry
standard for the realization of new Internet-mobile-fixed converged services that can be accessed by any end-user device, this technology and Cloud computing are researched to understand to what extent an environment can be realized that makes the creation and providing of highly available third generation services possible. In the following sub-section an elaboration is provided of the focus change of this project.

**Problem focus change**

As described in the previous section, initially both the IMS architectural framework and open source Cloud solutions would be researched to realize an environment that makes the creation and providing of highly available third generation services possible. This entails understanding the details of the IP Multimedia Subsystem architectural framework to design and implement a basic messaging solution, simulating this way some components of the Jiver platform, and Cloud computing, subsequently deploying these services in the Cloud. The objective was to use open source solutions to form a basic IMS architecture, and a couple of servers available at hand, using an open source private Cloud solution, this way forming an IMS messaging solution in the Cloud. Having this messaging solution in the Cloud in place performance tests would be carried out to test the responsiveness of this solution, using a new technology, i.e. IMS.

Based on time constraints and the complexity concerned with the implementation of a messaging solution using the IP Multimedia Subsystem architectural framework in the Cloud using open source solutions, the decision was made to focus on the technical implementation of a Cloud solution. During a long period experiments were conducted with different open source technologies to obtain the aforementioned messaging solution, but as time passed by and no successfully functioning messaging solutions in an open source Cloud were obtained, the choice was made to use existing proprietary solutions for both the messaging solution and the cloud solution. This way placing more emphasis on the extent to which the availability of messaging services and the response time of requests for messaging services improves using Cloud computing. Chapter 5 provides a brief description of the above-mentioned experiments using open source solutions. Hence, for the practical part of this project the messaging solution used is based on the messaging protocols Short Message Service and Hypertext Transfer Protocol, and is in turn deployed in the Cloud using the Amazon AWS cloud-computing platform.

In the following section the objective of this research is presented, including the problem statement, sub research questions and hypotheses. These are followed by a brief overview of the previously formulated problem statement with corresponding sub research questions for the open source IMS messaging service.

### 1.2 Objective

Based on the change in technologies used for both the messaging and the cloud solution, as described in the previous sub-section, the main objective of the research has been modified and adapted. The main objective of the research is to investigate the availability of mobile messaging services and the costs involved when deployed in the following three scenarios:

- on a single/dedicated server,
- in the Cloud and
- using load balancing

Additionally, the average response time required for processing a request is evaluated in each scenario and compared between all three scenarios.
The following will therefore be determined by this thesis:

To what extent will implementing messaging services on the Amazon Cloud improve the availability of mobile messaging services provided by telecom operators and reduce infrastructure costs?

This main research question has been divided in the following sub research questions:

- **RQ1** How are mobile messaging services implemented using the HTTP protocol?
- **RQ2** How can mobile messaging services be deployed in the Amazon Cloud?
- **RQ3** Which issues exist and should be tackled when providing mobile messaging services in the Amazon cloud?
- **RQ4** Is the availability of mobile messaging services preserved using a dedicated server, when the capacity of the infrastructure is reached?
- **RQ5** Is the availability of mobile messaging services preserved using auto-scaling technology, when the capacity of the infrastructure is reached?
- **RQ6** Is the availability of mobile messaging services preserved using load balancing, when the capacity of the infrastructure is reached?
- **RQ7** How is the response time for a subscriber’s request for service influenced using auto-scaling technology compared to using a dedicated server?
- **RQ8** How is the response time for a subscriber’s request for service influenced using load balancing compared to using a dedicated server?
- **RQ9** How is the response time for a subscriber’s request for service influenced using auto-scaling technology compared to using load balancing?
- **RQ10** What are the costs involved in deploying messaging services in the Amazon Cloud on a dedicated server?
- **RQ11** What are the costs involved in deploying messaging services in the Amazon Cloud using Auto Scaling?
- **RQ12** What are the costs involved in deploying messaging services in the Amazon Cloud using load-balancing technology?

The availability of mobile messaging services refers to the proportion of time that the services are in a functioning condition, i.e. accessible. The Service Availability Forum (SAF) (2013) defines two main aspects of service availability: high availability and service continuity. With availability being defined as the probability of a service being up and running at any instant without a breakdown and the corresponding downtime, high availability refers to an availability of at least 99.999%. According to Rossebø et al. (2006), service availability encompasses both exclusivity, i.e. the property of being able to ensure access to authorized users only, and accessibility, i.e. the property of being at hand and useable when needed. Service continuity is the ability of a service to maintain customer sessions for uninterrupted services, i.e. even in case of failure of a given component in the service infrastructure and subsequent take-over by a redundant component. To test the high availability of services particular, yet to be investigated, testing models or methods are used, e.g. the time it takes for people to receive the response after their
request has been processed is used to make a comparison between the current situation when proprietary servers are used, and the situation when cloud computing technology is used. This is researched to determine to what extent cloud computing technology improves the availability of mobile messaging services.

- The **auto scaling technology** refers to the automatic application of the 'Scalability of infrastructure' cloud characteristic, described in Lopez Barbosa (2011), where new nodes can be added or dropped from the network with limited modifications to infrastructure set up and software. Auto scaling allows users to automatically scale cloud services up or down, such as virtual machines (VM) and server capacities, using self-defined rules.

- The **response time** of a subscriber’s request for service refers to the total amount of time it takes to respond to a request of a subscriber for service, i.e. in this context the total amount of time it takes to receive the entire response. That service can be any telecommunications-related service provided by a telecom operator. The response time is based on both the service time and the wait time, and may also be based on the transmission time, i.e. response time is the sum of service time and wait time. The service time is the time it takes to process the request, in other words do the work that was requested. The wait time is the time the request had to wait in a queue before being serviced and it varies from zero, when no waiting is required, to a large multiple of the service time, as many requests are already in the queue and have to be serviced first. Transmission time can also form part of the response time when it is required for the request and the resulting response to travel over a network and it the transmission time is significant in the calculation or measurement of the response time.

**Hypotheses**

The following hypotheses have been defined, which this thesis will support or reject:

- **H1** The availability of services at event-based peaks of traffic is not preserved using a dedicated server, when the capacity of the infrastructure is reached.

- **H2** The availability of services at event-based peaks of traffic is preserved using auto-scaling technology, when the capacity of the infrastructure is reached.

- **H3** The availability of services at event-based peaks of traffic is preserved using load-balancing technology, when the capacity of the infrastructure is reached.

- **H4** The response time for a subscriber’s request for service improves when using auto-scaling technology instead of a dedicated server.

- **H5** The response time for a subscriber’s request for service improves when using load balancing instead of a dedicated server.

- **H6** The response time for a subscriber’s request for service improves when using load balancing instead of auto-scaling technology.

To answer the problem statement given above, the listed sub-questions and to support or reject the stated hypotheses, this thesis will focus on the comparison of the availability of one single messaging service. This particular service concerns a central messaging gateway, which is a piece of software code that enables the receiving of a text message, and forwarding of this message to other mobile messaging services, based on the message’s content. This way a text messaging traffic is generated
making it possible to measure the availability of the messaging service, making a distinction between the following three scenarios:

1. SMS messaging solution deployed on a proprietary server,
2. SMS messaging solution deployed in the Amazon AWS cloud-computing platform,
3. SMS messaging solution deployed using Amazon’s Elastic Load Balancing feature.

The SMS messaging service involves a simplified version of messaging service currently used at T-Sparx, implemented using the HTTP protocol. Comparing the results obtained in each of these scenarios, the extent to which the availability of mobile messaging services improves using the Amazon Web Services cloud-computing platform is determined.

The following example will clarify the idea:

- **Business case:** Suppose that the telecom service provider has implemented a messaging service, which is deployed in the Cloud for scalability and high availability reasons. The telecom service provider’s clients are telecom operators that access its services and in turn provide these services to their own subscribers. Subscribers make a request for this particular service and the operator forwards this request to the telecom service provider to access this particular service. The telecom service provider processes the request and sends a response to the operator, which notifies the subscriber about the status of the service. When two different operators interact with each other, making use of the telecom service provider’s service, all requests are received and processed by the telecom service provider as a central gateway, which sends subsequently a response to the destination.

As telecom service providers’ client base scales dynamically, i.e. the number of operators accessing the messaging service changes dramatically, the number of subscribers that actually use the service provided by the telecom service provider may be enormous at certain peak hours. To continually guarantee high availability of services Cloud computing technology is used.

To make the above possible and prove that its implementation does actually function, the following topics are researched:

- Design and implementation of an SMS messaging solution using the Hypertext Transfer Protocol (HTTP)
- The Amazon AWS cloud-computing platform, including its features and capabilities,
- Security issues concerning deployment of services in the Amazon AWS cloud-computing platform, this way possibly limiting the number of services that can be deployed in a real world scenario.

**Previous problem statement**

The following previously formulated problem statement and sub research questions were used as the basis for the formulation of the problem statement and sub research questions in the previous section. Research was done to determine what new services can be provided to telecom operators using IMS, compared with the currently used cellular technology, i.e. to what extent will the diversity of services provided to telecom operators improve by implementing IMS. Also, initially the intention was to do research to determine the extent to which the availability of services will improve when using Cloud computing instead of proprietary servers. The main objective of the research was to investigate the possibility to provide highly available third generation mobile messaging services to telecom operators by designing and implementing these services according to the IP Multimedia Subsystem architectural framework, subsequently deploying these services in the Cloud.
The following would therefore be determined:

To what extent will implementing IP Multimedia Subsystem messaging services in the Cloud improve diversity and availability of mobile messaging services provided by telecom operators?

This main research question was divided in the following sub-questions:

- How can mobile messaging services be implemented using the IMS framework?
- Which additional services can be provided by telecom operators using the IMS framework?
- How can a low cost IMS messaging solution in the Cloud be achieved?
- How can IMS messaging services be deployed in the cloud?
- Which issues exist and should be tackled when providing mobile messaging services in the cloud?
- How is the availability of mobile messaging services influenced using cloud computing technology?

- The diversity of services offered by telecom operators involves the types of mobile messaging services each operator provides. Each operator is located in a certain area in a certain country and offers particular services based on choices made because of different factors, including e.g. the costs for the development and launching of the services, licenses that are required, the infrastructure required for new services, the local expertise for dealing with the development and support for services etc. Offering IMS messaging services may make it much easier for operators to provide a greater number of mobile messaging services, without having to concern with additional costs, expertise etc. This will be researched to determine to what extent IMS improves the diversity of mobile messaging services.

- The availability of mobile messaging services refers to the proportion of time that the services are in a functioning condition, i.e. accessible. The Service Availability Forum (SAF) (2013) defines two main aspects of service availability: high availability and service continuity. With availability being defined as the probability of a service being up and running at any instant without a breakdown and the corresponding downtime, high availability refers to an availability of at least 99.999%. According to Rossebø et al. (2006), service availability encompasses both exclusivity, i.e. the property of being able to ensure access to authorized users only, and accessibility, i.e. the property of being at hand and useable when needed. Service continuity is the ability of a service to maintain customer sessions for uninterrupted services, i.e. even in case of failure of a given component in the service infrastructure and subsequent take-over by a redundant component. To test the high availability of services particular, yet to be investigated, testing models or methods are used, e.g. the time it takes for people to receive the response after their request has been processed is used to make a comparison between the current situation when proprietary servers are used, and the situation when cloud computing technology is used. This is researched to determine to what extent cloud computing technology improves the availability of mobile messaging services.

To answer this problem statement, this thesis would provide a theoretical overview of the services that can be provided using the IMS architectural framework, compared to using SMS technology, and focus on the comparison of the availability of the same single messaging service, making a distinction between the following two scenarios:
1. IMS Messaging solution deployed on a proprietary server, and
2. IMS Messaging solution deployed in the Cloud.

As telecom service providers’ client base scales dynamically, i.e. the number of operators accessing the messaging service changes dramatically, the number of subscribers that actually use the service provided by the telecom service provider may be enormous at certain peak hours. To continually guarantee high availability of services Cloud computing technology would be used.

To make the above possible and prove that its implementation does actually function, the following topics were researched:

- design and implementation of an IMS infrastructure:
  - required IMS components for T-Sparx to make the provisioning of the IMS gateway service possible,
  - interfaces needed to enable these IMS components to interact adequately internally, and also externally with other IMS networks,
- implementation of the particular text messaging service using the IMS architectural framework,
- cloud computing (an analysis of the different relevant open source Cloud computing providers/solutions, including cloud interfaces, hypervisors and cloud platforms, etc., the deployment of software components in the cloud and virtualization)
- security issues concerning deployment of services in the Cloud, this way possibly limiting the number of services that can be deployed in a real world scenario.

Next the assumptions made during execution of the project are defined.

1.3 Assumptions

Research is done on the topics mentioned above to make the design and implementation of a scalable, reliable, low cost, highly available messaging solution possible, which results in a prototype system that serves as a model for telecom service providers in general. The following assumptions are made during execution of the project.

- The number of telecom operators using the mobile messaging services is variable.
- The telecom operators have a dynamically changing number of subscribers. The number of subscribers using the messaging gateway may change dynamically as the number of operators using provided service(s) changes.
- In this project T-Sparx is the service provider owning the required platform to enable the providing of messaging services based on the HTTP protocol.
- T-Sparx offers messaging services in the Cloud, making these services highly available to operators.
- For the thesis project the required infrastructure to enable the single SMS messaging service based on the HTTP protocol is implemented (with all necessary components) to test the prototype system.
- Operators include clients of T-Sparx, such as Digicel and UTS in Curaçao, and Telesur in Surinam
- The complexity concerned with the implementation of the single HTTP-based messaging solution using the HTTP protocol is undefined for the author and will be analyzed in this thesis, implying that the feasibility of such service is undefined within the time constraints of this project.
- The complexity concerned with the deployment of the messaging service in the Cloud is undefined and will be analyzed in this thesis, implying that the feasibility of such service is undefined within the time constraints of this project.

### 1.4 Research strategy

In order to solve the above-presented problem, several topics need to be thoroughly researched, including the main topics of SMS, HTTP and Cloud computing. As not much theory is available about the deployment of mobile messaging services in the cloud, this thesis will research these topics extensively.

For the comparison of the three listed scenarios in section 1.2 an abstract model of the currently provided messaging service is used, which involves an architecture consisting of main components required to make text messaging possible. This model is used in each scenario with the purpose to obtain a fair and consistent evaluation.

![Figure 1.2: Abstract model of messaging solution](image)

Figure 1.2 consists of the following elements:

- **Mobile Clients**, which are the devices sending requests and receiving responses. The request that is sent involves one for a particular service to the Messaging Gateway, which in turn forwards and/or processes the request. This includes forwarding the request based on its content to the appropriate component to further handle the request, or process the request based on its content. Afterwards a response is returned to the mobile client.

- **Messaging Gateway**, which offers message routing between mobile devices and from mobile devices to applications. The current service is based on the SMPP protocol using Short Message Service (SMS) as transport and operating on a TCP/IP network.

- **Request Manager**, which involves applications providing a particular service. This means that the application receives a request, which has been forwarded by the messaging gateway or another application, and processes this request based on its contents, subsequently sending a response to the sender of the request. This component may also just forward the request to another request manager for processing.
As can be seen in this figure, the arrows are two-sided, meaning that both requests and responses are sent between the two sides. Mobile clients send direct requests to the messaging gateway, which in this model is the only messaging service component that can be accessed by mobile clients. This messaging gateway component is turn forwards to a variable number of request managers, depending on the messaging service configuration. Each of these request managers can forward requests to other components, or immediately process the request based on its contents. For this project a particular messaging service configuration is used and described in Chapter 2.

This project is divided into the following phases, which in turn form the main parts of this thesis.

The first phase in this thesis is a clear theoretical overview of the topics of SMS, HTTP, IMS, and Cloud computing, in order to provide a clear understanding to the reader of the technologies researched and used in this project, and also insight into the current messaging services provided by T-Sparx.

The second phase in this project involves the design and technical implementation of the messaging service in the Cloud. Firstly research done for the topic of IMS is elaborated, including the design and implementation of the IMS messaging service and the IMS infrastructure. This IMS messaging service involves an IMS environment consisting of main IMS components that are required to make text messaging possible. The open source tools evaluated and used for designing and implementing the IMS infrastructure, the IMS application server and the IMS messaging service are described. Next, a description is provided of the design and implementation of the Cloud infrastructure using open source tools. Subsequently, a description is given of the design and implementation of an abstract model of a basic messaging service currently provided by the telecom service provider T-Sparx. Using the abstract architecture described above and illustrated in figure 1.2, a text messaging service is designed and implemented based on the HTTP protocol enabling the transfer of SMS messages between entities, followed by its deployment in the Cloud using Amazon AWS. Generating high traffic loads of SMS messages, both the availability and performance of the HTTP-based messaging gateway deployed on a single server are tested and evaluated. Afterwards, the messaging service is deployed in the Amazon Cloud with the purpose to test both the system’s availability and performance when generating high traffic loads of SMS messages. Also, load-balancing technology is used to evaluate both the system’s availability and performance when high traffic loads of SMS messages are generated. The method and tools used for availability and performance evaluation are investigated, using available literature, and described. Finally in this phase the actual evaluation of the availability and performance results obtained for each scenario is described. The results of these tests contribute in determining if the design and implementation of a messaging service, deployed in the Cloud, or using load balancing, lead to high availability of services, and also to what extent the messaging service’s performance improves.

The third phase involves a discussion of the drawn conclusions, recommendations for the design and implementation of a highly available messaging service, limitations encountered during execution of the project and future work.

1.5 Deliverables

This project was originally proposed by the company T-Sparx as the need arose for upgrading their current messaging solutions to be highly available, this way to be able to manage a large number of customers. The thesis will tackle the problem from a general perspective delivering this way a reference architecture that can provide insight to and be used by any small- to medium-sized telecom service providers interested in the transition from using their current infrastructure to using messaging solutions deployed in the Cloud. By designing and implementing a prototype, the proposed architecture design can be verified, i.e. it delivers the expected results.
The deliverables of this research project include:

- A **reference architecture** for providing high availability of messaging services to telecom operators, with the architecture acting as a central messaging gateway forwarding responses to requests, from and to operators. In this context ‘reference architecture’ refers to a suggestion of an architecture for a messaging solution that provides the key feature of improving diversity and availability of mobile messaging services provided by telecom operators, using particular technologies.

- A **prototype** of the proposed architecture to show that the proposed architecture design consisting of its components using the selected technologies does deliver the expected results, i.e. high availability of mobile messaging services.

- A **thesis** including the result of the proposed architecture and system, with corresponding explanation of used techniques and technologies.

- **Recommendations** for the most feasible and qualitatively high technical realization.

## 1.6 Relevancy of research

Because of the intense competition between telecom operators and telecom service providers in providing innovative communications services, it is essential for these companies to position themselves appropriately in the market to take advantage of their core competencies. As more customers are using mobile phones, including the popular smartphones, the demand for services provided by telecom service providers is increasing dramatically. It is essential for telecom operators and telecom service providers to continue to guarantee a high quality of their services to their customers. This fast growth of the customer base makes the high availability of services a necessity. In the last few years Cloud computing has become very popular and is a great alternative for companies that deliver services. Hence, interest in cloud computing is growing and telecom service providers are interested in deploying their services in the cloud ensuring scalability to respond quickly to the dynamically changing demand and ensure high availability of their services. Doing research on the topic of Cloud computing gives more insight into the benefit, realization-feasibility and drawbacks that may be involved in the design and implementation of a mobile messaging solution in the Cloud. Doing this research adds value to science in the sense that a relatively new method of implementing mobile messaging services is researched and documented, providing a description for other companies in the same sector interested in the transition to providing mobile messaging services in the Cloud. Also a comparison is provided of the availability of services when deployed using a single server, using the Cloud and using the load balancing technology.

## 1.7 Scope of research

This thesis includes research about the topics Short Message Service (SMS) using the Hypertext Transfer Protocol (HTTP), Cloud computing and load balancing. The topic of IP Multimedia Subsystem (IMS) architectural framework is described and an evaluation is provided of why the messaging solution based on this framework was not implemented. The open source solution for the implementation of a private cloud is described and an evaluation is provided of why this private cloud was not implemented.

SMS, HTTP and Cloud computing are described on a general level in order to provide an understanding to the reader of their purposes and similarities. These are all very broad topics; hence the focus of this project is on the design of an architecture and implementation of a solution to enable the providing of a **highly available mobile text messaging service** using the HTTP protocol and
Cloud computing. This document offers an overview of the architecture and configuration of an SMS messaging solution based on the HTTP protocol, deployed on a proprietary server and in the Cloud.

Research about SMPP and HTTP includes an overview of what these protocols are, how and where these can be used, its key characteristics and what kind of services can be provided using these protocols. Also current HTTP solutions are researched for the design and implementation of a text messaging solution for performance evaluation purposes.

Research about Cloud computing and load balancing includes determining the configuration needed to properly deploy the implemented messaging solution in the Amazon Cloud.

1.8 Thesis outline

This thesis consists of the following three parts:

Part I introduces the technologies Short Message Service, Hypertext Transfer Protocol, IP Multimedia Subsystem and Cloud Computing. With a clear definition of what the SMS technology is with related protocols, such as SMPP, it provides insight of the messaging solution currently used by T-Sparx in Chapter 2. Also problems faced with the current messaging solution architecture and infrastructure architecture, are given followed by possible solutions and challenges. Chapter 3 concerns the previously carried out research topic of the IP Multimedia Subsystem architectural framework. It provides an introduction of IMS, followed by IMS commercial and open source solutions, and a description of IMS challenges. Cloud computing is introduced with its security challenges, followed by commercial and open source solutions in Chapter 4.

Part II presents the technical implementation of the messaging solution deployed in the Cloud. In Chapter 5 the design and implementation of the previously researched IMS messaging solution are elaborated, followed by the design and implementation of the open source cloud infrastructure. The design and implementation of the HTTP based messaging solution, followed by the design and implementation of the cloud infrastructure using Amazon AWS are described in Chapter 6. Also the design and deployment of the HTTP messaging solution in the Cloud are presented in Chapter 6. Hereafter, Chapter 7 presents the performance tests that were carried out, with corresponding result evaluations, of the HTTP-based messaging solution:
- deployed on a single server,
- deployed in the Cloud using Amazon Auto Scaling, and
- deployed in the Cloud using Amazon Elastic Load Balancing.

Part III presents the conclusions in Chapter 8, while in Chapter 9 Recommendations and future work can be found.
PART I

Introduction to messaging technologies and cloud computing
Chapter 2
Short Message Service messaging solution

This chapter starts with the introduction of both the Short Message Service protocol and the Hypertext Transfer Protocol. Subsequently T-Sparx’ current messaging solution architecture and infrastructure architecture are described and illustrated. Next, problems faced with the current messaging infrastructure architecture are elaborated, which is followed with possible solutions for these problems, and corresponding challenges. The next section provides a description and overview of the Short Message Service and HTTP protocols.

2.1 Introduction to Short Message Service and Hypertext Transfer Protocol

Short Message Service (SMS) since its first appearance in 1992 in Europe, has become an extension of people’s lives and is still considered the de-facto communication tool for many (Samanta et al., 2012). With more than 2.4 billion active users in the world, which is approximately 74% of all mobile phone subscribers, SMS has become the most widely used data application in the world. Formerly this technology was included in the Global System for Mobile (GSM) standards. Later it was included in other wireless technologies like Code Division Multiple Access (CDMA) and Time Division Multiple Access (TDMA). As GSM is the standard used in the region this project was carried out, all protocols explained in this document refer to the GSM standard. SMS is still one of the major revenue generators for wireless carriers, and is the text communication service component of mobile communication systems, using standardized communication protocols allowing the exchange of short messages between mobile phone devices.

SMS is a bi-directional service for sending a short alphanumeric message in a store-and-forward process. SMS can be used both ‘point-to-point’ as well as in cell-broadcast mode.

An SMS message, as specified by the ETSI organization, has a maximum length of 160 character using 7-bits Latin character encoding, and a maximum length of 70 characters using 16-bit Unicode UCS2 character encoding. Still many messages that are received on mobile devices are generated from computers running SMS-based applications connected to the Global System for Mobile (GSM) Communications network (3GPP TS 03.40, 2010). These messages can be generated using e.g. the Computer Interface to Message Distribution (CIMD) protocol or the Short Message Peer-to-Peer (SMPP) protocol over TCP/IP layer. This latter protocol is described next, followed by a description of a more secure SMPP implementation, this way providing a better understanding of the currently used messaging solution.

2.1.1 Short Message Peer-to-Peer Protocol

According to SMPP Developers Forum (1999), the SMPP protocol is an open, industry standard protocol that allows the exchange of or transfer of short message data (SMS messages) between a Message Center, such as a Short Message Service Center (SMSC), a GSM Unstructured Supplementary Services Data (USSD) server or other type of Message Center, and an SMS application system, called the External Short Message Entity (ESME). An SMSC is a network element in the mobile telephone network with the purpose of storing, forwarding, converting and routing SMS messages. ESMEs are
non-mobile entities that submit messages to, or receive messages from an SMSC. Subscribers that are connected to an SMS-capable cellular network are able to receive short messages on their mobile station or user equipment (UE) from ESMEs.

Some examples of ESMEs include:
- Voicemail alerts originating from a VPS (Voice Processing System), indicating voice messages at a customer’s mailbox.
- Information services, such as an application that enables mobile subscribers to query currency rates or share-price information from a database or the WWW and have it displayed as a short message on the handsets.
- Calls directly dialed or diverted to a message-bureau operator, who forwards the message to the SMSC, for onward delivery to a subscriber’s handset.

SMPP supports, other than the GSM digital cellular network technology, also UMTS, IS-95 (CDMA), CDMA2000, ANSI-136 (TDMA) and iDEN, and is the most commonly used protocol for the exchange of short messages.

Using the SMPP protocol an ESME can initiate an application layer connection with an SMSC over a TCP/IP or X.25 (a packet-based protocol that is used mainly in Europe) network connection, and can then send and receive short messages to and from the SMSC respectively. The short messages exchanged between an SMSC and an ESME are request and response protocol data units (PDUs). Every SMPP operation consists of a request PDU and associated response PDU (with the exception of the alert_notification request PDU for which there is no response).

The SMPP protocol defines:
- the operations and associated PDUs for the exchange of short messages between an ESME and an SMSC
- the data that an ESME application must exchange with an SMSC during SMPP operations.

The SMPP protocol is an application layer protocol and is not intended to offer transport functionality. The underlying transport interface between an SMSC and ESME may be based on a TCP/IP or X.25 connection, and it is assumed that the underlying connection between the SMSC and ESME will provide reliable data transfer including packet encoding, windowing, flow control and error handling.

As can be seen in Figure 2.1 there exists three groups of transactions for the exchange of messages between an SMSC and an ESME, namely:

I. messages sent from the ESME (Transmitter) to the SMSC
II. messages sent from the SMSC to the ESME (Receiver)
III. messages sent from the ESME (Transceiver) to the SMSC and messages sent from the SMSC to the ESME (Transceiver)

In group (I) an ESME, which sends short messages to an SMSC must be connected to this SMSC as an ESME Transmitter or an EMSE Transceiver. In group (II) the SMSC delivers short messages to an ESME, where the ESME is connected to the SMSC as an ESME Receiver or ESME Transceiver. In Group (III) messages are exchanged in both directions, where the ESME is connected to the SMSC as an ESME Transceiver. A typical case is a two-way message exchange between a mobile station and an ESME, such as a WAP Proxy Server, where the mobile subscriber initiates an information request to the WAP Proxy Server and the information response is returned via the SMSC to the mobile station.

A technical description of these transactions can be found in SMPP Developers Forum (1999) as this description is not relevant in this thesis.
According to Samanta et al. (2012), messages travel from an ESME to an SMSC in plain text, with no encryption standard, which implies that this information can be easily intercepted and read. The SMPP protocol can be secured by using the Transfer Layer Security (TLS), which was previously known as Secure Sockets Layer (SSL). TLS is a cryptographic protocol, which was designed to provide security for communication over the Internet (RFC 5246, 2008), using public-and-private key encryption system, which also includes the use of a digital certificate. Using cryptography TLS provides endpoint authentication and communication confidentiality over the Internet.

To provide customers (content providers) with a simpler implementation of their SMS solution, T-Sparx makes it possible for them to create simple HTTP requests that include all necessary information to be processed by Jiver. Customers often do not have the required in-house knowledge or resources to be able to create SMPP requests. For this reason creating simpler HTTP requests is a nice solution for these customers. T-Sparx ensures that these HTTP requests are properly received by Jiver and translated to SMPP protocol requests, which are subsequently forwarded to the required Jiver component to be processed. In the following section the HTTP protocol is described, including all necessary information to understand how this protocol works.

2.1.2 Hypertext Transfer Protocol

The Hypertext Transfer Protocol (HTTP) is a request/response protocol (Fielding, R. et al., 1999), where a client sends a request to a server in the form of a request method, the so-called Uniform Resource Identifier (URI) (Berners-Lee, T. et al., 1998) and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible content over a connection with a server. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity metadata, and possible entity-body content. HTTP allows systems to be built independently of the data being transferred, because of its typing and negotiation feature. Most HTTP communication is initiated by a user agent and consists of a request to be applied to a resource on some origin server.

Figure 2.1: SMPP interface between SMSC and ESME
HTTP has been in use by the World-Wide Web global information initiative since 1990. The first version of HTTP, named the HTTP/0.9, was a simple protocol for raw data transfer across the Internet, and was really primitive and never specified in any standard. HTTP/1.0, as defined by RFC 1945 (Berners-Lee, T. et al., 1996), improved the protocol by allowing messages to be in the format of MIME-like messages, containing meta-information about the data transferred and modifiers on the request/response semantics. HTTP/1.1 extends and improves HTTP/1.0 by including some extensions for authoring documents online via HTTP and an extra feature allowing the connection to be kept open after a request, so that this connection does not have to be reestablished for a next request. HTTP communication takes place over a TCP/IP connection (but can implemented on top of any other protocol on the Internet or other networks), which means that the bits and bytes that move back and forth across the network is the responsibility of the TCP/IP. Thus, HTTP defines what the browser and web server say to each other, not how they communicate.

There are several HTTP request types including GET, POST, HEAD defined in HTTP/1.0 (Berners-Lee, T. et al., 1996) and the PUT, DELETE, HEAD, TRACE, OPTIONS added in the HTTP/1.1 specification (Fielding, R. et al., 1999).

The HTTP GET request is used to process requests sent to the messaging gateway as defined in Chapter 6, and is the one covered as an illustration in this thesis. A HTTP GET request is used if the interaction is a kind of question, i.e. it is a safe operation such as a query, read operation or a lookup (Jacobs, I., 2004). All information necessary for the web interaction forms part of the URI, while with a POST request some information necessary for the interaction that affects the state of the resource on the server form part of the protocol headers, and not in the URI. With POST URI addressability is lost which means that only using a URI for a server application to carry out a type of interaction is not anymore the case. Some metadata not required for the interaction may not be part of the URI. The way the GET URI is used in the technical implementation of the messaging gateway is described in Chapter 6.

When sensitive data, such as passwords, credit card numbers or bank account numbers, is involved in in web interactions, this information can be protected using the Secure Sockets Layer (SSL) protocol. Some costs of using SSL include the time required to establish an SSL connection and the memory required on the server to create a new copy of encrypted data for each request. A POST request can also be used when the use of protocols, such as SSL are inappropriate to carry credentials for example.

### 2.2 Current messaging solution architecture and infrastructure architecture

This section starts with an overview of the GSM standard, including the GPRS/EDGE overlay architecture, followed by a brief overview and description of the Jiver platform consisting of the core services provided by T-Sparx, and finally an illustration and elaboration are given of the architecture and infrastructure architecture of the SMS gateway (SMS messaging solution).

#### 2.2.1 The GSM standard

For the sake of clarity, the meaning of the technical terms used in this section for several messaging components are given by describing the GSM network structure with the GPRS overlay architecture, consisting of its core elements. The GSM network structure is given in Figure 2.2 GSM architecture with essential component.
Global system for mobile communication (GSM) can be defined as a standard that has been globally accepted as the standard for digital cellular communication (TelecomSpace, 2014). GSM is the world’s main 2G standard, and it is the name of a standardization group established in 1982 to create a common European mobile telephone standard that would formulate specifications for a pan-European mobile cellular radio system operating at 900 MHz. GSM is one of the four main standards for 2G networks, including Time Division Multiple Access (TDMA), Code Division Multiple Access (CDMA) and Personal Digital Cellular (PDC, exclusively used in Japan) (Selian, 2006). GSM operates on the 900 MHz, 1800 MHz and 1900 MHz frequency bands. GSM’s network is considered to be secure because both speech and data are encrypted to prevent eavesdropping. Also GSM subscribers are identified by their Subscriber Identity Module (SIM) card, which holds their identity number and authentication key and algorithm. The GSM system architecture with essential components can be seen in Figure 2.2 and is briefly explained below.

![Figure 2.2: GSM architecture with essential components (Bettstetter et al., 1999)](image)

As can be seen in the figure a mobile device (mobile station MS in the figure). A base transceiver station (BTS) covers a radio area, which forms a cell in the GSM network. One base station controller (BSC) controls multiple BTSs together. Multiple BTSs together form a cell group, so a BSC is assigned a cell group. The BTS and the BSC components in the GSM network together form the base station subsystem (BSS). The traffic of all mobile stations combined in a cell is routed through a switch, which is called the mobile switching center (MSC). Calls or connections originating from or terminating in the fixed network are managed by a dedicated gateway mobile switching center (GMSC). GSM networks consist of at least one administrative region that is assigned to a MSC. Each administrative region consists of at least one location area (LA), which consists of several cell groups. As already given above, each cell group is assigned to a BSC. A GSM network includes several databases for call control and network management, i.e. the home location register (HLR), the visited location register (VLR), the authentication center (AUC), and the equipment identity register (EIR). The HLR stores permanent data, such as the user’s profile, and temporary data, such as the user’s current location, for all users registered with a network operator. The VLR, which is responsible for the current location of a subscriber, is responsible for a group of LAs and it stores data of users currently in its area of responsibility. This data includes parts of the permanent user data transmitted from the HLR to the VLR for faster access. The VLR may also assign and store local data, such temporary identification of users. The AUC generates and stores security-related data such as keys used for authentication and encryption, and the EIR stores equipment data.

The maximum circuit-switched data speed within the existing GSM encoding techniques is 9.6 Kbit/s or with improved encoding, up to 14.4 Kbit/s. Enhancements upon 2G GSM systems include the High
The General Packet Radio Service (GPRS) is a bearer service for GSM, which significantly improves and simplifies wireless access to packet data networks, such as the Internet. It applies a packet radio principle to transfer user data packets in an efficient way between mobile stations and external packet data networks (Bettstetter et al., 1999). GPRS can be added on top of existing GSM systems as an overlay technology, i.e. GSM still handles voice, and mobile devices support both voice and data via the overlay functions (Selian, 2006). GPRS is packet-switched and enhances circuit-switched data and short message services (SMS), and serves as an enabler of mobile wireless data services. As GPRS is packet-based, it promises data rates from 56 up to 114 Kbit/s, as well as constant connection to the Internet for mobile device users. “The circuit-switched communications network is a network in which a dedicated connection, or circuit, is established for the duration of a transmission, i.e. a fixed transmission bandwidth is pre-allocated for an entire call or session. The most universal circuit-switched network is the telephone system (PSTN). Since interactive data traffic occurs in short bursts, 90 percent or more of the pre-allocated bandwidth is wasted. Thus, as digital electronics became inexpensive enough, it became dramatically more cost-effective to completely redesign communications networks, introducing the concept of packet switching. The exclusive access to radio resources is not necessary for data applications with the use of packet switched techniques. In the packet-switched network the bandwidth is dynamically allocated, without attempting to pre-allocate bandwidth over the whole source-to-destination path. This type of allocation is superior to pre-allocation systems in connect time, reliability, economy and flexibility. Packet switching divides the input flow of information into small chunks of data, called ‘packets’, ‘segments’ or ‘datagrams’, which move through the network in a manner similar as the manual handling of mail. These packets contain, along with the data, information about the IP address of the source and the destination nodes, sequence numbers and some other control information. The packets are sent towards the destination irrespective of each other and they arrive at random intervals, forming queues at intermediate nodes, waiting for an outgoing channel to become available. Each packet finds its own route to the destination without any predetermined path, using the information it carries such as the source and destination IP addresses. Packet systems offer substantial economic and performance advantages over conventional systems. This has resulted in rapid worldwide acceptance of packet switching for low-speed interactive data communications networks, both public and private” (Lopez Barbosa, 2011). The GRPS system architecture with essential components is shown in the following Figure 2.3: GPRS system architecture.

Figure 2.3: GPRS system architecture (Bettstetter et al., 1999)
EDGE is an enhancement to the 2G GSM system that uses the 200 kHz GSM radio carrier bandwidth, allowing data transmission speeds of up to 384 Kbit/s to be achieved. It is a radio based high-speed mobile data standard allowing a more cost-efficient way of migrating to full-blown 3G services. EDGE’s core network still uses GPRS/GSM and it concentrates on improving the capacity and efficiency over the air interface by introducing a more advanced coding scheme allowing more data to be transmitted (Selian, 2006).

An important network node of the GPRS/EDGE network is the GPRS Support Node (GSN). This node makes the integration of GPRS into the existing GSM architecture possible (ETSI TC-SMG, 1996). A GSN is responsible for the delivery and routing of data packets between the mobile devices and the external packet data networks (Bettstetter et al., 1999). Two key variants exist of the GSN, namely the Serving GPRS Support Node (SGSN) and the Gateway GPRS Support Node (GGSN).

The SGSN is responsible for the delivery of data packets from and to the mobile devices within its service area and it has the following tasks:
- packet routing and transfer,
- mobility management (attach/ detach and location management),
- logical link management, and
- authentication and charging functions.

The GGSN includes a location register that stores location information, such as current cell and current VLR, and user profiles, such as IMSI and address(es) used in the packet data network, of all GPRS users registered with this SGSN.

The GGSN is the component in the GPRS network that acts as an interface between the GPRS network and external packet-switched networks, such as the Internet. The GGSN has of couple of functions. GPRS packets coming from the SGSN are converted by the GGSN into appropriate packet data protocol (PDP) formats, such as IP or X.25. Subsequently the GGSN sends them out on the corresponding packet data network. In the opposite way, when incoming data packets are received by the GGSN from a packet data network, the PDP addresses of the incoming data packets are converted to the GSM address of the destination user. Then these readdressed packets are sent to the responsible SGSN. For this purpose, the GGSN stores the current SGSN address of the user and his or her profile in its location register. Also, authentication and charging functions are performed by the GGSN. Generally, a many-to-many relationship exists between the SGSNs and the GGSNs, namely a GGSN is the interface to external packet data networks for several SGSNs, and an SGSN can route its packets over different GGSNs to reach different packet data networks.

In the following sections the services provides by T-Sparx are described, which may include several of the above described components.

### 2.2.2 Jiver

The Jiver Platform, also called the Jiver Application Suite, is an application suite that makes a significant contribution as revenue generator for telecom operators, aimed at delivering premium messaging, service provisioning and charging services to telecom operators (Bor, 2010). Jiver’s main contributions are in the following areas:
- Premium Message Routing: routing of external (i.e subscriber or content provider) messages to internal applications and vice versa. Additionally, routing of messages and confirmations for local to international message delivery is also a key feature.
- Premium Service Provisioning: activation and deactivation of premium subscriber services. For example, GPRS, BlackBerry, G3, MMS etc.
- Premium Service Charging: periodic and One-Off Charging for active services.
- Postpaid Reseller Recharge: an online multi-protocol interface for telecom partners for re-selling prepaid credit. This feature supports request/response, batch and periodic interface strategies.
- Billing: in order to support the above services, the Jiver platform must integrate with one or more business support systems (BSS) and operational support systems (OSS) systems used by the telecom services provider. This is a non-intrusive interface that will support any protocol available. Billing involves voucher recharge, non-voucher recharge, payment charging and subscriber validation services.

Telecom operators have access to the Jiver applications through the so-called Jiver Portal, which provides an interface for customer care agents and a platform management interface for the services configuration tasks (see section 2.2.2.1). Figure 2.4 below gives an overview of the Jiver Application suite, with the Jiver applications highlighted in yellow.

Figure 2.4: Jiver Application Suite

Figure 2.5 below provides a more detailed overview, including the system administrators and customer support actors and technologies used between components, namely HTTP, TELNET, JDBC, FILE or SMPP.
2.2.2.1 Jiver Portal

The Jiver Portal is a management web portal that can be accessed by system administrators and members of customer support. These platform portal users are able to define and configure services and their parameters. To do so they have to login onto the Jiver Portal application using credentials provided by the Jiver Support Team or from their own application support department. The Jiver Portal consists of modules and each Jiver portal user account is configured to have a specific role to give access to specific Jiver portal modules. An example of one such role is the ProCharge Supervisor role, which is illustrated in Figure 2.6. This means that this user will have access to the ProCharge module implemented in Jiver Portal and is granted access to all the functionality made available to the ProCharge Supervisor role in that module. This example is illustrated in the following figure, which illustrates that the ProCharge Supervisor role can access the ProCharge Portal and the Access and Application Management modules.
2.2.2.2 ProServ

ProServ enables telecom service providers to provision (i.e. activate, deactivate) services in their own network and in partner networks. It receives provisioning requests generated by customer management system and processes them.

2.2.2.3 ProCharge

ProCharge offers real-time provisioning and charging capabilities for prepaid subscription based services such as GPRS and BlackBerry by implementing a real-time end-user application. Subscribers can activate and deactivate services without human interaction.

2.2.2.4 BillingGateway

The BillingGateway offers the other Jiver Platform components a billing interface to the Telecom operator enterprise billing systems for recharging, charging and voucher transactions. This application is typically not accessible by non-Jiver systems. It is a billing system that tracks customer usage of services and calculates the impact of this usage, based on the price of the service, on the customer’s account.

2.2.2.5 ReCharge Service

The Jiver ReCharge Service enables resellers to recharge subscriber accounts based on mobile technologies and services. The service implements a simple interface that can be used within a broader business transaction, for example a top-up using funds from a bank account (online banking), ATMs or just through point of sale terminals.

2.2.2.6 MessagingGateway

The MessagingGateway offers end-user interaction based on Short Messaging technologies such as SMS and IM. It uses the BillingGateway for charging and subscriber validation. This solution is elaborated in section 2.2.2.

2.2.2.7 RadPas

For an understanding of the RadPas service, first an introduction is required of the RADIUS protocol. The Remote Authentication Dial-In User Service (RADIUS) is a networking protocol developed by the Livingston Enterprises Inc. as an access server authentication and accounting protocol. RADIUS is a client/server protocol where a Network Access Server (NAS), typically a GGSN in the GPRS network, operates as a client of RADIUS and the RADIUS server is a daemon process running on a UNIX or Windows NT machine (CISCO, 2006). The client is responsible for passing information received by the user (typically a user’s supplied credentials) to designated RADIUS servers, and then acting on the returned response by allowing or disallowing the user access to the requested service or resource. RADIUS servers are responsible for authenticating the user after receiving his connection requests, and returning the configuration information necessary for the client to deliver service to the user. A RADIUS server can also act as a proxy client to other RADIUS servers or other kinds of authentication servers. RADIUS is an AAA protocol, where the AAA stands for authentication, authorization and accounting. This means it controls which users are allowed access to which services, and tracks which resources these users have used.
The RadPAS application is an important part of the Jiver Platform, illustrated in the context diagram in Figure 2.7, which offers a simple and affordable solution for subscriber charging for RADIUS managed services. Requests from network service users will be processed by the GGSN and forwarded to RadPAS via one of the RADIUS to HTTP proxies. RadPAS will decide whether the user is allowed access to network services, and also tracks also which resources the user has used. One criterion for (initial) access is the account type (i.e. prepaid/postpaid) and provisioning status (i.e. has a service plan) of the service subscriber. Afterwards the GGSN will keep informing RadPAS of the amount of data and/or time used. RadPAS will eventually indicate to the GGSN, via e.g. KannelCG, when the subscribers account balance runs out. KannelCG is a RADIUS proxy, which means it will receive the UDP packets from the GGSN, record data for the mapping between the MSISDN and the IP address, and forward the UDP datagram again to a final RADIUS server. RadPAS uses the BillingGateway for subscriber validation balance checks and charging when the subscribers prepaid account balance is the basis for service access. RadPAS interfaces with a Rating System to query for charging tariffs for data and time used. The Portal user interface, described in 2.2.2.1, offers access to the operator to view service usage and charging information for subscribers.

### 2.2.3 Messaging solution architecture

T-Sparx currently provides a messaging solution to telecom operators based on the SMPP and HTTP protocols called the SMS Gateway. The SMS Gateway is one of the core services provided by T-Sparx. When applications or services provided by content providers need to send or receive SMS messages a connection must be made with SMSCs of telecom operators. This connection can be made using the Short Message Peer-to-Peer (SMPP) protocol, described in section 2.1.1 or the Computer Interface to Message Distribution (CIMD). Thus, the application of the content provider must support one of these protocols, usually the SMPP protocol, to be able to connect to an SMSC. This makes it difficult, time consuming, complex and expensive for content providers to implement SMPP compatible applications. For this reason T-Sparx has come with the SMS Gateway solution, which is placed between the content provider’s application and the SMSC. The SMS Gateway can thus be used as both ESME and SMSC. Since more than one SMSC can connect to the same SMS Gateway, the SMS Gateway ensures for correct routing of SMS messages. The SMS Gateway supports an HTTP interface making it easier to use for content providers. The SMS Gateway also ensures correct billing and reverse billing of SMS messages, besides the correct routing of SMS messages from customers to applications.

Figure 2.8 shows communication between two SMSCs via the SMS Gateway. Most SMSCs support the SMPP protocol, which is typically used as the communication protocol. SMSCs are servers and can receive requests only from ESMEs, which means that SMSCs cannot directly communicate. To solve this, the SMS Gateway is placed between two SMSCs enabling these to communicate.
SMSCs communicate with the SMS Gateway using the SMPP protocol, while end users interact with the SMS Gateway using the Simple Object Access Protocol (SOAP) or Representational State Transfer (REST) protocols. SOAP and REST both use HTTP as the transmission protocol. The SOAP protocol is used to exchange structured information in the implementation of web services. Its message format is based on XML Information Set and usually uses the HTTP or Simple Mail Transfer Protocol for message negotiation and transmission. REST is a method for sending and receiving data, as JSON, XML or even plain text, between client and server, and is considered more lightweight than SOAP.

The SMS Gateway includes two services, namely the SMS service and the Subscriber service as can be seen in Figure 2.9 above. The SMS service is responsible for routing incoming requests to their destination. The subscriber service enables content providers to acquire subscriber information. Subscribers can be placed in so-called “white and black lists”, when requesting a new service. Subscribers on the white list are charged for the specific service, e.g. receive interesting notifications. Subscribers on the black list are not charged for the service, and will not receive any SMS messages. For a more detailed technical description and overview of the SMS Gateway please refer to (Christiaan, 2011).
2.2.4 Messaging solution infrastructure architecture

The infrastructure architecture currently used by a common client of T-Sparx for a production environment can be seen in figure 2.10. Each application provided by T-Sparx typically consists of a cluster that includes two servers. In such a scenario where two servers are used, one server is used as the main production server (the master), receiving requests sent by clients and processing them, and one is used as a backup replica server (the slave), which is turned on when the first main server is down. Next follows a customer scenario as a concrete example.

**Customer scenario**

For the sake of a concrete example, a real-world scenario is generalized as follows. In *Figure 2.10 Typical Infrastructure architecture of production environment*, an example infrastructure architecture is shown of a client Telesur, which has in-house servers available for running applications provided by T-Sparx. These servers typically are obtained by Telesur itself and are subsequently installed, configured and hosted by T-Sparx to run all necessary production applications. In this figure 5 components are shown, namely the Jiver Platform, the Telesur backend, the 3rd party backend, the content providers and mobile device users (iOS/Android/BB). In this example the Jiver Platform and the Telesur backend are used for describing the infrastructure architecture. The Jiver Platform consists in this example of four components, i.e. the RadPas application (nodes TLSNET01 and TLSNET02), the databases (node TLSDB01 and TLSDB02) and the SMS gateway (node TLSAPP01 and TLSAPP02). The RadPas application is a Master-Master High Availability Cluster, implying that both nodes in this cluster run in parallel and are provided with requests based on the Round-robin DNS technique. The others are Master-Slave clusters, implying that only one node in the cluster runs at a time as the active production server, while the other acts as a backup server that hosts identical services as the first node. When the active server fails, Corosync Pacemaker ensures that the application on this node is recovered and made available on the other node. T-Sparx uses Corosync Pacemaker to automatically turn on a slave (backup node/server) on failure of the main active production node. Corosync Pacemaker is a scalable high availability cluster resource manager, which supports different deployment scenarios ranging from the simplest 2-node standby cluster to a 16-node active/active configuration. It monitors the system for both hardware and software failures, and in the event of a failure it automatically recovers the application and makes sure it is available on one of the other nodes in the cluster. Pacemaker uses advanced algorithms to rapidly determine after a failure, the optimal locations for services based on pre-defined constraints, such as relative node preferences or requirements to run with other cluster services.
2.3 Problems faced with current messaging infrastructure architecture

In this section the RadPas service is taken as an example to describe a major problem concerning the availability of the RadPas service when load bursts occurs, which leads to long queues of requests and a lot of request failures, leaving mobile device users waiting on a service activation that may never happen. Using this customer scenario the main problem currently faced by T-Sparx is denoted, where one or two servers may not be sufficient to handle a burst of requests, and alternative solutions are required to provide high availability of services to customers.

Customer scenario
For the sake of a concrete example, a real-world scenario is generalized as follows. A content provider (i.e. a client of T-Sparx) provides particular services to customers and charges these customers for the activation or usage of these services. To make this possible this content provider makes use of the RadPas service provided by T-Sparx, which is described in section 2.2.2.7. In figure 2.11 an abstract view of the communication between a mobile device and the RadPas system is given.
Figure 2.11: Abstract view of mobile device with RadPas communication

Figure 2.11 consists of the mobile device, a cluster consisting of GGSN servers and RadPas. Customers owning a mobile device with a GPRS data plan may pay on a monthly, weekly or daily basis and are provided with defined data access limits. As customers use this data plan by browsing on the Internet, streaming online videos, downloading images etc., their data access is updated to keep track when the data in the data plan runs out and is subsequently terminated. RadPas is used in this example to determine is a user may or may not access this network service. Consider a scenario when there is an outage and the active server with RadPas goes down. In this case no customers can access the services they paid for. When the outage problem is solved and this server comes up, all customer’s devices automatically will send a request to RadPas via the GGSN to reactivate the services that they paid for. In this case requests of all active customers are received at the RadPas production server, requesting for reactivation of their service(s). RadPas will process the response by checking if the user is allowed to use the requested service and will send a response to GGSN, which will activate the requested service on a success response. The production server will receive a huge amount of requests at the same time, leading to long queues being formed of requests to be processed. This implies that the processing of all requests will take a substantial amount of time. Also GGSN will expect a response from RadPas within a certain period, and when this period times out, the GGSN will retransmit the request to RadPas. This way RadPas keeps getting requests to process and will try to process all. Each time the GGSN timeouts a request and retransmits it, the transaction id of the request changes. RadPas keeps checking each request for its MSISDN (phone number) and processes each new request, based on its MSISDN number. This means that a request that has been dropped by the GGSN, retransmitted to RadPas and finally processed by RadPas is returned to the GGSN for activation of the service, while the transaction id has already been discarded. So this request has been unnecessarily processed. This way the RadPas server can keep getting requests, forming a long queue of requests to process, and many times unnecessarily. This causes services to be activated after long delays, causing of customer dissatisfaction. This shows that an infrastructure consisting of a single server cannot handle this type of real world problems.

2.4 Possible solutions and challenges

As described in section 2.3 owning a limited number of resources, while having an increasing number of customers may lead to serious problems. At a certain time peaks where e.g. all customers send requests at the same time the limited resources available cannot handle the load, leading to failures in activation of services and unsatisfied customers.
A first solution could be purchasing the required infrastructure to fulfill the needs of the company, creating an own cloud (described in Chapter 4) for example, that scales out and scales in as required. But the huge upfront investment in required infrastructure and maintenance of the Cloud makes this solution challenging. Some challenges that are faced when creating a private cloud are given by Claybrook (2010). The company investing in such infrastructure for a private cloud should figure out what the upper and lower bounds are for their return on investment (ROI). Also when having a private cloud in place the possibility must be in place to move to a hybrid cloud (described in Chapter 4) if public services are required, or when certain business application can be deployed in the public cloud distributing this way some of the workload of the private cloud to the public cloud. This integration of the private cloud with the public cloud involves security challenges and also making sure that the workloads work properly in both places. Current servers and other existing infrastructure may need to be disassembled as they are working to move into the private cloud, causing problems. One other important factor to consider when investing in private clouds is that the complexity and the speed of technology changes require companies to protect that investment by making sure it stays up to date with new releases of software components. Also it may be the case that the IT team of the company is not familiar with private clouds requiring a learning curve, which in fact is a growth opportunity for the people at the company by keeping them up to date with new skills of the today’s business environment.

A different possible solution includes using load-balancing technology with existing infrastructure, i.e. servers. But still relying on existing infrastructure requires maintenance and great investments in additional resources, such as servers to guarantee Quality of Service (QoS), i.e. guarantee a certain level of performance of the system/service to customers. Another solution would be using Amazon’s Elastic Load Balancing technology, by having a defined number of servers and load balancing incoming requests in a round robin manner to these available servers. An alternative solution would be using a public cloud-computing platform, such as Amazon to deploy all messaging services described above, and using the Auto Scaling feature provided by Amazon to scale out and scale in the infrastructure as required.

A last alternative would be to combine the Amazon features of Auto Scaling and Load Balancing. These alternatives, with the exception of the last mentioned, are described in Chapter 4 Cloud computing and evaluated for possible performance tests in Chapter 7 Performance tests. According to Casaretto (2013), the elements of risk, security and compliance have formed a significant barrier to public cloud adoption. For enterprises it is difficult to make the switch to the open cloud because of the lack of control and visibility into a cloud provider’s infrastructure. The transparency that Amazon operates with makes it difficult for enterprises to trust in such public cloud solution. According to Summit Security Consultant and industry veteran Eddie Mize there exists significant risk in using public clouds, such as Amazon. Security could be compromised by faulty or malicious APIs or other faults. He stated: “The cloud in various forms, and in any multitenant environment is by nature, insecure... In regards to the enterprise, it is my advice to proceed and pursue benefits in certain situations, but do so with caution and be strategically aware of risks, plan accordingly”. Compliance is a significant barrier to large enterprise adoption, due to Amazon’s limited flexibility in letting customers do among others things audits of its security.
Chapter 3

IP Multimedia Subsystem

This chapter introduces the IP Multimedia Subsystem (IMS) architectural framework, to provide a basic understanding of this technology, which is used in chapter 5 to design and partially implement an IMS based messaging solution. Section 3.1 introduces IMS with the IMS definition and a little about the IMS history and its emergence in section 3.2. The core IMS standards are described in section 3.3 and the IMS architecture consisting of the core components are described in section 3.4, followed by a brief overview of the steps required to get the IMS service in section 3.5. Section 3.6 elaborates the security aspect in IMS. Section 3.7 gives an overview of the IMS security challenges and section 3.8 finalizes the chapter with an overview of possible IMS solutions that can be used in practice.

3.1 IMS definition

Third generation (3G) networks aim at merging the most successful paradigms in communications, i.e. the cellular network and the Internet. The IP (Internet Protocol) Multimedia Subsystem (IMS) is considered, according to Camarillo & Garcia-Martin (2008), the key element in the 3G architecture that makes it possible to provide cellular access everywhere and at any time to all the services provided by the Internet. IMS is a standardized Next Generation Networking (NGN) architecture for telecom operators that want to provide mobile and fixed multimedia services. A NGN is a packet-switched network where the packet switching and transport elements are both physically and logically separated from the service/call control intelligence. Transport elements include routers, switches, gateways etc., and service/call intelligence is used to support all types of services over the packet-based transport network. More about Next Generations Networks can be found in the report written by Lopez Barbosa (2011).

IMS is defined in Gałczyńska et al. (2008) as follows:

“IMS is a standardized architectural framework specified by the 3GPP, characterized by an Internet Protocol and Session Initiation Protocol (SIP) based architecture for mobile operators who aim to employ Internet, real-time services into cellular telephony.”

In Porter et al. (2006) IMS is defined as follows:

“The IP Multimedia Subsystem is a next-generation multimedia communication framework that encompasses mobile, fixed, packet-switching, and traditional circuit-switching communication systems.”

The standard was first specified by the Third Generation Partnership Project (3GPP/3GPP2) and was later embraced by other standards bodies including ETSI/TISPAN. More about these standardization bodies can be found in the next “IMS History” section. The objective of IMS is described as follows according to 3GPP (2010):

“The objective of this feature is to efficiently support applications involving multiple media components as video, audio, and tools like shared online whiteboards, with the possibility to add and drop component(s) during the session. These applications are called IP Multimedia applications (or “services”).”
The IP Multimedia Subsystem architectural framework defines the standard interfaces, based on Internet protocols, to be used by service developers, enabling operators to take advantage of a powerful multi-vendor service creation industry, avoiding sticking to a single vendor to obtain new services. The idea of IMS is to offer users Internet services ubiquitously at any time using cellular technologies.

The Internet has experienced dramatic growth over the last decade evolving from a small network linking some research sites to an enormous worldwide network, due to the ability to provide useful services, such as the World Wide Web, e-mail, Voice over IP (VoIP), instant messaging, videoconferencing and presence services. Also because the Internet uses open protocols that are available on the web for any service developer, with all tools taught at universities and described in a large number of books, people with wide knowledge in some specific area can develop high quality Internet services in that specific area. Cellular telephone networks provide services to over one billion users worldwide, including telephone calls, messaging services ranging from simple text messages (such as SMS) to multimedia messages (such as MMS), and also connect to the Internet using a data connection. On one hand, IMS is considered the key element to make possible that Internet services are offered everywhere and at any time using cellular technologies, and on the other hand is given that cellular networks provide Internet services, such as instant messaging, by using data connections. The question arises of what is IMS needed for? To clarify what is meant with merging the Internet with the cellular world, the circuit-switched and packet-switched domains in 3G networks are introduced.

Circuit-switched domains are an evolution of the technology used in 2G networks. Circuits in the circuit-switched domain are optimized to transport voice, video and instant messages. Circuit-switched is being replaced with the more efficient packet-switched technology. A circuit-switched communications network is a network in which a dedicated communication-channel, or circuit, is established for the duration of a transmission or session, and released only when the session terminates. The telephone system (PSTN) is the most universal circuit-switched network. According to Roberts (1978), since interactive data traffic occurs in short bursts, 90 percent or more of the pre-allocated bandwidth is wasted. Communications networks were completely redesigned with technological advancements, introducing the concept of packet switching. With packet switching the exclusive access to radio resources is not anymore necessary for data applications. The bandwidth is dynamically allocated, without attempting to pre-allocate bandwidth over the whole source-to-destination path. The transmitted information is divided into small chunks of data, called ‘packets’, ‘segments’ or ‘datagrams’, which move through the network in a manner similar as the manual handling of mail. These packets contain in addition to the data, information about the IP address of the source and the destination, sequence numbers and some other control information. The packets are sent towards the destination irrespective of each other and they arrive at random intervals, forming queues at intermediate nodes, waiting for an outgoing channel to become available. Each packet finds its own route to the destination without any predetermined path, using the information it carries such as the source and destination IP addresses.

The packet-switched domain provides IP access to the Internet, while in the circuit-switched domain a 2G terminal can act as a modem for the transport of IP packets. 3G terminals use packet-switched technology to transmit data, which is much faster, and the available bandwidth for Internet access is greater. Users can access many more services, such as surf the web, read emails, download videos, using packet-switched technology. All the power of the Internet is available for 3G users through the packet-switched domain, but still IMS provides the following three main benefits compared to the packet-switched domain:
- **Quality of Service (QoS)**
  To provide real-time multimedia services IMS takes care of synchronizing session establishment with QoS provision so that users have a predictable experience, while the packet-switched domain provides a best-effort service without QoS, i.e. no guarantee is offered about the amount of available bandwidth a user gets for a particular connection or about the delay packets experience. For example with a VoIP conversation the quality of the conversation can vary dramatically.

- **Charging**
  IMS provides the ability to charge multimedia sessions appropriately. It enables operators to charge for a service as they think most appropriate. IMS provides information about the service being accessed and used by the user, making it possible for operators to decide whether to use traditional time-based charging, QoS-based charging, use a flat rate, or perform its own type of charging for the service. Typically operators charge for the number of bytes transferred because the operator is not aware of the contents of those bytes, i.e. the service being used by the user is unknown.

- **Integration of different services**
  IMS provides the ability to operators to use existing services provided by third parties, and combine them, integrate them with their own services, this way providing customers with complete new services. By deploying services on one or more separate Application Servers (described in section 3.4.5), these services (applications) are made accessible to service providers to be combined with each other, this way creating a single specialized service that consists of many features, such as presence service and video calling, which themselves are located on different SIP application servers.

In the following section the emergence of the IMS technology is described, including a description of the main standardization bodies that made IMS possible.

### 3.2 IMS history

The IMS architectural framework was originally defined by an industry forum called the 3G.IP, which was formed in 1999. The 3G.IP developed the first IMS architecture and handed it over to the Third Generation Partnership project (3GPP). The 3GPP, established in 1998, is a collaboration agreement between a number of telecommunication standardization bodies, as part of their standardization work for supporting the evolution of the system to a third generation cellular system, and to support the evolution of radio technology, with an emphasis on new higher bandwidth radio access methods. IMS was originally defined by 3GPP in the 3GPP Release 5 specification document (3GPP, 2010). The framework has been further enhanced in Releases 6 to Release 11, adding support for WLAN and fixed networks and to include features like presence and group management. 3GPP2 (3GPP2, 2012) is another standards body born to evolve North American and Asian cellular networks based on the standards ANSI/TIA/EIA-41 and CDMA2000 radio access into a third generation system (Camarillo & Garcia-Martin, 2008), (Chen et al., 2006). The 3GPP and the 3GPP2 have both standardized their own IMS. The 3GPP and 3GPP2 established collaboration with IETF to make sure protocols developed by IETF meet their requirements. The IETF is the standardization body, consisting of network designers, operators, vendors and research institutions that work together to develop the architecture, protocols, and operation of the public Internet (Camarillo & Garcia-Martin, 2008). It has developed most of the protocols that are currently used on the Internet. Other standardization bodies include the Open Mobile Alliance (OMA, 2010) and the European Telecommunications Standards Institute (ETSI, 2010). The OMA is the leading industry forum for developing market driven, interoperable mobile service enablers, and plays in addition to the 3GPP, 3GPP2 and IETF, an important role in developing IMS services.
3.3 IMS standards

This section provides a brief description of the main standards of the IMS architectural framework. The intention is to give a concise introduction of these standards to be able to understand this framework.

3.3.1 Session control protocol

For IMS three different session control protocols can be used, namely the Bearer Independent Call Control (BICC), H.323 and the Session Initiation Protocol (SIP). The latter was chosen by 3GPP as the session control protocol for IMS, and is the one described in this section. BICC’s specification is found at ITU-T (2008), while the specification for H.323 can be found at ITU-T (2003).

Session Initiation Protocol

The SIP protocol (Rosenberg et al., 2002) is a signaling protocol that is used to create, manage and terminate sessions, such as a simple two-way telephone call or a collaborative conference session, in an IP-based network. SIP was developed within the SIP working group in the IETF and was selected by the 3GPP in 3GPP Release 5 specification document (3GPP, 2010) as the session control protocol standard for IMS. The reason for this is its flexible syntax and to facilitate development and interconnectivity between 3GPP networks and fixed networks. SIP follows the client-server model and is composed of many design principles from the Simple Mail Transfer Protocol (SMTP) and the Hypertext Transfer Protocol (HTTP). Since SIP is based on the HTTP protocol, SIP services can be developed by using frameworks like Common Gateway Interface (CGI) and Java servlets, primarily developed for HTTP. SIP employs design elements similar to the HTTP request/response transaction model. Each transaction consists of a client request that invokes a particular method or function on the server and at least one response. SIP reuses most of the header fields, encoding rules and status codes of HTTP, providing a readable text-based format. SIP is more of a component that can be used with other IETF protocols to build a complete multimedia architecture. Examples of SIP services include voice-enriched e-commerce, web page click-to-dial or Instant Messaging with buddy lists in an IP based environment.

3.3.2 Diameter: the Authentication, Authorization and Accounting (AAA) protocol

Diameter (Calhoun, 2003) is the protocol chosen in IMS as the Authentication, Authorization and Accounting (AAA) protocol (pronounced “triple A” protocol). The Diameter protocol is widely used on the Internet to perform AAA, and it is a development of the RADIUS protocol, which is described in section 2.2.2.7., with improved transport, improved proxy, enhanced session control and higher security (Calhoun, 2003).

3.3.3 Real-time Transport Protocol and RTP Control Protocol

The RTP (Schulzrinne et al., 2003) and RTCP protocols are used as the transport protocol to transport real-time media, such as video and audio. RTP is standardized by the IETF and can be seen as a data transmission protocol. An example is in the case of VoIP, where it is used to transmit voice, but is also useful for the transmission of video or some real-time measurement data.

3.3.4 H.248 media control protocols

The H.248 protocol, also referred to as the Media Gateway Control (MEGACO) protocol (Cuervo et al., 2000), was developed by the ITU-T together with IETF and is a control protocol used between media
control functions and media resources. Media control functions include the Media Gateway Control Function (MGCF), described in section 3.1.4.9 and the Media Resource Function Controller (MRFC) described in section 3.1.4.6. Media resources include the Media Gateway (MGW), described in section 3.1.4.9, and the Media Resource Function Processor (MRFP), described in section 3.1.4.6.

3.3.5 IPv6

Internet Protocol version 6 (IPv6) is a network-layer IP standard used by devices to exchange data across a packet-switched network. It is the successor of the IPv4, which was also deployed for IMS from the very first IMS release, 3GPP Release 5. Currently dual-stack implementations (IPv4 and IPv6) are allowed in both IMS terminals and IMS network components. To make this possible the IMS Application Layer Gateway (IMS-ALG) and Transition Gateway (TrGW) were added to the IMS network.

3.4 IMS architecture and features

The IMS architecture can be represented in a layered and non-layered way. The layered IMS architecture consists of three layers (Chang et al., 2010):

- The Media/Transport Pane,
- The Control/Signaling Plane, and
- The Service/Application Plane

![Figure 3.1: Layered architecture of IMS (Chang et al., 2010)](image)

The Media/Transport plane refers to a wide range of different access technologies, which is the access network in the non-layered architecture figure. Within the IP transport layer users acquire network connectivity through e.g. DSL, Ethernet, Wireless LAN, GPRS or UMTS (Universal Mobile Telecommunication Systems) to acquire network connectivity. After connecting to the IMS network, users can access multimedia services. This is the Control/Signaling plane, which consists of the IMS
core components, which are described in following sections, including the HSS, and the CSCFs. SIP requests are processed and routed to the destination in this plane. The Service/Application plane consists of different application servers, which provide users with a wide variety of IMS services.

The non-layered IMS architecture is illustrated in Figure 3.2, which is called the IP Multimedia Core Network Subsystem according to (3GPP TS 23.228, 2011), showing the major components in IMS, which are linked by standardized interfaces.

![Figure 3.2: IMS architecture consisting of majority of components](image)

In the following figure the main components to make basic text messaging possible in IMS are highlighted, including the P-CSCF, I-CSCF, S-CSCF, HSS and SIP-AS. Figure 3.3 shows only these components and corresponding interfaces for illustration purposes.

![Figure 3.3: IMS architecture highlighting the components needed for basic messaging solution](image)
In following sub-sections the main components to make text messaging possible are explained followed by other core components to make multimedia services, and also the communication with circuit-switched networks, possible.

### 3.4.1 Access network

To connect to the IMS network users can use various methods that use the standard IP Protocol (IP) to acquire connectivity. IMS terminals, including mobile phones, computers or PDAs can directly register into an IMS network, either in the home or in a visited network. These terminals must support IPv6 and must run SIP User Agents. The following access technologies are all supported to connect to the IMS network:

- Fixed access technologies, such as DSL, Ethernet, cable modems etc.,
- Mobile access technologies, such as W-CDMA, CDMA2000, GSM, GPRS etc.,
- Wireless access technologies, such as WLAN, WiMAX etc., and
- Other phone systems like H.323, non-IMS-compatible VoIP systems are supported through gateways.

### 3.4.2 The Home Subscriber Server (HSS) and Subscription Locator Function (SLF)

The Home Subscriber Server (HSS) is the master user database in the IMS network that supports the IMS network entities that are actually handling the calls/sessions, by providing user- and subscription-related information, i.e. user profiles. The HSS performs a couple of functions including:

- user authentication,
- user authorization,
- mobility management, i.e. provide information about the physical location of user,
- identification handling,
- service authorization support,
- service provisioning support,
- session establishment support.

The HSS is similar to the Home Location Register (HLR) and Authentication Center (AUC) in a GSM network. When a user registers in the IMS network, the user profile information is downloaded from the HSS to be used by the I-CSCF and the S-CSCF (both Diameter clients), enabling these to establish a
session for the user. Multiple HSS can be used in an IMS network when the number of subscribers is too large to be managed by a single HSS. This redundant configuration of HSS servers is considered as a single logical node, meaning that all the data related to a particular user is stored in a single HSS, the HSS currently in use. This is where a Subscriber Location Function (SLF) is needed, which locates the HSS that holds the subscription data for a user. The SLF maintains a table that maps the IP Multimedia Public identities of users to the address of the HSS that contains the user information. Both the HSS and the SLF implement the DIAMETER protocol. The interface between the HSS and both the I-CSCF and the S-CSCF is the Cx interface, while the interface between the SLF and both the I-CSCF and the S-CSCF is the Dx interface, as can be seen in figure .. above.

3.4.3 Identification in IMS

In any network it must be possible for operators to uniquely identify users, so that a call can be directed to the proper user. In general, the following identities are used for users in 3GPP networks:

- International Mobile Subscriber Identity (IMSI), which is an identity only visible to the network and not to users used in GSM networks
- Temporal Mobile Subscriber Identity (TMSI), which is assigned to a subscriber by the VLR, described in section 2.2.1, in a GSM network, to uniquely identify a user in the local area handled by the VLR.
- International Mobile Equipment Identity (IMEI), which is a unique device identity and is phone specific
- Mobile Subscriber ISDN Number (MSISDN), which is a telephone number allocated to a user.

In IMS operators assign each user with both a Public User Identity, the IP Multimedia Public Identity (IMPU) and a Private User Identity, the IP Multimedia Private identity (IMPI), which are both URLs and not phone numbers. Besides the IMPU and IMPI used to identify users, the Public Service Identity is used to identify services.

**IMPU**

The public identity is used for routing SIP requests and has the format of a SIP URI or TEL URI. The operator may assign one or more IMPUs to a user. The SIP URI is required by the SIP protocol, and has the format of sip:first.last@operator.com or sip:phonenumber@operator.com. The TEL URI is used to make a call from IMS to a PSTN phone, and also when making a call from PSTN to IMS. The IMPU is to the IMS network what the MSISDN is to the GSM network.

**IMPI**

The private identity is exclusively used for subscription identification and authentication purposes and is unique for a user. The IMPI performs a similar function as the IMSI in a GSM network. The IMPI has the Network Access Identifier (NAI) format: username@operator.com.

**PSI**

The PSI is identity allocated to services hosted in an Application Server, which is described in section 3.1.5.4. PSIs have the format of a SIP URI or TEL URI, just like the IMPU. This identity does not have an associated private user identity, which is used for user authentication. The I-CSCF interfaces directly with the Application Server in order to route incoming SIP requests addressed to PSIs.

The HSS user database contains, but is not limited to, the IMPU, IMPI, PSI, IMSI, and MSISDN.
3.4.4 The Call/Session Control Function components

An essential component in the IMS network is the Call/Session Control function (CSCF). It is a SIP server, which is used to process SIP signaling in the IMS. According to Galczyńska et al. (2008) the CSCF’s main function is providing session control for terminals. This includes the secure routing of SIP requests, which is followed by the monitoring of the SIP session and communication with the policy architecture to support media authorization. The three types of CSCFs are described as follows:

- **Proxy CSCF (P-CSCF)**

  The P-CSCF is a SIP proxy and is the first point of contact for the IMS terminal in the IMS network (Camarillo & Garcia-Martin, 2008). As a SIP proxy server the P-CSCF receives all requests sent by the IMS terminal or sends all responses destined to the IMS terminal from the IMS network. It is located in the home network when the visited network is not yet IMS compatible, while in a full IMS network it is located in the visited network. A terminal performs the P-CSCF discovery procedure to attach to the IMS network prior to performing IMS registration and initiating SIP sessions. After registration in the IMS network, the IMS terminal is assigned a P-CSCF for the entire duration of the registration. All signaling messages traverse the P-CSCF, therefore it can inspect every request and verify every request’s correctness. The P-CSCF authenticates the user and establishes an IPsec security association with the IMS terminal. This prevents spoofing attacks and replay attacks and protects the privacy of the user. Other nodes trust the P-CSCF, and do not have to authenticate the user again.

- **Interrogating-CSCF (I-CSCF)**

  The I-CSCF is also a SIP proxy and is located at the edge of the IMS network (Balakrishna & Al-Begain, 2007). This SIP proxy acts as an entry point for SIP requests from external networks. Its IP address is published in the Domain Name System (DNS) of the domain, so that remote servers, such as a P-CSCF in a visited domain or a S-CSCF in an external domain, can find it. When external domains find the I-CSCF, it is used as a forwarding point for SIP packets to this domain, for example for registration purposes. The I-CSCF, which is usually located in the home network, has an interface with the HSS to retrieve user information to find out to which S-CSCF the user is registered to, and if not yet registered, to select a new S-CSCF. The I-CSCF forwards requests and responses to the S-CSCF, and it may also implement an interface to the Application Server so that requests can be routed that are addressed to services instead of users. An I-CSCF Topology Hiding Interface Gateway (THIG) is an I-CSCF that is used to hide the internal network from the outside world by encrypting part of the SIP message.

- **Serving CSCF (S-CSCF)**

  The S-CSCF is the central node of the signaling plane in the IMS network, and is always located in the home network (Camarillo & Garcia-Martin, 2008). It is a SIP server but also performs session control. The S-CSCF implements a Diameter interface to the HSS:

  - To download authentication information to authenticate the user,
  - To download user profile of the user that includes which services the user is subscribed to,
  - To save the S-CSCF in the HSS to which the user is allocated for the duration of the registration.

  Because all signaling messages also traverse the S-CSCF, it can inspect every message, to decide to which Application Server the SIP request is forwarded to, in order to provide the services. Also the S-CSCF provides routing services, which includes translation services for translation of a telephone number to a SIP URI. In addition, the S-CSCF enforces the policy of the network operator, keeping users from performing unauthorized operations.
3.4.5 The Application Server (AS)

An Application Server is a SIP entity that hosts and executes services and it resides either in the user’s home network or in a third party location. The third party could be a network or simply a stand-alone AS. The AS interfaces with the S-CSCF, making the integration and deployment of value-added services to the IMS network convenient for third-party providers. Also the AS interfaces with the I-CSCFs via the SIP protocol, and the HSS via the Diameter protocol, to download and upload data related to a user stored in the HSS. If the AS is located outside the home network there is no interface with the HSS. Also the AS can let IMS terminals interface with it for configuration purposes. There are three types of ASs in IMS, all behaving as SIP ASs for the IMS network:

- SIP AS, which is the native AS hoisting and executing IMS services based on the SIP protocol. New IMS services are deployed on this AS.
- Open Service Access-Service Capability Server (OSA-SCS) AS, which provides an interface to the OSA framework AS. OSA describes how services are designed in a UMTS network.
- IP Multimedia Service Switching Function (IM-SSF), which is a specialized AS used to enable services that were previously developed for GSM (Customized Applications for Mobile network Enhanced Logic - CAMEL) to be reused in IMS.

A single SIP AS can host multiple services or it can be used for just one service (Ericsson, 2007). Several services running on different SIP ASs can be combined to create a single specialized service, consisting of many features, such as presence service and video calling. The main benefits of SIP application server technology are: ease of application development, rapid network and centralization, making the combination of different services running on different ASs possible.

3.4.6 The Media Resource Function

The Media Resource Function (MRF) provides a source of media in the home network (Camarillo & Garcia-Martin, 2008), and is always located in the home network. According to Porter et al. (2006), the MRF is used for playing announcements (audio/video), multimedia conferencing (e.g. mixing of audio streams), text-to-speech (TTS) conversion, speech recognition, and real-time transcoding of multimedia data (i.e. conversion between different codecs). The MRF can be divided into the following:

- Media Resource Function Controller (MRFC), which is a signaling plane node that acts as a SIP User Agent, and contains a SIP interface towards the S-CSCF. The MRFC controls the MRFP via the H.248 interface described in section 3.1.4.4
- Media Resource Function Processor (MRFP), which is a media plane node that implements all media related functions, such as playing and mixing media.

3.4.7 The Breakout Gateway Control Function (BGCF)

The Breakout Gateway Control Function (BGCF) is a SIP server that includes routing functionality based on telephone numbers. The BGCF is only used when a session is started by an IMS terminal, and addressed to a user in a circuit-switched network, such as the Public Switched Telephone Network (PSTN) or the Public Land Mobile Network (PLMN) (Porter et al., 2006).

3.4.8 IMS Application Layer Gateway (IMS-ALG) and Transition Gateway (TrGW)

IMS supports two IP versions, both IPv4 and IPv6 (described in section 3.1.4.5), which means that interworking between these two may occur. To make this possible without requiring support from the terminal, the IMS-ALG and the TrGW make part of the IMS network. The former deals with SIP interworking and the latter with RTP interworking (between IPv4 and IPv6 and vise versa).
3.4.9 The PSTN/CS Gateway

The PSTN/CS gateway interfaces with circuit switched networks, enabling IMS terminals to make calls to and receive calls from a PSTN or any other circuit-switched network (Figure 3.5). The PSTN/CS Gateway is split into the following three functions:

- **Signaling Gateway Function (SGW)**
  The SGW interfaces the signaling plane of the circuit switched networks and performs lower-layer (transport layer) protocol conversions.

- **Media Gateway (MGW)**
  The MGW interfaces the media plane of the circuit-switched network and basically sends and receives IMS media over the Real-Time Transport Protocol (RTP), and connects to a circuit-switched network using one or more Pulse Code Modulation (PCM) time slots.

- **Media Gateway Control Function (MGCF)**
  The MGCF is the central node of the PSTN/CS gateway, which implements a state machine that is in charge of call control protocol conversion, mapping SIP to ISDN User Part (ISUP) over IP or Bearer-Independent Call Control (BICC) over IP (ISUP and BICC both call control protocols in a circuit-switched network).

![Figure 3.5: PSTN/CS gateway interfacing a circuit-switched network](image)

3.5 Fundamentals for getting the IMS service

This sub-section provides a brief overview of the steps required to get the IMS service, as can be seen in the following figure.
Before starting any IMS-related operation an IMS host must meet the following prerequisites (Camarillo & Garcia-Martin, 2008):

- Primarily the IMS service provider must authorize the end user to use the IMS service, which typically requires a subscription or contract signed between the IMS network operator and the end user. This contract can be compared to the subscription that authorizes an end user to receive and establish telephone calls over a wireless network.

- Next, connection between the IMS host and an IP Connectivity Access Network (IP-CAN) is required, such as GPRS (in GSM/UMTS networks), WLAN (Wireless Local Access Network) or ADSL (Asymmetric Digital Subscriber Line). The IP-CAN provides access to the IMS home network or to an IMS visited network. The concepts of home network and visited network are borrowed from GSM and GPRS. The home network is the infrastructure provided by the network operator in the residential area, while the visited network is the infrastructure outside the area of coverage of the home network, when roaming. The IMS host must acquire an IP address, following the same procedures as with GPRS as described in 3GPP TS 23.060 (2008). Normally the IP-CAN operator dynamically allocates the IP address to the IMS host for a limited period of time.

- After these two prerequisites are achieved, the P-CSCF discovery procedure is started, i.e. the IMS host gets to know the IP address of the P-CSCF, which acts as an inbound/outbound SIP proxy server. The inbound proxy server receives requests from IMS hosts, while the outbound proxy server sends responses or requests to IMS hosts. All SIP requests sent by the IMS host traverse the P-CSCF. After discovering the P-CSCF, which is allocated permanently for the duration of the IMS registration, the IMS host can start sending SIP requests to the P-CSCF and receiving responses from the P-CSCF.

- After fulfilling these prerequisites the IMS host registers at the SIP application level to the IMS network by regular SIP registration. This is called the IMS level registration. The IP-CAN layer (e.g. registration to a GPRS network) is independent from the IMS level registration. At the IMS level,
the registration allows the IMS network to obtain the IP address of the IMS host, also authenticate the user, establish security associations, and authorize the establishment of sessions.

A more detailed explanation of these steps can be found in Camarillo & Garcia-Martin (2008), sections 5.2 to 5.7.

3.6 IMS security

According to Camarillo & Garcia-Martin (2008), IMS security is divided into access security (3GPP TS 33.203, 2008) and network security (3GPP TS 33.210, 2008) as described in the following two subsections:

3.6.1 Access security

Access security includes authentication of users and the network, and the protection of the traffic between the IMS UE and the network. A user accessing the IMS network first needs to be authenticated and authorized to use IMS before he can use a service. The S-CSCF authenticated and authorizes the user with the authentication information downloaded from the HSS. Subsequently security associations are established between the user and the P-CSCF, which is either an IPsec connection, a TLS connection or the security association is supported by the IP-CAN. IMS supports several authentication mechanisms, where the actual mechanism used is determined by the presence of a security module or smart card in the IMS host, such as a Universal Integrated Circuit Card (UICC), and the connection of the IMS network to the IP-CAN. The most important authentication mechanisms in SIP include:
- HTTP Digest Access Authentication
- HTTP Digest Access Authentication using Authentication Key Agreement (AKA)

The description of these two fall outside the scope of this report and can be found in Camarillo & Garcia-Martin (2008).

3.6.2 Network security

Network security is about securing traffic between network nodes, which may belong to the same operator (non-roaming scenario) or to different operators (roaming scenario). It is about securing traffic between different security domains. A network that is managed by a single administrative authority is called a security domain. Sessions where the P-CSCF and the S-CSCF are in different networks involve traffic between different security domains. Traffic entering and leaving a security domain traverses a security gateway, abbreviated as SEG. This means that traffic that is sent from one domain to another traverses two SEGs. Traffic between security domains is protected using Internet Protocol Security Encapsulation Security Payload (IPsec ESP) running in tunnel mode. IPsec is a protocol suite for securing Internet Protocol communications by authenticating and encrypting each IP packet of a communication session. IPsec can be used in protecting data flows between a pair of hosts (host-to-host), between a pair of security gateways (network-to-network), or between a security gateway and a host (network-to-host). The interface between network elements in the same network is called the $Z_1$ interface, and the interface between network elements of different networks is called the $Z_2$ interface, as can be seen in Figure 3.6. $Z_1$ and $Z_2$ interfaces. IPsec provides two protocols to protect data, Encapsulation Security Payload (ESP) and Authentication Header (AH) (Kent & Atkinson, 1998). While AH provides only integrity, ESP provides both integrity and optionally confidentiality. The difference in integrity provided by these two is that ESP protects the contents of the IP packet excluding the IP header, while AH protects both. In IMS the ESP protocol is used, which has two modes of operation, namely the transport mode used between a pair of hosts and the tunnel.
mode used between security gateways to create virtual private networks. ESP in transport mode (figure 3.7) protects the payload of an IP packet, while ESP in tunnel mode (figure 3.7) protects an entire IP packet by encapsulating it into another IP packet. The outer IP packet carries the IP addresses of the security gateways, leaving the inner IP packet untouched.

**TLS usage for Network Security**

The Transport Layer Security (TLS) (Dierks & Allen, 1999) is a security protocol based on the Secure Sockets Layer (SSL) protocol.
3.6.3 IMS Security architecture

As described in the previous two sub-sections, the IMS security mechanisms are divided into two parts: access security, i.e. authentication related mechanisms and traffic protection between the UE and core network, and network domain security, i.e. traffic protection between network elements taking into account roaming and non-roaming scenarios. In the following figure the IMS security architecture is shown, in which the required security associations between the User Equipment (UE) and IMS core network are shown. IMS authentication key and functions are stored on a Universal Integrated Circuit Card (UICC) on the UE side. The IMS Subscriber Identity Module (ISIM) shown in the figure indicates a collection of IMS related security information and UICC functions.

In the figure below five security associations for the IMS security architecture are numbered:

1. This association concerns mutual authentication between the UE and the HSS. Each UE has a user private identity (IP Multimedia Private Identity - IMPI) and one or more user public identity (IP Multimedia Public Identity - IMPU). The pre-shared long-term key in the ISIM and the Authentication Center (AuC) of HSS is associated with the IMPI.

2. After registration has taken place, a security link and corresponding security associations between the UE and the P-CSCF are provided. This interface is called the Gm reference point.

3. The interface between the HSS and both the I-CSCF and S-CSCF provides security associations for the HSS Database, and is called the Cx-interface.

4. This interface provides link security for network elements between different network domains, and is called the interface Za (as described in section 3.1.6), which the Encapsulating Security Payload (ESP) tunnel mode.

5. This interface provides link security for network elements within the same network domain, and is called the Zb interface (as described in section 3.1.6).

![Figure 3.9: IMS Security architecture (3GPP TS 33.203, 2008)](image)

3.7 IMS Security Challenges

The IMS architecture presents significant security challenges that must be addressed by telecom operators implementing IMS, as IMS moves into widespread deployment. This section provides an overview of the IMS security challenges from the network providers or carrier’s perspective and also from the application developer and user’s perspective (Hunter, Clark & Park, 2007). The next section will elaborate on the security concerns for the network providers.
3.7.1 IMS Security Challenges for Network Providers

This sub-section provides a brief overview of the IMS security challenges for the network providers. The concerns from the perspective of the network providers or carriers include:

1. **Toll Fraud**
   There is an inability for carriers to audit or confirm the SIP User Agent (UA) reporting mechanism is working correctly, which causes an increase in the possibility that users may utilize unauthorized services, which will result to both intentionally and unintentionally executed toll fraud. These toll fraud attacks can affect landline, by crossing boundaries without additional charges, and cell phone services, by falsifying the usage minutes of call duration and making any type of data transfer without chance of detection of the core network.

2. **IPv4 vs IPv6**
   The original design of IMS to have IPv6 be the common protocol version on the Internet had to be modified to function in both IPv4 and IPv6, as IPv6 has yet to be largely deployed in the Internet.

3. **NAT and IPSec**
   Network Address (and Port) Translation (NAT) provides a certain level of security by limiting the direct access to the NAT hosts, while allowing internal and outside hosts to connect. However, NATs violate the fundamental semantic of the IP address, which is a globally reachable communication point. The features of IPSec provide overall confidentiality, integrity and authenticity of every packet sent and received. However, IPSec is not fully compatible within a network behind the NAT.

4. **Authentication**
   Username and password authentication is used for devices, which do not have a shared secret key, which is usually stored in the SIM (Subscriber Identity Module). However, the current username and password implemented in IMS is prone to force and replay attacks.

5. **Gateway Attacks**
   When converting data to different media, integrity checks should be performed to verify if content converted is the same content as before in different format, and resulting data is still considered not malicious. However, an attacker may perform an inverse conversion from a malicious script, which may look not malicious before conversion, to something that may harm the network after conversion.

6. **Denial of Service**
   The IMS faces the threat of Denial of Service attacks from several sources, while having a greater exposure to these attacks compared to any prior telecommunication infrastructure. One threat is that posed by its Internet connectivity, where a determined attacker with sufficient resources can cause at least temporarily disruption to any Internet host.

7. **Network Topology**
   Many carriers prefer that the network capabilities and structure of their service to be kept proprietary and confidential. However, there are several ways to discover bits and pieces of information from closely examining the packets, like when observing via the route, record-route
or path headers of the SIP packets, the number of CSCF servers in the network and how the packet is being routed can be revealed.

The following sub-section describes the IMS security concerns for the network users.

### 3.7.2 IMS Security Challenges for Network Users

This sub-section provides a brief overview of the IMS security challenges for the network providers. The concerns from the perspective of the network users or application developers are the following:

1. **Denial of Service**
   The quality of the service guarantees for an IMS terminal to receive content is one of the most fundamental guarantees IMS provides. While the IMS cannot account for so called “layer 1” problems, like a sudden loss of signal or RF interference, it is still the IMS’s responsibility to make sure the provisioned bandwidth is made available to the IMS terminal. A malicious entity would constitute a DoS by seeking to make some or all of this bandwidth unavailable to the legitimate IMS terminal.

2. **User Agent Applications**
   The essential role of IMS is to provide safe applications content in the IMS terminal. One important goal of the IMS is to enable carriers to act as providers of third-party application content. However, the security threat to the IM terminal posed by malicious content poses serious risks to IMS terminals and user assets.

3. **Identity and Presence Considerations**
   The security of personal data is the user’s principal concern. However, a threat to a user is posed by the potential of an attacker to pretend to be someone else and be accepted by the IMS. Users are then able to create multiple different public identities, e.g. business vs. personal, which are tied to a single private identity, what is considered falsified identity.

4. **Personal Data and Privacy**
   More and more data are being trusted by users to the digital technology. One area of privacy concern is raised by location data, where the Global Positioning System (GPS) allows users to know, and potentially share their location information. Another area of concern for the user is that of voice and data privacy, namely encryption, where encryption becomes pretty difficult to implement, due to the current lack of IPv6, or because the IMS terminal does not have enough CPU or battery to support voice encryption.

This chapter provided an introduction of the basic concepts regarding the IP Multimedia Subsystem architectural framework. Based on this literature a basic IMS infrastructure and IMS-based messaging service was designed and implemented. Chapter 5 provides an overview of the design and implementation.
Chapter 4
Cloud computing

This chapter provides an introduction to Cloud computing starting with the emergence of the cloud-computing paradigm in section 4.1 and followed by the cloud computing definition in section 4.2. Section 4.3 compares the two concepts of cloud computing and grid computing. In section 4.4 the cloud computing service models and deployment models are described. After that the cloud computing characteristics and benefits are given in section 4.5. The cloud computing security challenges are elaborated in section 4.6. In section 4.7 the main cloud-computing providers, including both commercial and open source solutions, are listed and described.

4.1 Cloud computing emergence

Electricity, water, gas and telephony are the four essential utility services that everyone need access to. These utilities need to be accessible anytime so that consumers can access them at any time. The vision exists that computing will be the fifth essential utility after the four just mentioned utilities, and will provide the essential basic computing services to the public to meet its everyday needs (Buyya et al., 2009). The computing utility vision foresees the transformation of the computer industry in the 21st century, enabling computer services on demand to the general public, i.e. consumers pay computing service providers only when they access these services. Consumers do no longer need to invest in building their own complex IT infrastructures and maintain it. This computing utility vision is achieved thanks to the Internet, forming a worldwide system of computer networks where computers in one location can communicate with other computers in other locations in the world. One of the computer paradigms that emerged in recent years is Cloud computing.

4.2 Cloud computing defined

Cloud computing is associated with a new paradigm for the provision of computing infrastructure, shifting the location of computing infrastructure to the network to reduce the costs associated with the management of hardware and software resources (Hayes, 2008). It is considered the long-held dream of computing as a utility, which has the potential to transform a large part of the IT industry, making software even more attractive as a service and shaping the way IT hardware is designed and purchased (Armbrust et al., 2010). Cloud computing opens new ways for small- to medium sized companies, such as software development companies to provide innovative Internet services without the need to invest in hardware to deploy their new services, or human expense to operate and maintain it. Also required capacity can be increased and decreased dynamically without the need to invest in new infrastructure that may be sitting idle when not needed, i.e. over-provisioning. Also cloud computing eliminates the concern of under-provisioning, where too little resources are available to provide services that suddenly become very popular, and as a result missing potential customers and revenue.

Using cloud computing users make requests for services instead of requests to specific end-hosts as was the case with older computing models, making the physical location of servers and their arrangement structure unknown to the end-user (Danielson, 2008).
The cloud computing industry represents a large ecosystem of many models, vendors and market niches, and the following definition given by the National Institute of Standards and Technology (NIST) attempts to encompass all of the various cloud approaches (Mell & Grance, 2011):

“Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”

According to Vaquero et al. (2009) the following cloud definition can be extracted from more than 20 different definitions from different sources (Geelan, 2009):

“Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure provider by means of customized SLAs.”

The three core technologies, i.e. virtualization, multi-tenancy and Web services (Grossman, 2009), that enable the provision of on-demand computing resources and services over the Internet are described as follows:

- Virtualization is the technology that hides the physical characteristics of a computing platform from the users, presenting an abstract, emulated computing platform, which behaves like an independent system (Vouk, 2008). The emulated computing platform can be configured on demand, maintained and replicated very easily. Virtual machines simulate a complete physical computer, enabling the virtual machine to run any software, from operating systems to end-user applications (Naone, 2010). Virtualization makes it possible that virtual machines from multiple customers can share the same physical server. It has transformed the thinking from physical to logical, treating IT resources as logical resources rather than separate physical resources (Desai et al., 2013). Virtualization is used to provide the essential cloud characteristics of location independence, resource pooling, and rapid elasticity. To provide virtualization, the “hypervisor” became the fundamental software, which is also known as Virtual Machine Monitor (VMM). The hypervisor is briefly described in section 4.3.

- Multi-tenancy refers to a single instance of an application that serves multiple clients, allowing better utilization of resources of a system, such as memory and processing time. An application virtually partitions its data and configuration, allowing each client or organization to work with its own customized virtual application.

- The World Wide Web Consortium (W3C) defines a Web service as “a software system designed to support interoperable machine-to-machine interaction over a network” (Haas & Brown, 2004). The term web-services refers here to the communication between clients and servers over the HTTP protocol, which is used on the Web, standardizing interfaces between applications, making it easier for a software client to access server applications over a network.

The actors involved in the deployment of clouds include Service Providers, Service Users and Infrastructure Providers (Vaquero et al., 2009). Service Providers make services accessible to Service Users through Internet-based interfaces. The purpose of cloud computing is to outsource the provision of computing infrastructure required to host services. Infrastructure Providers offer this infrastructure “as a service”, this way moving computing resources from the Service providers to the
Infrastructure Providers, so Service providers gain can gain in flexibility and reduce costs. Figure 4.1 shows the moving of computer resources from the Service Providers to Infrastructure Providers. Service providers have access to the infrastructure provided by Infrastructure Providers through the Infrastructure Interface and can manage services that are deployed on this infrastructure, while Service Users access the deployed services.

Figure 4.1: Cloud actors

In the following section a comparison is made between three related computing paradigms, namely cluster, grid and cloud computing.

4.3 Hypervisor

The hypervisor, also known as the Virtual Machine Monitor (VMM), runs on a host computer and is responsible for creating and running the virtual machines (VMs) on this host machine (Zissis & Lekkas, 2011). Its main responsibility is to monitor the VMs that are running on top of it. According to Desai et al. (2013), “Hypervisor is a thin software layer that provides abstraction of hardware to the operating system by allowing multiple operating systems or multiple instances of the same operating system, termed as guests, to run on a host computer.” Hypervisors can be classified in two types, namely the Type I Hypervisor and the Type II Hypervisor. The Type I Hypervisor is known as the “native hypervisor” or “bare metal hypervisor” and runs directly on top of the underlying hardware. This is illustrated in the figure 4.2. The hypervisor is in this case a small code, which is responsible for the scheduling and allocation of system resources to VMs as there is no operating system running below it. The VMM provides device drivers that guest OS uses to directly access the underlying hardware (Galvin, 2009). Examples of the Type I Hypervisor include VMware ESX and Xen. The Type II Hypervisor is shown in figure 4.3 and is known as the “hosted hypervisor”, which runs as an application in a normal operating system, which is known as “host operating system”. The Host OS does not have any knowledge about the Type II Hypervisor and treats it as any other process. It typically performs I/O on behalf of the Guest OS (Galvin, 2009).
4.4 Cluster, Grid and Cloud computing comparison

The comparison between a cluster, a grid and a cloud is important, yet difficult due to the great divergence in definitions of these concepts, to better understand what the cloud-computing paradigm is.

According to Pfister (1998) and Buyya (1999) a cluster is defined as follows:

“A cluster is a type of parallel and distributed system, which consists of a collection of inter-connected stand-alone computers working together as a single integrated computing resource.”

According to Buyya (1999), a grid is defined as follows:

“A Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed ‘autonomous’ resources dynamically at runtime depending on their availability, capability, performance, cost, and users’ quality-of-service requirements.”

Autonomous resources include supercomputers, storage systems, data sources, and specialized devices owned by different organizations for solving large-scale resource-intensive problems in science, engineering and commerce (Buyya et al., 2009).

According to Foster (2002), a Grid is “a system that coordinates resources which are not subject to centralized control, using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service”.

More recent definitions of a grid emphasize the ability to combine resources from different organizations for a common goal (Bote-Lorenzo et al., 2004).

According to Buyya et al. (2009), a cloud is defined as follows:

“A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers.”

In Buyya et al. (2009), it is stated that clouds are not a combination of clusters and grids, but clouds are next-generation data centers consisting of nodes that are created as virtual machines using a hypervisor application, which was defined in section 4.1.2. The virtual machines are dynamically provided on demand as a personalized resource collection to meet specific service-level agreements that are established through negotiation.

In Appendix A the characteristics of Cluster, Grid and Cloud computing are listed from which the
following relevant characteristics are compared:

- **Distribution**
The resources in a cluster are located in a single administrative domain, the same local area network (LAN), and managed by a single entity, while with Cloud and Grid systems, resources are geographically distributed across multiple administrative domains with their own management policies and goals, i.e. a cluster is tightly coupled, while a grid and a cloud are loosely coupled.

- **Ownership**
A cluster is owned by a single party in a single location, while a grid is owned by multiple parties in multiple locations and connected together so that users can share the combined power of resources, and a cloud is owned by a single party in multiple locations. Also an important difference between cloud computing and other distributed systems is that the ownership and responsibility of the assets are handled by someone other than the end user.

- **Node operating system (OS)**
The operating systems used in a cluster is one of the standard OSs, while with a grid it is any of the standard OSs, but more dominated by UNIX, and in a cloud a hypervisor is used that creates and runs virtual machines on which multiple OSs can run.

- **Capacity**
The capacity of a cluster is stable and guaranteed, while the capacity of a grid is high but it may vary, and in a cloud the capacity is dynamically allocated based on demand.

- **Potential for building 3rd party or value-added solutions**
This characteristic is limited in a cluster due to its inflexible architecture. In a grid it is also limited due to its strong orientation for scientific computing, while in a cloud this characteristic has high potential. New services can be created by dynamically provisioning of compute, storage and application services.

In the following section the services and deployment types of Cloud computing are presented.

### 4.5 Cloud computing models

There exist several scenarios where Clouds are used, which are known as the Cloud computing models or service models. The most common models include Infrastructure as a Service, Platform as a Service and Software as a Service. In addition to these three service models, there are also the Data-Storage as a Service (DaaS) and Communication as a Service (CaaS) service models (Rambhadjan & Schutijser, 2010). These major models can be offered in any of the deployment models of Cloud computing, namely the private, public, community and hybrid cloud.

#### 4.5.1 Service models

The following three most common service models are described according to Joyent (2013), Mell & Grance (2011), Zissis & Lekkas (2011) and Clavister (2009):

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

In the Infrastructure as a Service model, providers offer resources, often virtual machines where consumers can setup a complete platform virtualization environment as a service, i.e. consumers can
provision processing, storage, networks, and other fundamental computing resources as a service. The consumer can deploy and run software, including operating systems and applications. Through the virtualization technology consumers can split, assign and dynamically resize resources to build ad-hoc systems designed for customers. The consumer does not manage or has no control on the underlying cloud infrastructure, but has control over operating systems, storage, deployed applications and sometimes limited control over select networking components, such as a host firewalls. Many companies make use of this model as a means to expand their current computing capabilities without drastically increasing capital outlays on new hardware and software. Some examples of IaaS include Amazon’s S3 storage service and EC2 computing platform, Rackspace Cloud Servers, Joyent and Terremark.

Cloud Platform as a Service (PaaS)

In this model the Platform as a Service provider provides developers with the required facilities to support the complete lifecycle of building and delivering web applications and services. The programming languages and tools required to make this possible are provided to developers, enabling them to deploy their custom-made applications onto the cloud infrastructure. Instead of supplying a complete virtualized infrastructure to consumers, the software platform where systems run on are provided. The consumer is not responsible for the management or control of the underlying infrastructure including servers, network, operating systems, storage or application containers, but manages and controls the deployed applications and possible environment configurations for hosting the applications. The resources required to equal the application demand scale automatically, without requiring the consumer to allocate resources manually. PaaS is useful for sharing development efforts between partner companies or divisions, and also when development teams are widespread geographically. PaaS enables engineers to share and backup a central repository of application data and also implement better version control and environment variables. Some examples of PaaS include Amazon’s Relational Database Services (RDS), Rackspace Cloud Sites, Google App Engine, Microsoft’s Azure Services Platform and Salesforce’s Force.com.

Cloud Software as a Service (SaaS)

Consumers of the SaaS model are provided with a wide variety of sophisticated applications that run on the provider’s cloud infrastructure, using cloud clients such as web browsers. The provider licenses the application to the consumer, typically a monthly or yearly flat fee per user, for use as a service on demand that runs in the cloud, without the need for the consumer to install and run the application on the consumer’s computer. Such enterprise SaaS application examples include product lifecycle management or supply chain management. A typical example of a SaaS is the online alternatives of typical offline applications, such as word processors. This SaaS model may save companies the expense of buying hardware, software and maintenance, and may have drawbacks such as the inability to customize the solution for specific business requirements, complexities in integrating existing business IT infrastructure with the SaaS application and difficulties in predicting and budgeting pay-for-use pricing. The consumer does not control or manage the underlying cloud infrastructure, including servers, network, operating systems, storage or applications, but the consumer may have access to limited user-specific application configuration settings. Some enterprise-level SaaS providers include Salesforce, Netsuite and Google Apps and some personal applications include Gmail, Facebook and Twitter.

4.5.2 Deployment models

The following four deployment models have been identified according to Mell & Grance (2011).
**Private cloud**
A private cloud is cloud infrastructure used exclusively by a single consumer, i.e. a single organization, which may encompass multiple business units. The cloud infrastructure is owned, managed and operated internally by the organization, a third-party or a combination of these, and hosted internally or externally. When creating a private cloud the virtualization of the business environment requires a significant level and degree of engagement, and requires organizations to also reevaluate decisions about existing resources. This cloud model is the most expensive proposition because the organization must purchase, house and maintain all the hardware and software for the cloud.

**Public cloud**
The public cloud is cloud infrastructure owned by an organization that sells cloud services, and can be accessed by the general public or large industry groups. This cloud is the least expensive but for certain types of business computing it is less desirable due to the lack of control ad resource sharing in public clouds.

**Community cloud**
In the community cloud model several organizations from a specific community share the cloud infrastructure. These organizations have common concerns such as mission, security requirements, compliance and jurisdiction. The cloud infrastructure is owned, managed and operated by one or more of the organizations in the community, a third party or a combination of them, and hosted internally or externally. The costs of this cloud are spread among fewer users than the public cloud, but more users than the private cloud.

**Hybrid cloud**
The hybrid cloud consists of a combination of two or more clouds, either private, community or public clouds, where each cloud remains a unique entity but the clouds are bound together by standardized or proprietary technology enabling data and application portability. Two closely related examples of hybrid cloud include cloud bursting and cloud balancing. These concepts are described as follows.

- **Cloud-bursting**
The cloud-bursting concept combines the resources currently used internally by an organization with additional resources from a cloud provider to maintain acceptable response time during workload peaks (Agrawal, 2011). It is defined as the practice of ‘bursting’ into a cloud when capacity has been reached in the corporate cloud/data center (Macvittie, 2009). This model is particularly attractive for organizations that require high performance and/or data-intensive computing. The cloud-bursting model allows organizations to maintain an optimal availability of their services using additional resources at certain seasonal or event-based peaks of traffic when the capacity of the internal infrastructure is reached. Because these peaks only occur at certain moments or during certain periods, investment in additional resources, that would sit idle otherwise, is thus not justified. Organizations can continue to invest in internal hardware and/or infrastructure in order to provide the best performance, or for security needs or want more control over particular resources in the internal infrastructure. Organizations may own sensitive applications that may process sensitive data, and thus require considerable security. These applications can be managed in the internal infrastructure while the more common ones can be provisioned using additional resources from cloud providers.

Cloud bursting uses the capacity variable to determine when to use which cloud (internal or external). When the capacity of the internal infrastructure is reached, i.e. when the resources are depleted, requests are redirected to an external cloud. The closely related concept is cloud balancing which
uses the location variable to determine where to route requests, based on technical or business metrics.

- **Cloud Balancing**

In this model an organization’s applications reside in multiple clouds and are always accessible (MacVittie, 2010). Application requests are thus routed across multiple clouds by applying specified technical or business metrics or service level agreements (SLAs) to every request (Triebes, 2010). It differs from cloud bursting in the sense that applications are already deployed and running in multiple clouds (both internal and external), while with cloud bursting an application is deployed and/or launched at the moment the capacity of the internal infrastructure is reached and resources are depleted. It may be the case with cloud balancing that the majority of application users are served by applications deployed in external cloud providers’ environments, while the local application, if there is one, has enough capacity to serve these users.

Now that the different models in which a cloud can be deployed are described, its characteristics and benefits are listed and explained, to give better understanding in the need for a cloud architecture.

## 4.6 Characteristics and benefits

### 4.6.1 Characteristics

The following list gives an overview of key characteristics of cloud computing according to Mell & Grance (2011) and Zissis & Lekkas (2011):

- **On-demand self-service**
  
  Users can rapidly access computing resources, including storage, processing power, memory and network bandwidth, as required automatically without human interaction with the service provider.

- **Rapid elasticity**
  
  Resources can be easily provisioned and released in any quantity at any time, mostly automatically, to rapidly scale out or scale in based on demand.

- **Scalability of infrastructure**
  
  The same way physical servers are added to and dropped from a network, this same way nodes can be added to or dropped from the network, with limited modifications to infrastructure set up and software.

- **Broad network access**
  
  Using different types of platforms, such as mobile phones, tablets, laptops etc., capabilities over the network can be accessed.

- **Location independence**
  
  Customers making use of cloud services have no knowledge and no control over the exact location of the provided resources.

- **Reliability**
  
  Cloud computing guarantees reliability, through the use of multiple redundant sites, this way making cloud computing suitable for business continuity and disaster recovery.

- **Economies of scale and cost effectiveness**
  
  To take advantage of economies of scale cloud are implemented as large as possible, regardless of the deployment model. To lower costs large cloud deployments are often located close to cheap power stations and in low-priced real estate.

- **Sustainability**
Sustainability is accomplished through improved resource utilization, more efficient systems, and carbon neutrality.

- **Resource pooling**
  The provider’s computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.

- **Measured Service**
  Cloud computing has the characteristic of being able to monitor, and automatically control and optimize resource usage by making use of measurements at some level of abstraction appropriate to the type of service, such as storage, processing, bandwidth, and active user accounts.

### 4.6.2 Benefits

According to Marston et al. (2011), Agrawal (2011) and Clavister (2009), cloud computing has several benefits. Some of these benefits are described as follows:

- Cloud computing enables small- and medium sized businesses to benefit from compute-intensive applications that require large amounts of computing power that were previously only available to and affordable by large organizations or enterprises.
- Cloud computing leads to faster time-to-market of services, due to the almost immediate access to hardware resources and no up-front capital investments.
- Enterprises are able to scale their services based on client demand.
- Particular computers or devices are no longer needed to use an application.
- Cloud computing has shifted the location of the computer infrastructure to the network, enabling consumers to access information wherever they are, rather than having to remain at a specific location.
- The need to upgrade an application on a local computer is eliminated, as applications accessed on the cloud are always the latest version.
- Users do not need to wait long for the computer to start up, as applications in the cloud are always on.
- It also makes possible that new types of applications can be deployed making the delivery of new kinds of services that were not possible before, such as mobile interactive applications, parallel batch processing etc.
- Cloud computing offers more flexibility than past computing methods, enabling e.g. organizations to outsource their whole infrastructure or just segments of it.

### 4.7 Cloud computing security challenges

The shift to cloud technologies is fast and affordable, and it has become very popular at an exceptional pace. But this shift brings new security challenges for important business-level security policies, processes and best practices. The Cloud Security Alliance (CSA) is a non-profit organization that promotes the use of best practices for providing security assurance within cloud computing and to provide education on the uses of cloud computing to help secure all other forms of computing (Cloud Security Alliance, 2014). The CSA deals with the creation of industry-wide standards for effective cloud security. To help organizations make risk-management decisions concerning cloud adoption strategies, the CSA identified the most current nine critical threats to cloud security, this
way letting organizations better understand the nature of cloud security threats (Cloud Security Alliance, 2013). These nine threats include:

1. **Data breaches**
   Cloud computing introduces significant new avenues of attack risking an organization’s sensitive internal data to fall into the hands of its competitors, i.e. data loss or data leakage.

2. **Data loss**
   Consumers and businesses risk losing their data stored in the cloud, including sensitive data, due to not only malicious attackers but also due to any accidental deletion by the cloud service provider, or a physical catastrophe such as a fire or earthquake, or even when the consumer encrypts its data and loses the encryption key.

3. **Account or service traffic hijacking**
   Attackers make use of attack methods such as phishing, fraud, and exploitation of software vulnerabilities, to steal cloud provider’s credentials, this way eavesdrop on its activities and transactions, manipulate its data, return falsified information, and redirect its clients to illegitimate sites. With these stolen credentials, attackers may access critical areas of deployed cloud computing services, allowing them to compromise the confidentiality, integrity and availability of those services.

4. **Insecure interfaces and APIs**
   To interact with cloud services organizations use interfaces and APIs provided by the cloud provider. Management, monitoring, provisioning and orchestration are performed using these interfaces and APIs. Security and availability of cloud services is dependent upon the security of these APIs, leading organizations to face serious security issues related to confidentiality, integrity, availability and accountability when the provided APIs and interfaces are weak.

5. **Denial of Service**
   Denial-of-service (DoS) attacks prevent users of cloud services from being able to access their data or their applications. Attackers cause unbearable system slowdowns by letting the cloud service consume excessive amounts of limited system resources, including processing power, memory, disk space and network bandwidth, leaving service users angry and confused about the not responding service.

6. **Malicious insiders**
   According to Computer Emergency Response Team (2014) “a malicious insider threat to an organization is a current or former employee, contractor, or other business partner who has or had authorized access to an organization’s network, system, or data and intentionally exceeded or misused that access in a manner that negatively affected the confidentiality, integrity, or availability of the organization’s information or information systems.” An example is a system administrator that has access to sensitive information in an incorrectly designed cloud. In an IaaS the malicious insider would have access to more critical systems and data.

7. **Abuse of cloud services**
   Companies rent time on thousands of servers from a cloud-computing provider, instead of purchasing and maintaining their own thousand of servers. This is more affordable for companies, but for the cloud provider this may lead to attackers using its infrastructure to carry out attacks, such as cracking an encryption key that may take years on the attacker’s own limited hardware,
staging a DDoS attack, or to serve malware or distribute pirated content. The challenge for the cloud provider would be how to detect people abusing its service, how would it define abuse and how would it prevent abusers from doing it.

(8) **Insufficient due diligence**

Many times enterprises or organizations jump into the cloud without a complete understanding of the cloud service provider’s environment, which may lead to significant risk that the organization may not even understand. Enterprises or organizations moving to the cloud must have the required resources and perform broad cloud service provider due-diligence to understand the risks it undertakes when adopting the new cloud technology.

(9) **Shared technology vulnerabilities**

A single vulnerability or misconfiguration in any of the cloud service models, either IaaS, PaaS or SaaS, when sharing infrastructure, platforms and applications, can lead to a compromise across the entire provider’s cloud.

These threats define new challenges for organizations to consider when implementing their own private cloud, or when considering in using services from a cloud provider, or the combination of both.

### 4.8 Main Cloud computing providers

#### 4.8.1 Commercial cloud computing platforms

This section will discuss the 10 most important commercial cloud-computing platforms in 2013 according to Bort (2013). These include (1) Amazon AWS, (2) VMWare, (3) Microsoft, (4) Salesforce.com, (5) Google Cloud Platform, (6) Rackspace, (7) IBM, (8) Citrix Systems, (9) Joyent, and (10) SoftLayer. Computing platforms used for commercial cloud maintain a proprietary infrastructure with open interfaces.

(1) **Amazon AWS**

Amazon Web Services (AWS) started in 2006 offering IT infrastructure services to businesses in the form of web services, known as cloud computing. Amazon is nowadays a comprehensive, evolving cloud-computing platform provided by Amazon.com, providing a highly reliable, scalable, low-cost, easy-to-use and secure infrastructure platform in the cloud, including a range of cloud-based data storage options, providing architects and engineers with a diversity of options (Amazon Web Services, 2014). Amazon is one of the pioneers in cloud computing and one of the firsts in offering pay-as-you-go services, accessing virtual servers and data storage space. A couple of services offered by Amazon include:

(1) Amazon Elastic Compute Cloud (EC2), for resizable computing capacity in the AWS Cloud,
(2) Amazon Elastic Load Balancing, which is a load balancing solution provided by Amazon to distribute incoming application traffic across multiple Amazon EC2 instances, achieving greater levels of fault tolerance.
(3) Amazon Auto Scaling, which is a web service designed to launch and terminate EC2 instances automatically based on user-defined criteria,
(4) Amazon Simple Storage Service (S3), for on-demand storage capacity,
(5) Amazon Simple Database Service (SimpleDB), which is a database webservice for highly-available and flexible non-relational data storage,
(6) Amazon CloudFront, which is a content delivery web service that integrates with other Amazon web services to enable the easy and fast distribution of content to end users and
(7) Amazon Simple Queue Service (SQS), which is a scalable hosted queue messaging service on the AWS Cloud that enables the storage of messages as they travel between nodes (computers).

(2) VMware
VMware is a virtualization and cloud computing software provider for computers that are x86-compatible. VMware virtualization is based on the ESX/ESXi bare metal hypervisor, which supports virtual machines. The term "VMware" is often used in reference to specific VMware Inc. products such as VMware Workstation, VMware Horizon Application Manager, VMware View, and VMware vCloud Director.
VM stands for "Virtual Machine" (not to be confused with the broader term virtual machine), and is a widely-installed operating system for computers and servers that are IBM-compatible, which can host other operating systems in such a way that each operating system behaves as if it was installed on a self-contained computer with its own set of programs and hardware resources (Rouse, 2013).

(3) Microsoft
Windows Azure is provided by Microsoft as the cloud operating system PaaS, which stands for Platform as a Service. Microsoft is a cloud computing or cloud services operating system for the service hosting development and service management environment. It helps developers to build, scale and host web applications through Microsoft datacenters. Developers can use existing skills with the .Net framework and Visual Studio to build applications, and also deploy and run web applications written with non-Microsoft programming languages like Java, PHP or Ruby. Therefore, there is no need to purchase servers or storage, manage network equipment.

The Azure Services Platform is being developed to run on Windows Azure operating systems and therefore enables client organizations to have access to several online services like .Net, Live, SharePoint, SQL, and Microsoft Dynamic CRM. Microsoft does have an impact among enterprise developers, while already providing cloud services to Web and mobile companies in both the social and technological networking (SearchCloudComputing, 2011).

(4) Salesforce.com
SalesForce.com is the first well-known and successful SaaS application, which means Software as a Service, and which is merging into a well-defined and mature well-defined space. The company introduced an integrated set of tools and application services called Force.com, and which is used by corporate IT departments and independent software vendors to build any business application and run it on the same infrastructure that delivers the SalesForce Customer Relationship Management applications. SaleForce.com recently forced its way into the service market platform by purchasing Heroku, which is a cloud application platform for Ruby, which is a new way of building and deploying web applications (Marston et al., 2011).

(5) Google Cloud Platform
Google has a cloud-based platform, which can be accessed by organizations using Google App Engine. This provides tools to build and host web applications. Google takes part in the Platform-as-a-Service (PaaS) market with Google App Engine, and also offers SaaS (Software as a Service) with Google Apps, which is a set of online office productivity tools including calendaring, e-mail, word processing and a simple website creation tool. Google is pressuring the industry to move from packaged software to Web-hosted services using Google Apps. Postini has been acquired by Google since 2007 and offers a set of e-mail, web security and archiving services, to enter the area of electronic corporate communications (Marston et al., 2011). Google however still has to make an impact among enterprise developers (SearchCloudComputing, 2011).
(6) **Rackspace**

Rackspace is the global leader in hybrid cloud and founder of the open-source cloud operating system, OpenStack. Rackspace is seen by many customers for delivering the best-fit infrastructure for their IT needs, leveraging a product portfolio which allows workloads to run where they perform best, whether on the private or public cloud, dedicated servers, or a combination of platforms. The company’s Fanatical Support helps clients successfully architect, deploy, and run their most important applications (Rackspace, 2014).

(7) **IBM**

IBM provides their cloud-computing services offering companies, Blue Cloud, access to tools that allow them to manage large-scale applications and databases via IBM’s Cloud (Marston et al., 2011). Besides, IBM helps companies integrate their infrastructure into the cloud by offering consulting services. IBM together with Google works with several universities to promote new software development techniques, which help researchers and students address the challenges of future cloud applications. IBM has also launched recently an Infrastructure Service, which is called the Smart Business Test and Development Cloud. This is initially used just for test and development purposes (SearchCloudComputing, 2011).

(8) **Citrix Systems**

Citrix Systems, Inc., as a global leader in application server software and related services, offers innovative organizations the opportunity to run any device application over any connection. Its products give customers the facility to use a network of computers connected to a central server, where the software resides, while solving a number of network problems for companies. It also facilitates "thin" client/server computing, which implies that a low-powered computer can be made to work as well as a "fat," high-powered, large memory machine with a cutting-edge processor, as the software is at a remote location. The company’s main products are as follows:

- Citrix ICA (Independent Computing Architecture), which is a licensed technology that improves server computer processing;
- MultiWin, which is a process that allows more than one simultaneous user to connect to the same centralized server software in discrete sessions;
- MetaFrame, which is a server software;
- Nfuse and XPS, which are portal software systems (Reference for Business, 2014).

Citrix makes cloud software and created the open source cloud computing solution CloudStack, which is described in the previous section. Citrix solutions provide a more people-oriented, on-demand computing environment by IT. This cloud is concentrated in the basis for business innovation and growth which results in people flexibility and freedom, and IT security and control (Citrix, 2014).

(9) **Joyent**

Joyent has developed Joyent Smart Technologies with the intention to eliminate the complexities associated with managing hardware and software in cloud environments, while returning the focus to software innovation and business productivity. With the use of Joyent Smart Technologies, complex capacity planning decisions are dramatically reduced, like the required number of virtual machines, network bandwidth, CPUs, or storage capacity. Businesses can focus on improving applications with sizing and capacity issues minimized. Joyent Smart Technologies promotes innovative software development by eliminating many complexities in the environment. Software developers can therefore deliver more quickly stable applications. Joyent’s flexibility is expressed in the range of available customers’ deployment options. Businesses can use pre-configured and preinstalled software packages from a growing list of third-party software partners, choose for Linux / Unix
operating systems from existing software packages, or build entirely new applications with languages like PHP, Java, Ruby, Python, Rails, and more. Furthermore, if public cloud hosting is not acceptable, companies can license the deployment of Joyent Smart Technologies infrastructure at their site. The Joyent virtual private cloud is another option, which offers a completely dedicated private cloud infrastructure which is hosted at a Joyent data center (Andrew, 2010).

(10) SoftLayer
SoftLayer Technologies, Inc. is the largest privately held cloud computing infrastructure provider in the world. IBM acquired this company in the year 2013, and will strengthen hereby IBM’s leadership position in cloud computing, while also helping speed business adoption of public and private cloud solutions.

IBM acquired SoftLayer to make it easier and faster for customers around the world to incorporate cloud computing by marrying the simplicity and speed of SoftLayer’s public cloud services with the enterprise grade reliability, openness and security of the IBM SmartCloud portfolio. SoftLayer accelerates IBM’s ability to integrate private and public clouds for its customers, with flexibility that provides deployment options that will bring with it a faster, broader transformation for small, medium and large businesses with a range of security and performance security models (StreetInsider.com, 2013).

The SoftLayer’s network is actually three different and redundant gigabit network architectures, namely; public, private, and data center-to-data center. This design provides maximum accessibility, control, and security for the IT infrastructure. SoftLayer provides the following different interface services:

- API (Application Programming Interface): This provides system-to-system access and supports the use of both SOAP and XML-RPC interfaces and as an open, standards-based platform, it can fully integrate with any third-party or custom application.
- Customer Portal: This is the point-and-click Web-based solution for SoftLayer account management. There are various categories within this interface that consist of numerous related tools and features, from user details and billing to device management and load balancing.
- SoftLayer Mobile: Apps have been created available for Android OS, iOS, and Windows Phone devices so the most important account details accounts are accessible from a phone or tablet.
- Command Line Interface (CLI): This is built on SoftLayer’s API Python Client and it enables users in more technical and operations related roles to interact with accounts, products and services in any operating system command line (SoftLayer, 2014).

4.8.2 Open source solutions
This section will give an overview of the open source solutions, namely: (1) OpenNebula, (2) OpenStack, (3) CloudStack, and (4) Eucalyptus.

(1) OpenNebula:
OpenNebula is an open-source toolkit which is used to build private, public and hybrid clouds and which provides the most simple but feature-rich and flexible solution for the comprehensive management of virtualized data centers to enable on-premise IaaS clouds (OpenNebula.org, 2014). This toolkit is used to manage clouds while providing integration capabilities for internal clouds.

OpenNebula can be used to support in the following three clouds:
- Private Cloud: A platform to manage virtualized infrastructures in the cluster or data center.
- Hybrid Cloud: A combination of local infrastructure with public cloud-based infrastructure, which enables highly scalable hosting environments.
- Public Clouds: Cloud interfaces provided to expose its virtual machine’s functionality, storage and network management.

OpenNebula was designed to concentrate on the requirements of business use cases from leading companies to multiple industries, like e.g. Hosting, Telecom, eGovernment, Utility Computing. The principles that were used to design of OpenNebula are:
- Lightness: High efficiency;
- Openness: Architecture, interfaces, and code;
- Flexibility: To fit into any datacenter;
- Interoperability and portability: To prevent vendor lock-in;
- Scalability: Large scale infrastructures;
- Stability: Use in production enterprise-class environments;
- SysAdmin-centrism: Complete control over the cloud; and
- Simplicity: Easy to deploy, operate and use.

The objective of this open-source cloud-computing project is to build an industry standard tool, which manages the complexity, and heterogeneity of distributed data center infrastructures. Organizations can use it as a virtualization tool for the management of virtual infrastructures in their own private cloud, while it also fits into existing data center environments to build any kind of IaaS cloud deployment. Besides, this toolkit combines local infrastructures with public cloud-based infrastructure by allowing highly scalable hosting environments, and public clouds. Furthermore, OpenNebula coordinates storage, virtualization, network, monitoring, and security technologies, and which enables the dynamic group placements of interconnected virtual machines on distributed infrastructures. This is done by combining both remote cloud resources and data center resources in accordance with the allocation policies. OpenNebula interoperability does leverage existing IT assets, protect investments and avoid vendor lock-in, and makes hereby of cloud an evolution.

(2) OpenStack:
Cloud computing technologists, producing the universal open source cloud computing platform for both private and public clouds, form together with global collaboration of developers the OpenStack cloud (OpenStack, 2014). This project’s objective is to deliver solutions for all types of clouds by having key features like being simple to implement, massively scalable, and feature rich. This technology consists of a series of interrelated projects, which deliver different components for a cloud infrastructure solution.

Stakeholders of OpenStack are mainly corporations, service providers, VARS, SMBs, researchers, and global data centers, who are looking to deploy large-scale cloud deployments for public or private clouds leveraging the support and resulting technology of a global open source community (OpenStack, 2014).

(3) CloudStack:
Apache CloudStack is open-source software, which has been designed to deploy and manage large networks of virtual machines, as a highly available and scalable Infrastructure as a Service (IaaS) cloud-computing platform (Apache Cloudstack, 2014). A number of service providers do use CloudStack to offer public cloud services. Many companies also use CloudStack to provide an on-premises (private) cloud offering, or as part of a hybrid cloud solution.

CloudStack is a turnkey solution, which includes the complete "stack" of features the majority of the organizations would like to have with an IaaS cloud. These features are: compute orchestration, a full
and open native API, Network-as-a-Service, resource accounting, user and account management, and a first-class User Interface (UI). The most popular hypervisors are supported by CloudStack, namely: VMware, XenServer, KVM, and Xen Cloud Platform (XCP). Users can manage their cloud using a Web interface, command line tools, and/or a full-featured RESTful API. In addition, API is provided by CloudStack which is compatible with AWS EC2 and S3 for organizations that wish to deploy hybrid clouds.

(4) Eucalyptus:
Eucalyptus is an open-source cloud-computing framework which uses computational and storage infrastructure usually available to academic research groups to provide a platform that is open and modular to study and experimental instrumentation (Nurmi et al., 2009).

Eucalyptus was initially developed as part of a Computer Science’s academic research project at the University of California, Santa Barbara (Eucalyptus Systems, 2014). The Ubuntu Enterprise Cloud (UEC) private cloud solution is powered by the open source engine. Eucalyptus means ‘Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems’ and is becoming a standard for on-premise cloud computing. This means that it allows researchers and administrators to deploy an infrastructure for user-controlled virtual machine creation with the use of existing resources.

Eucalyptus therefore enables the creation of private clouds within the organization. It also supports the same API as Amazon Web Services, which is a set of services that together form a scalable, reliable, and inexpensive computing platform in the cloud, and is provided by Amazon, which is one of the largest public cloud providers. This is why it becomes much easier for small to medium-sized organizations to advance into the public cloud, offering the flexibility to move from their private cloud to a public cloud when resources are needed. There are two versions of Eucalyptus, namely the enterprise edition which supports VMWare’s vSphere and the open source edition, which supports the KVM and Xen virtualization techniques (Eucalyptus Systems, 2014).
PART II

Technical implementation of messaging technologies
Chapter 5

IP Multimedia Subsystem messaging service in the Cloud

In this chapter research done concerning the IP Multimedia Subsystem (IMS) messaging service in the Cloud is briefly described. The objective was to implement a basic open source IMS messaging service with the same architecture as the SMS messaging service as described in Chapter 2. Various open source IMS solutions were evaluated, as described in Chapter 3, with the purpose of designing and implementing this basic messaging service in a timely manner. The IMS implementation using Open IMS Core resulted to be more complicated than expected and could not successfully be implemented. In addition, the open source private cloud infrastructure using OpenNebula could not be realized due to the lack of required networking expertise and required resources. In this chapter the design and partial implementation of both the IMS messaging service and the private cloud are illustrated and briefly described.

Chapter 5 is organized as follows: section 5.1 provides a brief description and illustration of the design of the IMS messaging service, and followed by section 5.2 with a description of the IMS messaging service implementation, including the IMS environment, the Application Server installation, the actual SIP-based messaging service and the SIP request traffic generator. Section 5.3 provides a description and illustration of the design and implementation of the open source cloud service. Section 5.4 describes briefly the problems faced with both the implementation of both the IMS service and the open source private cloud, followed by section 5.5, which provides possible solutions and research recommendations.

5.1 Design of IMS Messaging service

The messaging service consists of the core elements of an IMS architecture that are described in section 3.4.2 (the HSS) and 3.4.4 (the I-CSCF, P-CSCF and S-CSCF). Also a SIP-Application Server is added to the basic IMS network to make it possible to deploy the IMS messaging service based on the SIP protocol. When the basic IMS environment is setup it can start to receive SIP requests to be processed. Based on the SIP request’s content a messaging traffic is generated when the request reaches the IMS messaging service on the SIP-Application Server. This way sending a high load of SIP requests a high workload at peak hours is simulated. To send high loads of requests the SIPP traffic generator tool is used, which acts as the user agent that sends SIP requests to the P-CSCF.

The following figure provides an illustration of the IMS environment required to deploy the IMS messaging service.
In the following figure a context diagram of the IMS messaging is shown with the relation between the entities and the protocols used between entities. This diagram consists of the same entities as the HTTP messaging service described in section 6.2, but with the SIP protocol used to communicate between entities.

**Figure 5.1: IMS messaging service architecture**

**Figure 5.2: IMS messaging service context diagram**

### 5.2 Implementation of IMS Messaging service

The implementation phase of the IMS messaging service consists of the implementation of the IMS core components, described in section 5.2.1, the implementation of the Application Server to deploy the messaging service as briefly described in section 5.2.2, and the actual IMS messaging service implementation as explained in section 5.2.3.
5.2.1 IMS core components implementation

The open source solution used to implement the IMS architecture was Open IMS Core (Fraunhofer FOKUS NGNI, 2008). As described in section 3.1.5, to implement an IMS architecture, the core components are required. This was made possible using the FOKUS Open IMS Core solution, which is an open source implementation of the core IMS elements created by the German Fraunhofer Institute FOKUS. The core IMS elements include the Call Session Control Functions (S-CSCF, P-CSCF and I-CSCF) and Home Subscriber Server (HSS) (Fraunhofer FOKUS NGNI, 2008). These four components are based on open source software, such as the SIP Express Router (SER) or MySQL. The SER is a high-performance, configurable, free SIP server licensed under the open-source GNU license (IPTEL.org, 2008). The Fraunhofer Institute FOKUS is known as a leading research institute in the field of open communication systems, and it created the “Open IMS Playground @ FOKUS” (Fraunhofer FOKUS NGNI, 2008), which is an open and vendor independent IMS test laboratory that offers the major IMS core components, that is used to benchmark and to test IMS conformance and interoperability.

Three options were available to create an own IMS architecture. First, a network could be created using functional components available from various vendors on the market. This would consume many research hours and finally the risk existed to be continuously testing the interoperability of the IMS components, as much as the performance of the IMS messaging service. Due to time constraints, this was not an option. A second options was to develop an own IMS application, consisting of components based on the 3GPP standards. This would also consume too many research hours, and would be risky if afterwards it would not succeed. For this reason the Open IMS Core, which is freely available and already includes code for the four core IMS components in the IMS architecture, was chosen as the IMS open source solution. The installation of Open IMS Core is based on the execution of Linux commands using the command line terminal on an Ubuntu server, and modifying configuration files adding the IP addresses of the components, so that these can reach each other. The installation instructions can be found in Appendix B Open IMS Core installation instructions.

5.2.2 Application Server implementation

For the Application Server the JBoss 7.1.1 Application Server was installed on the same Ubuntu server on which the Open IMS Core solution was installed. The IMS messaging service, described in the next paragraph, is deployed on this application server’s deployment folder, making the messaging service accessible.

5.2.3 IMS Messaging service

The IMS messaging service was implemented using the Eclipse IDE, Java EE and the Mobicents SIP Servlets API, and the Simple Logging Facade for Java (SLF4J) API.

For the implementation of an application to run on an Application Server in the IMS architecture, a SIP API is required to implement SIP based services. The Mobicents SIP Servlets is an open platform enabling the development and deployment of portable and distributable SIP and Converged JEE services. It is the first open source certified implementation of the SIP Servlet v1.1 (JSR 289 Spec) that can run on top of either the JBoss Application Server or the Tomcat Servlet Container (Sipservlets, 2012). Mobicents SIP Servlets is led by TeleStax Inc. (Telestax, 2013) and developed collaboratively by a community of individual and enterprise contributors. TeleStax is a global company based in Austin, Texas, USA with regional offices in Europe, Africa, Asia and South America. TeleStax provides Open Source Communications software and services that facilitate the shift from legacy SS7 based IN networks to IP based LTE and IMS networks hosted on private (on-premise), hybrid or public clouds,
which are described in Chapter 4 Cloud computing. An example of a SIP Servlet handling incoming SIP request in the IMS messaging service can be found in Appendix B IMS Messaging service.

5.3 Design and implementation of open source cloud infrastructure

For the implementation of an open-source private cloud a couple of popular options were available, as described in section 4.7 Main Cloud computing providers, from which at the time of the research the most popular two solutions, OpenStack and OpenNebula, were chosen as possible options. Research was done about their features and differences to come up with the following description.

OpenNebula is an open-source effort focused on user needs, while OpenStack is a vendor-driven effort (OpenNebula.org, 2014). The main differences between these two open-source models are as follows:

1. **Internal Organization**
   OpenNebula offers a single integrated, comprehensive management platform for all cloud subsystems, while OpenStack comprises many different subprojects aimed at building the different subsystems in a cloud infrastructure.

2. **Governance Model**
   OpenNebula follows a centralized single enterprise-ready approach, while OpenStack is controlled by a Foundation driven by vendors.

3. **Roadmap Definition**
   OpenNebula’s roadmap is completely driven by user’s needs with features that meet real demands, and not features that result from an agreement among the different vendors participating in the management board of the project, which is the case for OpenStack.

4. **Contributor Profile**
   While in OpenNebula most of contributions come from the users of the software, the contributors to OpenStack are mostly vendors building their own OpenStack-based cloud product.

OpenStack is governed by a consortium of competitors, trying to create its own product or to provide compatibility for its particular device. Because OpenNebula follows a centralized model with a strong individual leadership, it is the best open-source solution (among the two) to quickly build a production-ready enterprise-class open-source product, making it the most effective way to focus on engineering quality, to prioritize user needs, and also to ensure long-term support. OpenNebula is made for users by users, and OpenStack is made for vendors by vendors. At the time of the research about OpenNebula and OpenStack as the two options to implement an open-source private cloud, OpenNebula was the most stable solution, and had better and larger communities for technical support. Also OpenNebula is by design very light, in terms of dependencies and components, and easy to extend and adapt. In practice its simple and quite effective design has proven (and is accepted) to be very scalable and “hackable” leading to flexible and agile Cloud deployments. Because of OpenNebula’s true open source blood it guarantees users complete interoperability with every existing infrastructure component already available. Thus it avoids the vendor lock-in using common open industrial standards, such as EC2 API and Open Cloud Computing Interface (OCCI). Additionally, OpenNebula is flexible in the sense that it does not embrace a particular hypervisor and it also does not have any specific infrastructure requirements, thus fitting well into any pre-existing environment, storage, network, or user-management policies. For these reasons OpenNebula was
chosen as the private cloud open source solution. OpenNebula’s architecture (Archives.OpenNebula.org, 2014) is shown in Figure 5.3.

![OpenNebula Architecture Diagram](Archives.OpenNebula.org, 2014)

Figure 5.3: OpenNebula architecture (Archives.OpenNebula.org, 2014)

An OpenNebula system consists of the following components:

- **Front-end**, which is a server that executes the OpenNebula services.
- **Hosts**, which are hypervisor-enabled cluster nodes that provide the resources needed by the VMs.
- **Image repository**, which is any storage medium that holds the base images of the VMs.
- **Networking**, the physical network infrastructure required to support VLAN for the VMs and service links (e.g. storage).

To install OpenNebula the available physical infrastructure must adopt a cluster-like architecture, joining all network nodes via a physical network.

**Front-End**
The Front-End node is the server that holds the OpenNebula installation. This machine requires access to the image repository storage and network connectivity to each host. This server executes the OpenNebula services, including:

- Management daemon and scheduler
- Monitoring and accounting daemon
- Web interface server
- Cloud API servers

**Hosts**
The hosts are the physical machines that require a hypervisor to create and run VMs. The servers had proper features in the sense of memory and hard disk space, but their CPUs did not support virtualization. This was later noticed after trying out several installations, with several hypervisors types, including VirtualBox and XEN hypervisor. Hence, the decision was made to abandon this implementation, and seek for a more proper and feasible implementation.

**Image Repository & Storage**
The Image Repository handles the VM Image files and needs to be accessible through the front-end using any suitable technology, such as NAS, SAN or direct attached storage. Images are transferred to the hosts by the front-end node, to use them in the VMs.
These components were successfully installed with the exception of the hypervisor on the cluster nodes. This was due to the available servers not being compatible with the installation because of the lack of the virtualization feature.

5.4 Load test tool
To generate high loads of SIP requests to send to the IMS environment, subsequently reaching the SIP-Application server to generate messaging traffic, SIPp traffic generator tool is used. SIPp is a free Open Source test tool or traffic generator for the SIP protocol (SIPp, 2013), and is used to send high loads of SIP requests to a server, the same way JMeter is used for HTTP load testing.

5.5 Challenges and problems faced
5.5.1 IMS Messaging solution
The IP Multimedia architectural framework was thoroughly researched, together with possible technical solutions, but due to lack of support for the implementation of the IMS solution, and the time invested in research and development without any success, the decision was made to abandon the IMS implementation and try to focus more on the cloud aspect of this project.

5.5.2 Cloud solution
In the process of the implementation of cloud, in this case a private cloud, the installation of hypervisors is a requirement. The hypervisor is responsible for the creation and running of virtual machines, this way making the cloud implementation possible. Without hypervisors, virtualization is not possible; hence no cloud can be implemented. Several issues were encountered when trying to install hypervisors on the available infrastructure. In the end it was noted that these cluster nodes were too outdated, and thus missing important characteristics that are required for the successful installation of hypervisors on these servers, i.e. virtualization support. According to Desai et al. (2013), to make virtualization possible, the right hardware is required. Intel and AMD created new processor extensions to the x86 architecture, which are known as the Intel VT-x and AMD-VT. The Intel 80286 chipset found in this new type of processors has introduced two methods of addressing memory: real mode and protected mode. The protected mode provides features enabling support for multicasting, such as hardware support for virtual memory and x86 memory segmentation (Carvalho dos Santos, Ramos, 2009). Virtualization takes benefit of x86 architecture for providing protected mode virtualization. The available servers do not support virtualization, as these possess an outdated chipset. Based on this issue and the long time invested in trying to implement a private cloud, the decision was made to abandon the implementation of this private cloud.

5.6 Possible solutions and research
To successfully implement a private cloud, the required infrastructure must be acquired, including up to date servers that support virtualization, and thus hypervisor installation. This may require huge investments, which could be avoided by using a virtual private cloud (VPC), where the infrastructure is owned by a public cloud provider. VPCs provide secure data transfer between a private enterprise and a public cloud provider, ensuring that each customer’s data remains isolated from every other customer’s data both in transit and inside the cloud provider’s network (Rouse, 2013). Another solution would be making use of a public cloud, such as Amazon AWS cloud platform to deploy services for customers. It is up to each enterprise or organization to analyze each option and determine if their services can be deployed on any of these platforms, considering privacy issues and confidentiality of processes and information.
Chapter 6
HTTP-based SMS messaging solution in the Cloud

The purpose of this chapter is to provide insight and understanding in the design and implementation of the HTTP messaging solution that will be used in Chapter 7 to execute performance tests in three different scenarios. Firstly the methodology used for the development of the messaging solution is described in section 6.1, followed by the description and illustration of the design of this messaging solution, which consists of six messaging components, in section 6.2. Subsequently, the implementation phase is described in section 6.3, including a brief description of the tools required to implement the messaging solution, followed by a brief explanation of its implementation using the Java programming language and the Spring Web MVC framework. The design and implementation of the cloud infrastructure using the Amazon Web Service cloud-platform is described next in section 6.4, including a description and illustration for all three scenarios, i.e. the deployment on a proprietary server, the deployment in the Cloud using Auto Scaling and deployment in the Cloud using Elastic Load Balancing. In section 6.5 the design of the messaging solution in the Cloud is illustrated, followed by a description of how the deployment of the messaging solution in the cloud is carried out in section 6.6. The final section presents some issues that were encountered during the design and execution of this part of the thesis, and some recommendations for future work.

6.1 Messaging solution development methodology

The development methodology used for the HTTP-based messaging solution is based on extreme programming (XP), but does not entail all characteristics of XP, due to time constraints and the research character of this project. The intention was to create a messaging solution to be used to evaluate its availability when deployed in three different environments, as described in section 1.2. The quality of the implemented messaging solution was verified based on already existing similar messaging solutions implemented at T-Sparx.

Extreme programming is defined as a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software (Beck, 1999). This methodology has the following characteristics:

- Early, concrete, and continuing feedback from short cycles.
- An incremental planning approach, which quickly comes up with an overall plan that is expected to evolve through the life of the project.
- The ability to flexibly schedule the implementation of functionality, responding to changing business needs.
- Reliance on automated tests written by programmers and customers to monitor the progress of development, to allow the system to evolve, and to catch defects early.
- Reliance on oral communication, tests, and source code to communicate system structure and intent.
- Reliance on an evolutionary design process that lasts as long as the system lasts.
- Reliance on the close collaboration of programmers with ordinary skills.
- Reliance on practices that work with both the short-term instincts of programmers and the long-term interests of the project.

XP is a software development discipline designed to work with projects that can be built by teams of two to ten programmers. The development of the messaging solution has an XP character in the
sense that the solution was implemented in short cycles in an iterative way, providing constant feedback to the customer (company for which this thesis was carried out), including an incremental planning approach. In the following two sections the design and implementation of the messaging solution are described and illustrated.

6.2 Design of HTTP Messaging solution

The messaging solution consists of a simplified simulation of six separate applications representing six different services or components that are used in a messaging scenario. With “simulation” is referred to a simple application that just receives a request and processes this request based on its contents, subsequently sending a response to the request-origin or forwarding the request to a next component. This way forming a messaging traffic, simulating a real world messaging solution. All six applications are developed with the same code, with the purpose of generating message traffic between components, responding to received requests or forwarding requests.

These components include:

- MessagingGateway
- ProCharge
- BillingGateway
- PrePaid Platform
- HLR (located in the telecom operator’s environment)
- RIM (BlackBerry service)

All requests from clients, i.e. mobile devices, are first sent to the MessagingGateway component. This component receives the request, and based on its contents determines what happens next. This request may be ignored returning an error response or may be forwarded to other components to process the request. This is illustrated in the following messaging solution context diagram. A context diagram defines the boundary between the system and its environment, and shows the entities that interact with it. In the diagram below the protocol used for communication between components is shown, and it consists of the following entities:

- Mobile device sending the HTTP request,
- Jiver platform consisting of the services provided by T-Sparx, which include:
  - MessagingGateway,
  - ProCharge,
  - BillingGateway
  - Prepaid Platform
- External services, including RIM and HLR
In the two figures below an example is illustrated of the BlackBerry service activation. The arrows indicate a request<->response interaction between end-points, based on the request BB_ON to activate the BlackBerry service for a user account. The request is received by the MessagingGateway, which checks the contents of the request and determines that for the BB_ON request the request must first be forwarded to the ProCharge and BillingGateway components. First the request is forwarded to the ProCharge component, as can be seen in Figure 6.2 which subsequently forwards the request to the HLR and RIM components.

When this flow is done, the MessagingGateway forwards the request to the BillingGateway component, as can be seen in Figure 6.3, which afterwards forwards the request to the PrePaid component. When both request flows in both directions are successfully done, the MessagingGateway returns a success response to the client.
The implementation of this messaging solution is described in the following section including the tools, i.e. development environment, programming language and libraries, required to make the above messaging traffic possible.

6.3 Implementation of HTTP Messaging solution

In the following sub-section the tools used to implement the HTTP messaging solution are described followed by the logic description of the messaging solution implemented in Java in section 6.3.2.

6.3.1 Tools

The tools used to implement the messaging solution based on the HTTP protocol include the Eclipse Integrated Development Environment (IDE), Java Enterprise Edition (J2E) and the Java Spring framework. Libraries that were used for the technical implementation using Java include the Apache HttpClient, the Simple Logging Façade for Java (SLF4J) library for logging purposes.

- **Eclipse IDE**
  
  Eclipse is a popular Integrated Development Environment software, written in Java, used to develop applications in the Java programming language, and also with the use of plug-ins other programming languages, such as Java, Python, JavaScript, PHP and Ruby (Eclipse, 2014).

- **Java Enterprise Edition (Java EE)**
  
  Java EE is considered the de-facto standard for delivering secure, robust, scalable multi-platform applications and services. It is Oracle's enterprise Java computing platform, which provides an API and runtime environment for developing and running enterprise software, such as network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications (Java, 2012).

- **Apache HttpClient**
  
  The HttpClient library provides the ability to build HTTP-aware client and server applications such as web browsers, web spiders, HTTP proxies, web service transport libraries, or systems that leverage or extend the HTTP protocol for distributed communication.
- **Apache Maven software project management and comprehension tool**
  Apache Maven is a software project management and comprehension tool (Apache Maven Project, 2014). From the project object model (POM) Maven can manage a project's build, reporting and documentation. Maven’s dependency management is a mechanism for centralizing dependency information (Apache Maven Project, 2014).

- **Simple Logging Façade for Java**
  The Simple Logging Facade for Java (SLF4J) serves as an abstraction for various logging frameworks, such as java.util.logging, logback, log4j, allowing the end user to plug in the desired logging framework at deployment time.

- **Spring Web Model View Controller (MVC) Framework**
  The Spring Framework is an open source application framework that can make Java EE development easier. It consists of a container, a framework for managing components, and a set of snap-in services for web user interfaces, transactions, and persistence. A part of the Spring Framework is Spring Web MVC, an extensible MVC framework for creating web applications (NetBeans, 2013). According to Johnson et al. (2013), Spring's Web MVC framework is designed around a DispatcherServlet that dispatches requests to handlers. The default handler is a very simple Controllers Interface, just offering a ModelAndView handleRequest(request, response) method. Spring's web MVC framework is request-driven, designed around a central servlet that dispatches requests to controllers and offers other functionality facilitating the development of web applications. Spring’s DispatcherServlet is completely integrated with the Spring Inversion of Control (IoC) container and allows the use of every feature Spring has to offer. Spring Framework’s IoC container manages Java objects, from instantiation to destruction, through its BeanFactory. The name BeanFactory refers to the instantiation of Java components, called beans, by the IoC container.

- **Apache Tomcat 7**
  Apache Tomcat is an open source web server and servlet container developed by the Apache Software Foundation (ASF). It implements among others the Java Servlet specification and provides a “pure Java” HTTP web server environment for Java applications to run in. Tomcat is used in this project to deploy the WAR file that is exported from Eclipse, as the resulting messaging solution.

### 6.3.2 Java application

This section provides a brief description and overview of the logical implementation of the Java application. For each messaging components, namely:

- MessagingGateway
- ProCharge
- BillingGateway
- PrePaid Platform
- HLR
- RIM

(as described in section 6.2) a Java class was created named, the RequestProtocolHandler bean, each containing the same code. This way each component has the same behavior and a straightforward messaging traffic is generated. The logic implemented in Java to make the messaging traffic possible is defined in this section. The Spring Web MVC framework is used to implement a web application that is able to receive requests, process these requests and generate a response that is sent back to the sender of the request. The MVC design pattern, shown in figure 6.4, consists of
- Model, which represents the core business logic and state, i.e. it executes the business logic and changes the state of the model.
- View, which represents the presentation of the data, i.e. it renders the content of the model state by adding display logic
- Controller, which acts as an interface between the model and the view. It translates the interaction with the view into action to be performed by the model.

![Model-View-Controller (MVC) Architecture](image)

**Figure 6.4: Model View Controller architecture (Kumaraswamipillai, 2011)**

In the following figure the requesting processing workflow is shown. The DispatcherServlet is represented in this figure by the Front Controller, which receives all HTTP requests and delegates control to the corresponding Controller to handle the request, in this case the RequestProtocolHandler bean.

![High level view of the requesting processing workflow in Spring Web MVC](image)

**Figure 6.5: High level view of the requesting processing workflow in Spring Web MVC (Samsung SDS, 2009)**

In this project the view part was not required and thus not implemented.

To develop a web application using the Spring Web MVC framework two important files must be defined, namely the `web.xml` and the `servlet.xml` files. The `web.xml` defines the DispatcherServlet as the web application servlet with a corresponding self-defined name, and a URL mapping used to map requests that have to be handled by the DispatcherServlet.
In this case the corresponding servlet.xml file would be called rest-servlet.xml, which contains all Spring Web MVC-specific components (beans), in this case the definition of the RequestProtocolHandler bean, and the SimpleUrlHandlerMapping bean that belongs to the Spring framework, which maps URLs to request handler beans.

An example HTTP URL, which represents an SMS sent by a user, received by the DispatcherServlet is


This request is sent to the messaging-gateway application (which includes the above configuration). The URL includes the pattern /rest/* which is handled by the DispatcherServlet and delegated to the controller, the RequestProtocolHandler bean. This bean includes the ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) method, which processes the request and corresponding response. In this case the parameters present in the URL are extracted from the URL and added to a series of elements in a Java Enumeration. Subsequently is looped through these parameters to find the source of the request and the message_data, which is the content of the actual SMS. Each application has a corresponding "properties" file with the mapping between the SMS key sent and the route the request must be forwarded to. Two examples include:

BB_ON = forward|pro-charge forward|billing-gateway
BB_ON = reply|200

In the first example, when the messaging gateway receives the request with content BB_ON it forwards the request firstly to pro-charge, subsequently to the billing-gateway.

In the second, when the messaging gateway receives the request with content BB_ON it returns a response with contents 200.

Based on the contents of the SMS, if the key is 'reply', an exception is thrown in the application, which sends a 500 response to the origin. If the key is ‘forward’ the method int retrieveResponse(Request reqToSend, Action newAction) is called, which creates the destination URI consisting of the required port number for the specific application to forward the request to. This destination URI is used in the callUrl(String host, int port, String urlToCall) method to create the actual HttpRequest. With an HttpClientBuilder object an HttpClient object is built, which is used to send the HTTP request. Its response is received by the HttpResponse object.
This way the request is processed and a `ModelAndView` object is returned, which the `DispatcherServlet` will usually render, but in this case no view is rendered.

For a better understanding, the implementation in Java of the messaging solution can be found in the attached zip file with the implemented code.

The above logic is shown as pseudo-code as follows:

```java
Initialize ModelAndView object modelAndView.

ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) {
    Initialize empty enumerator series of elements.
    Initialize origin to null.
    Initialize data to null.

    Get parameters from the request and add them to the enumerator.
    While there are parameters in enumerator {
        Find source parameter and add to origin.
        Find message_data and add to data.
    }

    Initialize properties enumerator containing contents of properties file

    While there are properties in properties enumerator {
        Initialize action object with action key and action destination properties.
        Add property key as the action key.
        Add property value as action destination.

        If action key equals 'forward' {
            Initialize responseStatus string with value 500.
            Initialize HTTPResponse response.
            Create destination address.

            Create GET HTTPRequest request with destination address.
            Create HTTPClient to execute request.
            Add result of request execution to response.
            Get response status code and add to response-status.

            If responseStatus equals 200 {
                Return modelAndView
            } else if responseStatus does not equal 200 {
```

```
6.4 Design and implementation of cloud infrastructure using Amazon Web Services

Before diving into the description of the Cloud design and implementation, and the deployment of the messaging solution in the Cloud, the method and tool used to carry out remote tests in the AWS cloud is described in section 6.4.1. This is followed by a description of some basic concepts of the AWS cloud in section 6.4.2, including Amazon EC2 instances, instance types and security groups.

6.4.1 JMeter

Apache JMeter (The Apache Software Foundation, 2013) is an open source desktop application created with the Java programming language, designed to load test functional behavior and measure performance. Originally it was designed to test web applications, but has since expanded to other test functions. Apache JMeter may be used to test performance both on static and dynamic resources, such as web dynamic languages (including PHP, Java, ASP.NET, etc.), Java Objects etc. It can be used to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load types. JMeter can be used make a graphical analysis of performance or to test server/script/object behavior under heavy concurrent load. A JMeter test plan emulates the presence of numbers of users that send requests to the messaging solution, creating heavy loads to analyze the performance of the messaging solution on a particular infrastructure configuration, such as a single server, a load balancing or auto scaling scenario.

6.4.2 Remote testing method and tool

To make it possible to test instances remotely a method must be implemented to execute JMeter test plans on the Amazon EC2 instances from a local computer. Executing performance tests on Amazon EC2 instances is not possible by directly entering the Amazon EC2 instance URI in JMeter. Running a JMeter test plan on an Amazon EC2 instance is not as straightforward as it may seem. There exist some difficulties when trying to use a local machine to control the JMeter test on a remote Amazon EC2 instance. To make this possible Remote Method Invocation (RMI) must work over multiple subnets and firewalls to allow the local machine to control multiple remote EC2 instances. RMI enables the programmer to create distributed Java technology-based to Java technology-based applications. It provides a simple and direct model for distributed computation with Java objects. RMI is Java's remote procedure call (RPC) mechanism. This can be achieved by tunneling RMI communication and patching JMeter, but it is a pretty complex process as described in Oliver (2012).
Hence, research was done to find a possible solution for this problem, which is the jmeter-ec2 shell script. JMeter-EC2 is a simple, freely-available shell script that automates running Apache JMeter on Amazon EC2 instances, or any other server (Oliver, 2014). Some functionalities of this script include:
- Using Amazon’s API to launch instances
- Installing Java & JMeter
- Copying test files to each EC2 instance
- Adjusting thread counts to ensure the load is evenly distributed over each host
- Editing the Apache JMeter test plan file (.jmx) for file paths to external data files
- Displaying real-time aggregated results from the test as it is running
- Downloading and organizing all JMeter generated text files containing test results (JTL files)
- Terminating EC2 instances.

This jmeter-ec2 project consists of the following files that make the proper execution of the script possible:

- **jmeter-ec2.properties**

  This file contains several properties, such as the location of the jmeter-ec2 shell script, the AMI-ID of the instance to upload the script to, the instance type, the instance security group, the amazon key pair name, the PEM file name and the PEM path. These properties are required for running the script. The remote Amazon EC2 instance, which the shell script must be uploaded to, must be specified as the REMOTE_HOSTS property, in this case the Load generator instance (Figure 6.5). The amazon key pair is essential as Amazon EC2 uses public–key cryptography to encrypt and decrypt login information. Public–key cryptography uses a public key to encrypt a piece of data, such as a password, and then the recipient uses the private key to decrypt the data. The public and private keys are known as a key pair. The key–pair file name extension is .PEM.

For this project the following jmeter-ec2.properties file is defined:

```bash
# This is a java style properties file for the jmeter-ec2 shell script
#
# It is treated like a normal shell script and must have executable permissions
#
# See README.txt for more details about each property
#
# Suggested AMIs
#
# OS            Type   AMI id        User       Available sizes
# Ubuntu            64bit ami-e1e8d395 ubuntu   t1.micro, m1.small, m1.medium, c1.medium, m1.large, m1.xlarge, etc.
#
# IMPORTANT - t1.micro is not recommend for anything beyond development, it works from a shared resource pool that can fluctuate, skewing test results.
#
# The root for this script - all files should be put here as per the README
LOCAL_HOME="/Users/plopezbarbosa/Dropbox/AfstudeerProject/jmeter-ec2-master"

# This can be left as /tmp - it is a temporary working location
REMOTE_HOME="/tmp"
```
# A suitable AMI
AMI_ID="ami-f1456c98"

# Should match the AMI - I do not recommend using mic$
INSTANCE_TYPE="m1.large"

# The name of *your* security group in *your* Amazon$
INSTANCE_SECURITYGROUP="smsgwtest"

# The name of the Amazon Keypair that you want to us$
AMAZON_KEYPAIR_NAME="smsgwkey"

# The full name of the pem file you download$
PEM_FILE="smsgwkey.pem"

# The path to your pem file
PEM_PATH="/Users/plopezbarbosa/Dropbox/AfstudeerProject"

# Should match the AMI
USER="ubuntu"

# Email to be used when tagg$
EMAIL=me@piloba.com

# Specify the region you wil$
REGION="eu-east-1"

# Should match the AMI and be available in the regio$
INSTANCE_AVAILABILITYZONE="eu-east-1a"

# How often the script prints running totals to the $
RUNNINGTOTAL_INTERVAL="3"

# A list of static IPs that can be assigned $
ELASTIC_IPS=""

# The port number sshd is running on,
REMOTE_PORT="22"

# The version of JMeter to be used. Must be the full$
JMETER_VERSION="apache-jmeter-2.9"

# Specify the JAVA binary you will be using in the c$
JAVA_VERSION_32='jre-6u32-linux-i586.bin'

# Specify the JAVA binary you will be using in the c$
JAVA_VERSION_64='jre-6u32-linux-x64.bin'

# REMOTE_HOSTS
#
# If this is set then the script will ignore INSTANCE_COUNT passed in at the
# command line and read in this list of hostnames to run the test over
# instead. If it is not set then n number of hosts will be requested from
# Amazon.
#
# Must be a comma-separated list, like this:
# REMOTE_HOSTS="ec2-46-51-135-180.eu-west-1.compute.amazonaws.com,ec2-176-34-
# 204-10.eu-west-1.compute.amazonaws.com"
The following instance families are available:

- **Storage capabilities.**

The host computer used for the 2014.

A single AMI can be used to launch one or thousands of instances provided by AWS, configuration, including an operating system, which defines your operating environment. AMIs are provided by AWS, the AWS user community, on the AWS Marketplace, and can also be created and shared. A single AMI can be used to launch one or thousands of instances (Amazon Web Services, 2014).

Instances are available in different, so-called, instance types, which determine the hardware of the host computer used for the instance. Each instance type offers different compute, memory, and storage capabilities.

The following instance families are available:

- **General purpose,** which provides a balance of compute, memory, and network resources.
- **Compute optimized,** which is optimized for applications that benefit from high compute power.
- **Memory optimized,** which is optimized for memory-intensive applications.
- **Storage optimized,** which provide Intel Xeon processors and direct-attached storage options optimized for applications with specific storage capacity and random I/O requirements.
- **Micro instances,** which are a very low-cost instance option, providing a small amount of CPU resources.
- **GPU instances,** which provide Intel Xeon processors and high-performance NVIDIA GPUs intended for graphics and general purpose GPU compute applications.

6.4.3 **Amazon instances**

An EC2 instance is the fundamental building block in the AWS cloud, and can be thought of as a virtual server that can run applications. Amazon EC2 instances are created from an Amazon Machine Image (AMI) and by choosing an appropriate instance type. An AMI is a template that contains a software configuration, including an operating system, which defines your operating environment. AMIs are provided by AWS, the AWS user community, on the AWS Marketplace, and can also be created and shared. A single AMI can be used to launch one or thousands of instances (Amazon Web Services, 2014).

Instances are available in different, so-called, instance types, which determine the hardware of the host computer used for the instance. Each instance type offers different compute, memory, and storage capabilities.

The following instance families are available:

- **General purpose,** which provides a balance of compute, memory, and network resources.
- **Compute optimized,** which is optimized for applications that benefit from high compute power.
- **Memory optimized,** which is optimized for memory-intensive applications.
- **Storage optimized,** which provide Intel Xeon processors and direct-attached storage options optimized for applications with specific storage capacity and random I/O requirements.
- **Micro instances,** which are a very low-cost instance option, providing a small amount of CPU resources.
- **GPU instances,** which provide Intel Xeon processors and high-performance NVIDIA GPUs intended for graphics and general purpose GPU compute applications.
Instances can be launched into one of two platforms: the EC2-Classic and the EC2-VPC. In EC2-Classic the instances run in a single, flat network that is shared with other customers, while in EC2-VPC instances run in a virtual private cloud (VPC) that’s logically isolated to the user’s AWS account.

When accessing instances security groups can be used to control which users can access the instances. These are similar to an inbound network firewall that enables the specification of the protocols, ports, and source IP ranges that are allowed to reach your instances. It acts as a virtual firewall that controls the traffic for one or more instances. An instance can be associated with one or more security groups. To each security group rules are added that allow traffic to or from its associated instances. Security groups of instances that were already launched in EC2-Classic cannot be modified. However, rules can be added to or removed from a security group, and those changes are automatically applied to all instances that are associated with the security group. In EC2-VPC after instances have been launched its security groups can still be modified and rules can be added to or removed from the security group. These changes are also automatically applied to all instances that are associated with the security group. In this project a security group was created to which the created instances running the messaging solution were associated. By default the SSH, HTTP and HTTPS rules form part of the security group (Table 6.1). In addition a custom TCP rules was for port 8080 to make the messaging solution’s messaging-gateway component be accessible running on port 8080.

<table>
<thead>
<tr>
<th>Type</th>
<th>Protocol</th>
<th>Port Range</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH</td>
<td>TCP</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>0.0.0.0/0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTP</td>
<td>TCP</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>0.0.0.0/0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HTTPS</td>
<td>TCP</td>
<td>443</td>
<td></td>
</tr>
<tr>
<td>0.0.0.0/0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom TCP</td>
<td>TCP</td>
<td>8080</td>
<td></td>
</tr>
<tr>
<td>Rule</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0.0.0/0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 6.1: Security group rules*

**Chosen instance type**

For this project the **m3.xlarge** instance type was chosen, which belongs to the General Purpose EC2 instance family. This instance has the following hardware specifications:

<table>
<thead>
<tr>
<th>Name</th>
<th>Memory</th>
<th>Compute units</th>
<th>Storage</th>
<th>Architecture</th>
<th>I/O Performance</th>
<th>Max IPs</th>
<th>API Name</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>m3.xlarge</td>
<td>15.0GB</td>
<td>13 (4 core x 3.25 unit)</td>
<td>80 GB (2 x 40 GB SSD)</td>
<td>64-bit</td>
<td>High / 1000 Mbps</td>
<td>60</td>
<td>m3.xlarge</td>
<td>$0.450 hourly</td>
</tr>
</tbody>
</table>

*Table 6.2: Hardware specification m3.xlarge EC2 instance*

Briefly this type of instance consists of a 64-bit architecture, 4 virtual CPUs, 15 GiB RAM memory and 2 Solid State Drive (SSD) hard disks of 40GB each.
The reason for choosing this instance type was based on a trial and error method, which is characterized by repeated, varied attempts on a certain task, which are continued until success. These tasks included continuously attempting to start a JMeter test using the jmeter-ec2 script, using a test plan with a large number of threads on an Amazon EC2 instance. This large number of threads caused regular-sized instances, such as the m1.medium (1 virtual CPU (vCPU) and 3.75 GB memory) or the m1.large (2 vCPUs and 7.5GB memory) to use all processing power at the start of the test causing errors in the test that led to an incorrect test. It was noted that a JMeter test requires enough processing power to manage the first load sent to the instance, which subsequently runs smoothly using less processing power. Hence, an EC2 instance was required with enough processing power and enough memory, but that also was not the most expensive. After numerous tests, it was concluded that an instance with 4 vCPUs could handle large loads of requests. Based on the combination of 4 vCPUs and the lowest cost the following options were evaluated (note that the listed prices are effective at the time of writing and may have changed):

**General purpose**

<table>
<thead>
<tr>
<th>Name</th>
<th>Memory (GiB)</th>
<th>vCPU</th>
<th>Storage</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.xlarge</td>
<td>15.0</td>
<td>4</td>
<td>1680 GB (4 x 420 GB)</td>
<td>$0.480 per hour</td>
</tr>
</tbody>
</table>

*Table 6.3: Hardware specification m1.xlarge EC2 instance*

<table>
<thead>
<tr>
<th>Name</th>
<th>Memory (GiB)</th>
<th>vCPU</th>
<th>Storage</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>m3.xlarge</td>
<td>15.0</td>
<td>4</td>
<td>80 GB (2 x 40 GB SSD)</td>
<td>$0.450 per hour</td>
</tr>
</tbody>
</table>

*Table 6.4: Hardware specification m3.xlarge EC2 instance*

**Compute optimized**

<table>
<thead>
<tr>
<th>Name</th>
<th>Memory (GiB)</th>
<th>vCPU</th>
<th>Storage</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>c3.xlarge</td>
<td>7.5</td>
<td>4</td>
<td>80 GB (2 x 40 GB SSD)</td>
<td>$0.300 per hour</td>
</tr>
</tbody>
</table>

*Table 6.5: Hardware specification c3.xlarge EC2 instance*

**Memory optimized**

<table>
<thead>
<tr>
<th>Name</th>
<th>Memory (GiB)</th>
<th>vCPU</th>
<th>Storage</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>m2.2xlarge</td>
<td>34.2</td>
<td>4</td>
<td>850 GB (1 x 850 GB)</td>
<td>$0.820 per hour</td>
</tr>
</tbody>
</table>

*Table 6.6: Hardware specification m2.2xlarge EC2 instance*
**Storage optimized**

<table>
<thead>
<tr>
<th>Name</th>
<th>Memory (GiB)</th>
<th>vCPU</th>
<th>Storage</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2.xlarge</td>
<td>30.5</td>
<td>4</td>
<td>800 GB (1 x 800 GB SSD)</td>
<td>$0.853 per hour</td>
</tr>
</tbody>
</table>

*Table 6.7: Hardware specification i2.xlarge EC2 instance*

The choice was made for the m3.xlarge instance, because compared to the rest of the instances (with the exception of the c3.xlarge) it has the lowest price, enough memory and faster hard disks (using SSD), which may make a difference in the speed the test can be carried out. The c3.xlarge is also a good option that at the time of the tests execution was not considered as an option because of the lower memory, but it is a good option for carrying out the same tests in all scenarios, causing out of memory errors in less time.

### 6.4.4 Proprietary server

In this project no actual privately owned server was used, but two dedicated Amazon EC2 instance in the Amazon AWS cloud as an emulation of the actual proprietary server configuration. The instances used are instances “bought” from Amazon, which may be a basic Amazon On-Demand EC2 instance, an Amazon EC2 Dedicated On-Demand Instance or an Amazon Dedicated Reserved Instance.

The Amazon Dedicated Reserved instance can be purchased by making a low, one-time payment for each instance that needs to be reserved, and in turn a significant discount is received on the hourly charge for that instance. Three types of Reserved Instances exist, namely:

- **Light Utilization Reserved Instance**, which allows instances to be turned off at any point, this way stopping the hourly fee payment at any time. Light Utilization RIs are best suited for periodic workloads that only run a couple of hours a day or a few days per week.

- **Medium Utilization RI**, which allows instances to be turned off at any point, this way stopping the hourly fee payment at any time. Medium Utilization RIs are best suited for workloads that run most of the time, but have some variability in usage, such as web server traffic where demand may increase or decrease throughout the year.

- **Heavy Utilization RI**, which is ideal for steady-state workload, where instances are always running in exchange for the lowest hourly usage fee.

An Amazon EC2 Dedicated On-Demand instance is launched within an Amazon Virtual Private Cloud (Amazon VPC), which runs hardware dedicated to a single customer. This type of instance combines the benefits of both the Amazon VPC and the Amazon AWS Cloud, i.e. hardware-level isolation of Amazon EC2 instances, on-demand elastic provisioning, the pay-per-use feature, and a private, isolated virtual network. A VPC can easily be created, which contains dedicated instances only, providing physical isolation for all Amazon EC2 compute instances launched into that VPC.

In this project no dedicated or reserved instance was purchased, but instead a single Amazon EC2 instance was used for the execution of the performance tests. The infrastructure for the proprietary server scenario is shown in figure 6.6.
Figure 6.6: Proprietary server scenario infrastructure

This infrastructure consists thus of two Amazon EC2 instances, namely:
- The load generator instance (an m1.large instance) for the installation of JMeter and responsible for executing the test plan on the messaging instances, and
- The proprietary instance (an m3.xlarge instance), which contains the deployed messaging solution on Tomcat.

In this project only one proprietary instance is tested. The results of this test are used to calculate the response time and maximum number of requests processed when using two proprietary servers in a cluster-like environment using Corosync Pacemaker as is illustrated in figure 6.6.

The Cloud infrastructure for the messaging solution was designed using the Auto Scaling service and the Elastic Load Balancing service. These two scenarios are described and illustrated in the following two sections.

6.4.5 Auto Scaling

Auto Scaling is a web service that provides the ability to automatically launch or terminate Amazon Elastic Compute Cloud (Amazon EC2) instances based on user-defined policies, health status checks, and schedules (AWS Documentation, 2011). Scaling is the ability to increase or decrease the compute capacity of an application by either changing the number of servers (horizontal scaling) or changing the size of the servers (vertical scaling).

Figure 6.7: Amazon AWS infrastructure
6.4.5.1 Auto Scaling setup

Following the procedure to set-up Auto Scaling is briefly described including the different concepts related to Amazon Auto Scaling.

Auto Scaling group

With Auto Scaling, EC2 instances are categorized into Auto Scaling groups for the purposes of instance scaling and management. With auto scaling groups the minimum, maximum or desired number of running EC2 instances the auto scaling group must have, can be defined. An example of this command is:

```
as-create-auto-scaling-group sms-sim-autoscaling-group
    --launch-configuration sms-sim-autoscaling-lc
    --availability-zones us-east-1a
    --min-size 1
    --max-size 2
    --termination-policies "NewestInstance"
```

Launch configuration

A launch configuration is used by the auto scaling group to launch EC2 instances. With the launch configuration command, the AMI ID, the instance type to be used, key pairs, security groups and block device mapping are specified. An example of this command is:

```
as-create-launch-config sms-sim-autoscaling-lc
    --image-id ami-xxxxxxxx
    --instance-type m3.xlarge
    --key "smsgwkey"
    --group "smsgwtest"
```

Auto Scaling policy

With the Auto Scaling policy, which is attached to the already created Auto Scaling group, metrics can be defined for scaling in and scaling out. The adjustment is specified indicating the number of instances that need to instantiated or need to be terminated, based on the policy type, in this case change in capacity. An example:

```
as-put-scaling-policy sms-sim-scaleout-policy
    --auto-scaling-group sms-sim-autoscaling-group
    --adjustment 1
    --type ChangeInCapacity
```
A police Amazon Resource Name (ARN) is returned after this command is executed, e.g.

\[
\]

**CloudWatch alarm**

CloudWatch is a web service that provides monitoring for AWS cloud resources. Auto scaling on EC2 is based on triggers from CloudWatch. A CloudWatch alarm watches over the scale-in and sale-out metrics that were specified by the Auto Scaling policy. In the following example the alarm “SMSSimLoadTestingHighCPUUtilization” is created and associated with the CPU Utilization metric. An alarm is fired when the average CPU utilization is greater than or equal to 70 percent for a period of 60 seconds. This check is performed only once. The action the alarm takes is publishing to the created Auto Scaling policy, i.e. adjusting the capacity. The action is specified as an ARN. The example is as follows:

```
mon-put-metric-alarm
  --alarm-name SMSSimLoadTestingHighCPUUtilization
  --metric-name CPUUtilization
  --namespace "AWS/EC2"
  --statistic Average
  --period 60
  --threshold 70
  --comparison-operator GreaterThanOrEqualToThreshold
  --dimensions "AutoScalingGroupName=sms-sim-autoscaling-group"
  --evaluation-periods 1
```

In this example the metric CPUUtilization is used for the CloudWatch alarm. This was used in the beginning, but was later replaced by a custom metric based on an instance’s memory usage. More details about this and issues encountered can be found in section 6.7 Issues and recommendations for improvement. A custom metric was setup to monitor memory usage. This metric is subsequently attached to an alarm, and the auto scaling policy is based on that alarm. The custom metric is created using the following script that is added to the home directory on the Ubuntu Linux instance, e.g. the /home/ubuntu/memreport.sh file.

```
#!/bin/bash

export AWS_CLOUDWATCH_HOME=/home/ubuntu/CloudWatch-1.0.20.0
```
export AWS_CREDENTIAL_FILE=/home/ubuntu/aws_credentials
export AWS_CLOUDWATCH_URL=http://monitoring.us-east-1.amazonaws.com
export PATH=$AWS_CLOUDWATCH_HOME/bin:$PATH
export JAVA_HOME=/usr/lib/jvm/jre

# get ec2 instance id


memtotal=`free -m | grep 'Mem' | tr -s '' | cut -d ' ' -f 2`
memfree=`free -m | grep 'buffers/cache' | tr -s '' | cut -d ' ' -f 4`
let "memused=100-memfree*100/memtotal"

mon-put-data --metric-name "FreeMemoryMBytes" --namespace "System/Linux" --dimensions "InstanceId=$instanceid" --value "$memfree" --unit "Megabytes"

mon-put-data --metric-name "UsedMemoryPercent" --namespace "System/Linux" --dimensions "InstanceId=$instanceid" --value "$memused" --unit "Percent"

Once this script is added to the Ubuntu system a cron job is created to call this script every one minute for example:

```
*/1 * * * * /home/ubuntu/memreport.sh
```

This causes the measured memory utilization of the system to be sent to CloudWatch every minute. This way a CloudWatch alarm is created as follows:

```
mon-put-metric-alarm
   --alarm-name SMSSimLoadTestingHighMemoryUtilization
   --metric-name UsedMemoryPercent
   --namespace "System/Linux"
   --statistic Average
   --period 60
   --threshold 30
   --comparison-operator GreaterThanOrEqualToThreshold
   --dimensions "AutoScalingGroupName=sms-sim-autoscaling-group"
   --evaluation-periods 1
```
This fires an alarm when the memory percentage used by the system is greater than or equal to 30%, and causes auto scaling policy sms-sim-scaleout-policy to take action.

**Elastic Load Balancer**

In this scenario, using Auto Scaling, when an instance reaches its capacity to process requests, an identical copy of this instance is booted up to continue handling subsequent requests. To make it possible for subsequent requests to be sent to the new auto scaled instance, a load balancer must be created. Using Auto Scaling with Elastic Load Balancing makes it easy to increase or decrease back-end capacity to meet varying traffic levels. Using the commands for creating an Auto Scaling group, launch configuration and policies etc. the auto scaling is set up. An extra command is used to create a load balancer, namely:

```
elb-create-lb sms-gw-sim-load-balancer
  --headers
  --listener "lb-port=80,instance-port=8080,protocol=HTTP"
  --availability-zones us-east-1a
```

This load balancer is attached to the auto-scaling group using the defined name *sms-gw-sim-load-balancer*:

```
as-create-auto-scaling-group sms-sim-autoscaling-group
  --launch-configuration sms-sim-autoscaling-1c
  --availability-zones us-east-1a
  --min-size 1
  --max-size 2
  --termination-policies "NewestInstance"
  --load-balancers sms-gw-sim-load-balancer
```

This way the load balancer knows which instances are attached to it, in this case the instances belonging to the auto-scaling group. The instances are registered with the load balancer using the IP addresses associated with the instances. In the previous as-create-auto-scaling-group command two auto-scaling policies are used to determine how many instances are added or removed (change of capacity) when scaling out or scaling in, respectively.

These policies, attached to the auto-scaling group, just add or remove instances and the load balancer does not know that it should route new requests to the new created instance when scaling out. To make this possible a health check is required for the load balancer to know when to route new requests to a different instance. For example, a condition can be set declaring that when the number of healthy instances behind a load balancer goes down to two, two or more instances are launched.

Or, setting a condition to monitor the latency of the load balancer, and when the latency exceeds certain time period, such as three seconds, capacity is increased. As can be seen in Figure 6.5, the Load Generator instance sends the HTTP request to the load balancer. Subsequently, the load balancer determines to which instance the request is forwarded next. In this case, first only one instance is available behind the load balancer in which case the request is forwarded to this instance.
When the EC2 instances capacity scales out based on the configured “ChangeInCapacity” auto scaling policy described above, an instance is added making it two instances available behind the load balancer. The load balancer starts with the round robin distribution of the requests to the available instances behind the load balancer. The round robin distribution is straightforward, as the first request is routed to the first available instance behind the load balancer, while the second request is routed to the second instance, and so on.

Also health checks on registered EC2 instances guarantee that requests are forwarded to healthy instances. Elastic Load Balancing routinely checks the health of each registered Amazon EC2 instance based on the configurations specified, such as the protocol, port, URL, timeout, and interval specified. Instances that e.g. take too long to process requests because of lack of processing power or lack of memory are declared as unhealthy. In the scenario using Elastic Load balancing with a fixed number of instances behind the load balancer, traffic is no longer routed to unhealthy instances, but the load balancer spreads the load across the remaining healthy EC2 instances. In the scenario using Auto Scaling, when Auto Scaling determines that an instance is unhealthy, it terminates that instance and launches a new one. Using Amazon’s web-based user interface, the Amazon AWS Management Console, the health check for the load balancer was configured, as can be seen in the following figure:

![Figure 6.8: Health Check configuration](image)

The configured health check in this project, as can be seen in Figure 6.8, is described as follows: The load balancer sends a request to HTTP:8080/messaging-gateway/rest every 10 seconds. The load balancer allows 2 seconds for the web server to respond and if it does not get any response after 2 attempts, the instance is taken out of service. If the load balancer gets 10 successful responses, the instance is put back in service. Instances that are in service at the time of health check are marked healthy and the instances that are out of service at the time of health check are marked unhealthy.

### 6.4.5.2 Auto Scaling infrastructure architecture

The design of the cloud infrastructure using Amazon AWS is shown in Figure 6.9. It consists of four entities, namely the jmeter-ec2 script running on the local computer, the load generator instance, a load balancer and the amazon ec2 instances that are tested.
Using this architecture, performance tests were carried out, using the jmeter-ec2 shell script described in section 6.4.1, to test the availability of the HTTP messaging solution when using Amazon’s Auto Scaling service. The infrastructure used in this scenario on the Amazon AWS platform includes:

- One `m1.large` instance, as the load generator instance
- One `m3.xlarge` instance as the performance tested instance, which scales out and scales in as required. The maximum number of instances to scale out to was set to two instances (in figure 6.9 showing 1, 2 to N instances).

The Elastic Load Balancer was used as the load balancer, which is provided by Amazon and can easily be configured using the Amazon AWS Management Console.

### 6.4.6 Amazon Elastic Load Balancing

The Amazon ELB is a load balancing solution to distribute incoming application traffic across multiple Amazon EC2 instances. Elastic Load Balancing consists of two components, namely the load balancers and the controller service. The load balancers monitor the traffic and handle requests that come in through the Internet. The controller service monitors the load balancers, adding and removing load balancers as needed and verifying that the load balancers are functioning properly (AWS Documentation, 2012).

In the previous section an elastic load balancer is already used to forward incoming new requests to the newly created instance after scaling out. This cloud setup also uses a load balancer, but has a predetermined number of Amazon EC2 instances behind the load balancer. When the load balancer receives a request, it distributes the request to an instance based on round robin distribution. The various components of load balancing are shown in Figure 6.10.
In this figure two Availability Zones are used for the EC2 instances. Amazon Elastic Compute Cloud (Amazon EC2) provides the ability to launch EC2 instances in multiple Availability Zones. The load balancer can be configured to load balance incoming application traffic across multiple instances in a single Availability Zone or across multiple instances in several Availability Zones in the same region. When multiple instances are load balanced across two Availability Zones, and all the instances in the first Availability Zone become unhealthy, the load balancer will route traffic to the healthy instances in the other Availability Zone.

### 6.4.6.1 Amazon Elastic Load Balancing setup

The setup for this scenario includes. Another option would be using Amazon’s web-based user interface, the Amazon AWS Management Console.

- Setting up a load balancer with the command:

```sh
elb-create-lb sms-gw-sim-load-balancer
    --headers
    --listener "lb-port=80,instance-port=8080,protocol=HTTP"
    --availability-zones us-east-1a
```

- Using the Amazon AWS Management Console, a Health check is configured with the following parameters and default values.

```sh
elb-configure-healthcheck sms-gw-sim-load-balancer
    --interval 30
```
- Subsequently, the Amazon EC2 instances that need to be behind the load balancer are registered with the instance’s ID.

```
elb-register-instances-with-lb sms-gw-sim-load-balancer --instances i-802ffe77b
```

- Also a listener is created to forward requests to the 8080 port on which the HTTP messaging solution is listening.

```
elb-create-lb-listeners sms-gw-sim-load-balancer --listener "protocol=HTTP,lb-port=80,instance-port=8080,instance-protocol=HTTP"
```

When using HTTPS, an SSL server certificate must be installed on the load balancer. The load balancer uses the certificate to terminate and then decrypt requests before sending them to the back-end instances. Elastic Load Balancing uses AWS Identity and Access Management (IAM) to upload the certificate to the load balancer. When using SSL the following command is issued:

```
```
6.4.6.2 Amazon Elastic Load Balancing infrastructure architecture

Figure 6.11 shows the scenario to carry out performance tests to test the availability of the HTTP Messaging solution using the Elastic Load Balancing service of Amazon.

![Figure 6.11: HTTP Messaging solution using Amazon Elastic Load Balancing infrastructure](image)

The infrastructure used in this scenario on the Amazon AWS platform includes:

- One m1.large instance, as the load generator instance
- Two m3.xlarge instances as the performance-tested instances, which received round robin distributed requests (in figure 6.11 showing 3 instances).

The Elastic Load Balancer was used as the load balancer, which is a free resource provided by Amazon and can easily be configured using the Amazon AWS Management Console.

6.5 Design of HTTP messaging solution on the Amazon AWS platform

Next, the design of the HTTP messaging solution deployed on the Amazon platform is illustrated and described for all three scenarios. First the design of the dedicated instance on the Amazon AWS Platform is illustrated as a combination of figure 6.1 and figure 6.6, as can be seen in figure 6.12. Auto Scaling is a combination of Figure 6.1 and Figure 6.9 as can be seen in figure 6.13, while the design of the HTTP messaging solution deployed in the cloud using Elastic Load Balancing is a combination of Figure 6.1 and Figure 6.11 as can be seen in figure 6.14.

Proprietary instance

In figure 6.12 the design of the HTTP messaging solution on a proprietary server is shown. As a single Amazon EC2 instance was used to deploy the messaging solution, only two EC2 instances were required to setup this environment. The load generator instance directly sends HTTP requests to the proprietary instance using its IP address in the JMeter test plan. This way the response time of this environment can be measured, and the web server limit can be determined, by stress testing the instance.
Auto Scaling

In figure 6.13 the HTTP messaging solution in the cloud using Auto Scaling is illustrated where one instance is shown behind the Elastic Load Balancer. Using the Auto Scaling web service new instances are created that are added to the available instances for the load balancer.

Elastic Load Balancing

In figure 6.14 the HTTP messaging solution deployed in the cloud using load-balancing technology is illustrated. In this scenario two predetermined servers are made available behind the load balancer, each with the messaging solution deployed in tomcat.
Here two EC2 instances were created using the same AMI, each with Tomcat installed and the messaging solution deployed on Tomcat. Both these EC2 instances were registered with the load balancer. Based on health checks performed on each registered instance the load balancer determines to which instance the next requested is routed to. If Elastic Load Balancing finds an unhealthy instance, it stops sending traffic to the instance and reroutes traffic to healthy instances. When only healthy instances are available, the load balancer routes requests based on round robin distribution.

### 6.6 Deployment of HTTP messaging solution in the cloud

For the deployment of the six messaging components of the messaging solution, six Tomcat instances were created on the EC2 instance, i.e. six Tomcat folders were made available, e.g. `/usr/local/tomcat7-1/`, `/usr/local/tomcat7-2/` etc. The deployment procedure of each developed messaging component includes primarily in the Eclipse IDE exporting each web application as a Web application Archive (WAR) file, which is a JAR file used to distribute a collection of web resources that together form a web application. Subsequently, each exported WAR file is uploaded to an instance that has been booted from the AMI that is used as the template for new instances. This is done by using the new instance’s Public DNS address to secure-copy (SCP) each WAR file from the local machine to the instance. A shell script was created on the local machine, which was called in the command line to secure-copy each WAR file to the Amazon EC2 instance. When running the script the Amazon EC2 instance DNS name is provided as a parameter for secure copying of the WAR files to the proper instance. Note that for this the private key (smsgwkey.pem file) on the local machine is required:

```bash
#!/bin/bash
echo Please enter the Amazon EC2 instance URL...
read URL

sudo scp -i smsgwkey.pem billing-gateway.war ubuntu@$URL:~/
sudo scp -i smsgwkey.pem hlr.war ubuntu@$URL:~/
```
sudo scp -i msgwkey.pem messaging-gateway.war ubuntu@$URL:~/
sudo scp -i msgwkey.pem prepaid.war ubuntu@$URL:~/
sudo scp -i msgwkey.pem pro-charge.war ubuntu@$URL:~/
sudo scp -i msgwkey.pem rim.war ubuntu@$URL:~/

Once this is done successfully, a Secure Shell (SSH) connection is made to this instance, and all six WAR files are moved to the `webapps` directory of the Tomcat instance. For this moving of WAR files, a shell script was used with the following contents:

```
#!/bin/bash

mv /home/ubuntu/messaging-gateway.war /usr/local/tomcat7-1/webapps
mv /home/ubuntu/rim.war /usr/local/tomcat7-2/webapps
mv /home/ubuntu/prepaid.war /usr/local/tomcat7-3/webapps
mv /home/ubuntu/hlr.war /usr/local/tomcat7-4/webapps
mv /home/ubuntu/billing-gateway.war /usr/local/tomcat7-5/webapps
mv /home/ubuntu/pro-charge.war /usr/local/tomcat7-6/webapps
```

After this step, the instance has been updated with the new messaging solution. This instance is used to create a new AMI to be used as the template to start each new instance from. Every time a new AMI is created its AMI ID changes. This means (for the Auto Scaling scenario) that the current launch configuration must be deleted and a new one must be created with the new AMI ID, and afterwards the `as-update-auto-scaling-group my-asg --launch-configuration my-new-launch-config` command can be called to update the current auto scaling group.

### 6.7 Issues and recommendations for improvement

The following issues were encountered:

#### Issue #1
An encountered issue during the implementation and testing of the Auto Scaling web service was that the documentation of Amazon Auto Scaling did not clearly indicate to create a load balancer to route incoming requests to the newly created instance. The documentation only specifies the steps required to scale out or scale in, and what happens next is to the developer to figure out how to route the messages to the newly created instance. The required step was the creation of an Elastic Load Balancer provided by Amazon, which routes requests to available instances behind the load balancer. After scaling takes place and an additional instance is added to the auto scaling group, incoming requests are routed to both instances behind the load balancer based on round robin distribution, as described in section 6.4.5.1.

#### Issue #2
In the Auto Scaling scenario firstly tests were carried out with a CloudWatch alarm based on the CPU utilization metric, as shown above. This was because a general-purpose instance type with only 2 vCPUs was used, as described in section 6.4.3. Instances with 2 vCPUs resulted in a lot of processing power being used, giving the impression that the system was functioning correctly and causing the processing to be stressed. This however resulted in many test errors in JMeter causing tests to never finalize without errors. The intention was always to process all requests without any errors from the beginning of the test, until the scenario cannot handle any more requests, because of out-of-memory errors or errors caused by a lack of processing power.
Based on tests carried out in the first scenario using instances with 4 vCPUs, which is described in section 7.1.4, it was determined that the messaging solution did not stress the processor, but instead kept consuming memory till all memory was out, causing tomcat instances to be killed by the operating system. Hence, a CloudWatch alarm that is associated with a memory-utilization metric was required. By default, CloudWatch does not collect data about memory usage on an EC2 instance. Therefore, a custom metric was setup to monitor memory usage. This metric is subsequently attached to an alarm, and the auto scaling policy is based on that alarm. Two installations were tried out, including the Amazon CloudWatch Monitoring Scripts for Linux (AWS Documentation, 2010), and the second one described in section 6.4.5.1. The process to install the first custom metric is described briefly in the following. The Amazon CloudWatch Monitoring Scripts for Linux are sample Perl scripts that make it possible to report memory, swap, and disk space utilization metrics for an Amazon EC2 instance, and it consists of the following three scripts:

- The CloudWatchClient.pm, which is a shared Perl module that simplifies calling Amazon CloudWatch from other scripts.
- mon-put-instance-data.pl, which collects system metrics on an Amazon EC2 instance (memory, swap, disk space utilization) and sends them to Amazon CloudWatch.
- mon-get-instance-stats.pl, which queries Amazon CloudWatch and displays the most recent utilization statistics for the EC2 instance on which this script is executed.

These files are added to a folder on the instance running Ubuntu and subsequently a cron job is scheduled to execute these scripts after a given time interval. In this case the mon-put-instance-data.pl script was used to collect memory utilization and send it to Amazon CloudWatch. An issue encountered with this solution was the following. When setting up the Auto Scaling environment, using the steps listed in section 6.4.5.1, based on an AMI that was created using an Ubuntu OS that includes this script installation, the instance ID of the instance that was previously used to create the AMI, was always sent as the ID belonging to the measured memory utilization values, and NOT the ID of the currently running instance in the auto scaling group. This was also the case with the new instance that was added when scaling occurred. Because of this issue this solution was abandoned and the second solution was used, which is described in section 6.4.5.1.

**Recommendation # 1**

One recommendation includes designing the messaging solution in the cloud, for testing purposes, with each application that actually needs it, such as the MessagingGateway, ProCharge or BillingGateway (see section 7.1.2) running on a separate server or EC2 instance. These components will then be deployed separately on different servers or EC2 instances, this way creating a more realistic scenario. This way performances tests can be executed that may provide more accurate results.

**Recommendation # 2**

A recommendation includes carrying out the same tests on smaller instance types, such as the c3.xlarge instance type, which has the same features but with less memory. This can be done in combination with recommendation number 1, this way simulating a scenario closer to the real world, consisting of cheaper instance types (compared to the m3.xlarge) but with more instances, one for each selected messaging solution component, and for each instance a configuration that matches the weight of the messaging component (see section 7.1.2 for an idea of the JVM heap size distribution for these messaging components).
Chapter 7
Performance test of HTTP-based SMS messaging solution

In this chapter the performance tests carried out in the given three scenarios, i.e. using a proprietary server, using the Amazon cloud with the Auto Scaling service, and using the Amazon Elastic Load Balancing service, are elaborated. Besides performance tests to determine the response time of the messaging service using different computing scenarios, in addition the same test plans were used to perform stress tests to determine each scenario’s robustness in terms of extreme load.

Chapter 7 is organized as follows: the methodology used for the performance tests is described in section 7.1.1, followed by a description of the performance test plan used in all three scenarios in section 7.1.2. Section 7.1.3 described the performance test execution environment, including an overview of additional tools, certain tasks and configuration required to guarantee a consistent execution of performance tests. The performance tests carried out on a proprietary instance, is described in section 7.1.4. Section 7.1.5 describes the performance test performed in the cloud using Amazon Auto Scaling service and documents the test results. In section 7.1.6 the performance tests executed in the cloud using Amazon Elastic Load Balancing service are described with documented test results, and is followed by an evaluation of the executed performance tests in section 7.1.7. Section 7.2 elaborates on the comparison of results and is divided in sub-section 7.2.1, which describes the performance comparison methodology used, followed by the comparison of results in section 7.2.2. Section 7.2.3 finalizes the chapter with an evaluation of the comparison of the results. Section 7.3 provides an evaluation of the costs involved for each of the three scenarios, while section 7.4 finalizes the chapter with the issues encountered, and recommendations for improvement.

7.1 Performance test
The Amazon cloud platform is used to carry out performance tests (described in section 7.1.2) in all three scenarios, which are described in sections 7.1.4, 7.1.5 and 7.1.6.

7.1.1 Performance test methodology
Following a proper methodology guarantees a successful performance test project. It is important that the same methodology is followed on all projects. In this case the performance test project refers to the performance test carried out in a particular scenario. The methodology used for performance testing is defined by Johann du Plessis (Plessis, 2008), whom is a senior technical test consultant working for Micro to Mainframe, located in Johannesburg, South Africa. After carrying out many performance tests projects, the following performance testing methodology was defined, consisting of six phases:

<table>
<thead>
<tr>
<th>Phase</th>
<th>Deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Project assessment</td>
<td>Assessment report</td>
</tr>
<tr>
<td>2 - Planning</td>
<td>Test plan</td>
</tr>
<tr>
<td>3 - Scripting</td>
<td>Test scripts</td>
</tr>
<tr>
<td>4 - Test execution</td>
<td>Test scenarios, test results</td>
</tr>
<tr>
<td>5 - Results analysis</td>
<td>Results summary</td>
</tr>
<tr>
<td>6 - Reporting</td>
<td>Performance test report and presentation</td>
</tr>
</tbody>
</table>

Table 7.1: Performance test methodology
In this thesis the performance tests were carried out based on this methodology, but excluding the delivery of the six deliverables. The methodology was used as a guide to carry out the tests. The deliverables are elaborated in this chapter, and are not delivered as separate documents. These six phases are described in the following sub-sections and an indication is given of the significance of each phase in the performance tests carried out.

(1) Project assessment
In this phase is determined what the purpose of the performance test is for the particular environment, and if it can be achieved with available resources. It is a process of gathering information and specifying requirements for the performance test. In this project the three different scenarios (described in section 1.2) are analyzed and the resources used for testing are specified.

(2) Planning
Based on the information gathered during the project assessment phase, a performance test plan is created. The performance test plan contains all the details and acts as a checklist and reference for test execution.

(3) Scripting
In this phase the scripts required for executing the tests are written and executed to test their correctness. Here is where the testing actually starts. For this project the jmeter-ec2 script is used to upload the actual test script on the Amazon EC2 load generator instance, described in section 6.3. The test script is created using the Apache JMeter tool.

(4) Test execution
In this phase the performance test using the scripts defined in the ‘scripting’ phase are carried out. The tests are initially used as a means to determine the availability of services in all three scenarios. When carrying out load tests in all three mentioned scenarios, at some point (i.e. at a particular number of requests) the tested environment may not be able to handle the high load of requests anymore, leading to failures, such as out of memory errors. At this point the tested service’s availability is affected, i.e. the service is interrupted. In all three scenarios a specific number of tests are carried out, each test with the same number of requests for each scenario. This way the availability for each of these scenarios can be compared and determined. These same tests are finally used to compare the average response time of a request for service for all three scenarios. This way making it possible to determine which scenario has a faster performance. All response time results are monitored and documented, which are used in the next phase ‘results analysis’.

(5) Results analysis
This phase is the most challenging aspect of performance tests as the documented results for the different tested scenarios must be analyzed and compared to be able to draw concrete conclusions that answer the stated hypotheses and the given problem statement in Chapter 1, section 1.3.

Concerning the analysis of the availability of the messaging service for the different scenarios, the collected results are analyzed and compared, determining this way at what number of requests the service’s availability is affected and the service interrupted. The following hypotheses have been defined in section 1.2 concerning the availability of the messaging service:

- The availability of services at event-based peaks of traffic is not preserved using a dedicated server, when the capacity of the infrastructure is reached.
- The availability of services at event-based peaks of traffic is preserved using auto-scaling technology, when the capacity of the infrastructure is reached.
- The availability of services at event-based peaks of traffic is preserved using load-balancing technology, when the capacity of the infrastructure is reached.

To test these hypotheses a number of tests are carried out in each scenario, sending in each test a defined number of requests to the test environment until the test environment can handle no more requests, i.e. errors start to occur (e.g. out of memory errors). As only a number of tests are carried out it cannot be determined for each scenario at what point this particular scenario will handle no more requests, if the number of requests required for this to occur is out of the limit of the number of requests being sent. In such case, more research time is required to carry out many more tests till this point is reached for each scenario. This is a limitation of the method used in this thesis for carrying out the performance tests, as the time required for each test is considerable and more time is required to carry out a larger number of tests obtain better, more concise results.

Concerning the analysis of the average response time of a subscriber’s request for service, the collected results are used to perform t-tests to test the following stated hypothesis:

- The response time for a subscriber’s request for service improves when using auto-scaling technology instead of a dedicated server.
- The response time for a subscriber’s request for service improves when using load balancing instead of a dedicated server.
- The response time for a subscriber’s request for service improves when using load balancing instead of auto-scaling technology.

To test these hypotheses the response time required in each test to process a request sent by a subscriber is measured in each scenario using the jmeter-ec2 script and JMeter tool, described in section 6.4.2. Subsequently t-tests are carried out to determine if the average response time of one scenario improves compared to another scenario.

(6) Reporting

Typically this phase is used to report back on the findings and progress of the whole project. A full performance test report is delivered with a presentation to communicate the content of the report to the relevant people. In this project the findings are documented in this thesis in the following sections, and conclusions are drawn in Chapter 8.

These steps are not elaborated separately for the tests carried out in this project. However, these steps are elaborated together as a whole for each of the three scenarios described in following sections.

7.1.2 Performance test plan

This section provides a description of the performance test plan used in all three scenarios, which are described in the following sections. The test plan was created with the Apache JMeter tool (The Apache Software Foundation, 2013). First a Thread Group was created (Figure 7.1), which is the starting point of each test plan. This thread group element controls the number of threads JMeter will use to execute the test. The thread group consists of the following controls that can be modified: threads number, ramp-up period and number of times to execute the test. A thread can be seen as a connection to the server application. Each thread executes the test plan independently from other threads. By default, the thread group is configured to loop once through its elements. The ramp-up period is the period in seconds required to have all threads up and running. When executing a performance test with a ramp-up period of 0, meaning that all threads will be up and running at the same time, this usually causes a useless stress on a server generating bad response times, server
hangs or just total failure, because an application cannot handle all incoming requests at the same time. To prevent this a ramp-up period is used in a test plan.

To be able to send a high load of HTTP requests to an Amazon EC2 instance or to an Amazon Elastic Load Balancer, an HTTP Request sampler is required in JMeter. Samplers allow JMeter to send specific types of requests to a server. The HTTP request sampler lets you send an HTTP/HTTPS request to a web server, Figure 7.2. This sampler is used to specify the information required to send HTTP requests to the Amazon EC2 instances via the jmeter-ec2 shell script. In this figure the Server Name or IP used is the thesis.piloba.com DNS name of the Elastic Load Balancer, which was configured as a CNAME using an existing host domain name piloba.com. On the web-host’s configuration web page, the host name “thesis” was used as the CName alias pointing to the DNS name of the load balancer, i.e. sms-gw-sim-load-balancer-937342516.us-east-1.elb.amazonaws.com. For the scenario using a proprietary server, the Public DNS address of the instance is directly specified in the JMeter test plan. When using the Auto Scaling or Elastic Load Balancing services, the address of the load-balancer is defined in the JMeter test plan, in this case the domain name alias thesis.piloba.com was used.
In the JMeter test two random variables were added to the HTTP Request sampler to be used as parameters in the URI path. These parameters are source and the message_data. The message_data has the value BB_ON, while the source variable has the value ${sourcephonenumber}. To the HTTP Request sampler a Random Variable configuration element was added, as can be seen in Figure 7.3, where this sourcephonenumber output variable was defined to hold the telephone format 59990000000 of local phone numbers in Curaçao for testing purposes. The Os in the output format were replaced with values between 510000 and 690999, which are common phone numbers in Curaçao.

![Figure 7.3: Random Variables in JMeter](image)

The number of threads in the Thread Group interface is adapted as given in the following table, with corresponding ramp-up period (2.4% of the number of threads). The loop count remains 1, which means that each test is once in sequence. The number of threads chosen to be documented, as can be seen in the first column in Table 7.2, is based on a great number of tests that were carried out to find the number of threads that are relevant number.

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Ramp-up period</th>
<th>Loop count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2500</td>
<td>60</td>
<td>1</td>
</tr>
<tr>
<td>5000</td>
<td>120</td>
<td>1</td>
</tr>
<tr>
<td>10000</td>
<td>240</td>
<td>1</td>
</tr>
<tr>
<td>12500</td>
<td>300</td>
<td>1</td>
</tr>
<tr>
<td>15000</td>
<td>360</td>
<td>1</td>
</tr>
<tr>
<td>17500</td>
<td>420</td>
<td>1</td>
</tr>
<tr>
<td>20000</td>
<td>480</td>
<td>1</td>
</tr>
<tr>
<td>22500</td>
<td>540</td>
<td>1</td>
</tr>
<tr>
<td>25000</td>
<td>600</td>
<td>1</td>
</tr>
<tr>
<td>27500</td>
<td>660</td>
<td>1</td>
</tr>
<tr>
<td>30000</td>
<td>720</td>
<td>1</td>
</tr>
<tr>
<td>32500</td>
<td>780</td>
<td>1</td>
</tr>
<tr>
<td>35000</td>
<td>840</td>
<td>1</td>
</tr>
<tr>
<td>37500</td>
<td>900</td>
<td>1</td>
</tr>
<tr>
<td>40000</td>
<td>960</td>
<td>1</td>
</tr>
<tr>
<td>42500</td>
<td>1020</td>
<td>1</td>
</tr>
<tr>
<td>45000</td>
<td>1080</td>
<td>1</td>
</tr>
<tr>
<td>47500</td>
<td>1140</td>
<td>1</td>
</tr>
<tr>
<td>50000</td>
<td>1200</td>
<td>1</td>
</tr>
<tr>
<td>52500</td>
<td>1260</td>
<td>1</td>
</tr>
<tr>
<td>55000</td>
<td>1320</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table 7.2: Test plan thread number*
7.1.3 Performance test execution environment

This section provides an overview of additional tools, certain tasks and configuration required to guarantee a consistent execution of performance tests.

7.1.3.1 htop

During the performance tests htop was used to monitor the CPU and memory usage of each tested instance. htop is an interactive process viewer for Linux, which shows a frequently updated list of the processes running on a computer, normally ordered by the amount of CPU usage. Using htop the reasons for errors happening during tests in JMeter cloud be tracked, such as out of memory exceptions.

7.1.3.2 Tomcat tasks

After each test execution all running Tomcat instances were restarted to reset the heap memory usage of the previous test. This is done to guarantee a consistent test environment for each new test that is carried out. If this is not done, Out Of Memory exceptions occur causing subsequent tests to fail. The exact cause of an Out Of Memory Error is that a given Tomcat instance uses up all of the heap memory allocated to it, causing an application or server crash. The allocation of heap memory was done based on the relevance of each application in this project. As can be seen in Figure 6.1, the MessagingGateway component receives all requests.

This was done using a shell script with the following contents:

```bash
#!/bin/bash
sudo /etc/init.d/tomcat7-1 restart
sudo /etc/init.d/tomcat7-2 restart
sudo /etc/init.d/tomcat7-3 restart
sudo /etc/init.d/tomcat7-4 restart
sudo /etc/init.d/tomcat7-5 restart
sudo /etc/init.d/tomcat7-6 restart
```

Also every time a request is received by a messaging component each action executed in the Java application is logged in a file on the server. This causes memory to be used on the instance, and the more tests are executed the bigger these files become, causing a lot of memory to be wasted for the tests. For this reason, after each test these log files are emptied using the following shell script, this way having a fair execution tests, each with the same amount of memory. The /dev/null in the following script is a virtual file that can be written to and discards all the data written to it.

```bash
#!/bin/bash
cat /dev/null > /var/log/tomcat7/rim.log
cat /dev/null > /var/log/tomcat7/pro-charge.log
cat /dev/null > /var/log/tomcat7/prepaid.log
cat /dev/null > /var/log/tomcat7/messaging-gateway.log
cat /dev/null > /var/log/tomcat7/hlr.log
cat /dev/null > /var/log/tomcat7/billing-gateway.log
```

7.1.3.3 Performance test heap space distribution

For the proper execution of performance tests, appropriate heap memory allocation for each Tomcat instance is required. Default size of heap space in Java is 128MB. This means all six applications have 128MB of heap space allocated. The m3.xlarge instance has 15 GiB (14980 megabytes to be exact, using free -m command on the server) of memory available to distribute among all six applications.
A portion is saved for the operating system and the rest is used for heap space allocation. In this case 13 GB (13312 megabytes) of memory is used for the Java Virtual Machine heap space, while approximately 1.6 GB (1668 megabytes) is saved for the operating system. This is configured in each Tomcat instance’s bin folder on the EC2 instance in the calatina.sh file as follows:

```
JAVA_OPTS="-server -Xms128m -Xmx3328m -XX:PermSize=128m -XX:MaxPermSize=256m -Xincgc $JAVA_OPTS"
```

The Xms flag specifies the initial memory allocation pool for the Java Virtual Machine (JVM), i.e. the memory the JVM will be started with, and the Xmx flag specifies the maximum memory allocation pool for a JVM, i.e. the maximum memory the JVM is allowed to use. The flag -XX:PermSize is the initial size that will be allocated to the JVM during startup, while the specifies the maximum value the JVM will allocate up to -XX:MaxPermSize.

The distribution of the available memory on the EC2 instance was based on the frequency each instance was accessed. As can be seen in Figure 6.1 the MessagingGateway component is always accessed, and forwards requests to two components. Also the ProCharge component forwards requests to two components. The BillingGateway forwards incoming requests to just one component, while the PrePaid, Rim and Hlr components simply reply to each incoming request. This means that the MessagingGateway and the ProCharge components are allocated with the most heap memory, followed by the BillingGateway which needs memory to forward requests to the PrePaid component. The PrePaid, Rim and Hlr components are allocated with the least memory because of their simple function. The distribution is as follows:

<table>
<thead>
<tr>
<th>Tomcat instance name</th>
<th>Application name</th>
<th>Memory (in MB)</th>
<th>Memory (in GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tomcat7-1</td>
<td>MessagingGateway</td>
<td>3328</td>
<td>3.25</td>
</tr>
<tr>
<td>tomcat7-2</td>
<td>Rim</td>
<td>1331</td>
<td>1.30</td>
</tr>
<tr>
<td>tomcat7-3</td>
<td>Prepaid</td>
<td>1331</td>
<td>1.30</td>
</tr>
<tr>
<td>tomcat7-4</td>
<td>Hlr</td>
<td>1331</td>
<td>1.30</td>
</tr>
<tr>
<td>tomcat7-5</td>
<td>BillingGateway</td>
<td>2662</td>
<td>2.60</td>
</tr>
<tr>
<td>tomcat7-6</td>
<td>ProCharge</td>
<td>3328</td>
<td>3.25</td>
</tr>
</tbody>
</table>

*Table 7.3: Heap space allocation*

Total heap size allocated = 13311 megabytes = 13 GB

### 7.1.3.4 Ubuntu Linux environment configuration

After carrying out a large number of tests, at a certain point the following error occurred: “Cannot assign requested address”, without the system memory or operating system begin stressed. This error occurred because at a certain point no more ports were available on the Linux system, based on configuration of default kernel parameters. These defaults values needed to be modified to make it possible to carry out tests with a large number of threads. or pthe number of threads sufficiently the error Ubuntu Linux required configuration regarding the maximum number of TCP/IP connections that can be opened. The default value was not sufficient for the processing of high loads of requests sent to a performance-tested instance. For this reason the following limits for the kernel parameters at runtime on the Ubuntu operating system were modified (Stackoverflow, 2014):

- `net.ipv4.ip_local_port_range` - the ephemeral port range was increased.
- `net.ipv4.tcp_fin_timeout` - the minimum time sockets will stay in the `TIME_WAIT` state, which is the time sockets remain unusable after being used once. This value is decreased from 60 to 5.
- `net.ipv4.tcp_tw_recycle` is by default disabled, which means that fast recycling of `TIME_WAIT` is disabled. This is enabled for fast recycling of connections.
- `net.ipv4.tcp_twReuse` is also by default disabled and means that it is not allowed to reuse `TIME_WAIT` sockets for new connections. This is enabled to allow the reuse of connections.
- `net.core.somaxconn`, which limits the maximum number of requests queued to a listen socket, is also increased to 1024.
- `txqueuelen`, the length of the transmit queue of the network card device is increased.
- `net.core.netdev_max_backlog`, which describes the maximum number of incoming packets that can be queued up for upper-layer processing, is increased to 5000.
- `net.ipv4.tcp_max_syn_backlog`, which is the maximum number of remembered connection requests, which still did not receive an acknowledgment from connecting client, is increased to 5120.

Also, increasing the `ulimit` maximum open file descriptors value to a much larger value helps in allowing the operating system to open more connections. Each socket connection is represented as a file descriptor, which is an integer value that uniquely represents an opened file in the operating system. In this project, the open file value was changed to 262144 on the Ubuntu system in the `/etc/security/limits.conf` configuration file as:

```
ubunto    soft    nofile    262144
ubunto    hard    nofile    262144
root      soft    nofile    262144
root      hard    nofile    262144
```

### 7.1.3.5 Performance test results format

The following is an example of results documented by the JMeter-ec2 plugin:

```
> 02:01:42: Generate Summary Results + 604 in 15s = 40.2/s Avg: 69 Min: 14 Max: 1496 Err: 0 (0.00%) Active: 3 Started: 3207 Finished: 3204 | host: ec2-50-19-184-187.compute-1.amazonaws.com
> 02:01:57: Generate Summary Results + 599 in 15.3s = 39.3/s Avg: 58 Min: 16 Max: 522 Err: 0 (0.00%) Active: 2 Started: 3805 Finished: 3803 | host: ec2-50-19-184-187.compute-1.amazonaws.com
> 02:02:12: Generate Summary Results + 598 in 15s = 39.9/s Avg: 84 Min: 14 Max: 1754 Err: 0 (0.00%) Active: 8 Started: 4409 Finished: 4401 | host: ec2-50-19-184-187.compute-1.amazonaws.com
> 02:02:13: Generate Summary Results = 4402 in 111s = 39.7/s Avg: 928 Min: 14 Max: 17869 Err: 0 (0.00%) | host: ec2-50-19-184-187.compute-1.amazonaws.com
>
> 02:02:13: [RUNNING TOTALS] total count: 4402, current avg: 84 (ms), average tps: 39.7 (p/sec), recent tps: 39.9 (p/sec), total errors: 0
```

The meaning of these lines is explained next:

- The "Generate Summary Results" is the name of the element.
- The "+" after the element name means that the line is a delta line, i.e. it shows the changes since the last output.
- The "+" after the element name means that the line is a totals line, i.e. it shows the running total.
- Entries in the JMeter log file also include time-stamps.
- The example "4402 in 111s = 39.7/s" means that there were 4402 samples recorded in 111 seconds, and that works out at 39.7 samples per second.
- The Avg (Average), Min (minimum) and Max (maximum) times are in milliseconds.
- "Err" means number of errors (also shown as percentage).

Note: all test results for all scenarios are included as separate files in the attached ZIP file.
7.1.4 Performance test using single proprietary server

For the scenario where a proprietary server is used, the performance tests were carried out on a single Amazon EC2 server, the m3.xlarge. The technical specifications of this instance are the same as the instances used for the Cloud computing scenarios described in sections 7.1.5 and 7.1.6. The purpose is to simulate an environment using available resources, in this case the Amazon cloud computing platform. Performance tests were carried out using the jmeter-ec2 script on the already described load generator instance. Running this script on the load generator instance, requests are sent to the EC2 instance used as the proprietary server. The JMeter test plan used for this scenario can be seen in the following figure:

![Figure 7.4: JMeter test plan proprietary server](image)

In this test plan the server DNS name and port numbers are directly used as values in JMeter, as the load generator instance running the jmeter-ec2 script must know to which instance to forward the requests. This JMeter test plan is created on the local computer and is carried out with the purpose to track the messaging solution’s performance.

The following table summarizes the results of all test plans that were carried out on the dedicated server, before the configuration in step 7.1.3.4 was done:

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Ramp-up Period (seconds)</th>
<th>Loop count</th>
<th>Overall average response time (milliseconds)</th>
<th>Transactions per second (tps)</th>
<th>Errors</th>
<th>Error Rate (in %)</th>
<th>Memory usage from 14980MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>120</td>
<td>1</td>
<td>844</td>
<td>39.6</td>
<td>0</td>
<td>0</td>
<td>8789</td>
</tr>
<tr>
<td>7500</td>
<td>180</td>
<td>1</td>
<td>422</td>
<td>38.3</td>
<td>0</td>
<td>0</td>
<td>9256</td>
</tr>
<tr>
<td>10000</td>
<td>240</td>
<td>1</td>
<td>354</td>
<td>38.3</td>
<td>0</td>
<td>0</td>
<td>11120</td>
</tr>
<tr>
<td>12500</td>
<td>300</td>
<td>1</td>
<td>342</td>
<td>38.1</td>
<td>0</td>
<td>0</td>
<td>11724</td>
</tr>
<tr>
<td>15000</td>
<td>360</td>
<td>1</td>
<td>224</td>
<td>37.7</td>
<td>0</td>
<td>0</td>
<td>12247</td>
</tr>
<tr>
<td>17500</td>
<td>420</td>
<td>1</td>
<td>203</td>
<td>37.2</td>
<td>0</td>
<td>0</td>
<td>12678</td>
</tr>
<tr>
<td>20000</td>
<td>480</td>
<td>1</td>
<td>236</td>
<td>36.6</td>
<td>0</td>
<td>0</td>
<td>12832</td>
</tr>
<tr>
<td>22500</td>
<td>540</td>
<td>1</td>
<td>186</td>
<td>36.1</td>
<td>0</td>
<td>0</td>
<td>13142</td>
</tr>
<tr>
<td>25000</td>
<td>600</td>
<td>1</td>
<td>153</td>
<td>35.2</td>
<td>0</td>
<td>0</td>
<td>13380</td>
</tr>
<tr>
<td>27500</td>
<td>660</td>
<td>1</td>
<td>148</td>
<td>34.4</td>
<td>0</td>
<td>0</td>
<td>13813</td>
</tr>
<tr>
<td>30000</td>
<td>720</td>
<td>1</td>
<td>149</td>
<td>33.6</td>
<td>1773</td>
<td>5.91</td>
<td>13671</td>
</tr>
<tr>
<td>32500</td>
<td>780</td>
<td>1</td>
<td>143</td>
<td>32.9</td>
<td>4273</td>
<td>13.15</td>
<td>13672</td>
</tr>
<tr>
<td>35000</td>
<td>840</td>
<td>1</td>
<td>134</td>
<td>32.1</td>
<td>6773</td>
<td>19.35</td>
<td>13681</td>
</tr>
<tr>
<td>37500</td>
<td>900</td>
<td>1</td>
<td>186</td>
<td>31.0</td>
<td>9273</td>
<td>24.73</td>
<td>13851</td>
</tr>
<tr>
<td>40000</td>
<td>960</td>
<td>1</td>
<td>125</td>
<td>30.3</td>
<td>11773</td>
<td>29.43</td>
<td>13742</td>
</tr>
</tbody>
</table>

Table 7.4: Proprietary instance test plan results reaching maximum number of open ports available
The red-colored rows indicate the tests that included errors, with corresponding number of errors. From these rows the maximum number of processed requests is calculated:

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Requests not processed</th>
<th>Total processed requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>30000</td>
<td>1773</td>
<td>28227</td>
</tr>
<tr>
<td>32500</td>
<td>4273</td>
<td>28227</td>
</tr>
<tr>
<td>35000</td>
<td>6773</td>
<td>28227</td>
</tr>
<tr>
<td>37500</td>
<td>9273</td>
<td>28227</td>
</tr>
<tr>
<td>40000</td>
<td>11773</td>
<td>28227</td>
</tr>
</tbody>
</table>

*Table 7.5: Proprietary server scenario total processed requests overview*

It can be noticed that the total number of processed requests, remains the same. This was used as the foundation for finding out what was precisely going on with these tests. Based on the error “Cannot assign requested address”, research was done to figure out that no out-of-memory errors (all available heap space for the Java virtual machine (JVM) was consumed on the instance) occurred, but no more addresses could be assigned to the application, i.e. no more ports or sockets were available for the creation of new connections on the server. The maximum number of available ports on this instance type was 28227 based on the default configuration on this instance. The solution for this is explained in section 7.1.3.4.

**Correct results**

This section provides the results of performance tests executed in the configured environment, as described in section 7.1.3.4. These tests resulted in out-of-memory errors. This was traced back in the available log files, e.g. messaging-gateway.log file, indicating that the “Connection refused” error occurred. This error is the result of a Tomcat instance that has been killed by the operating system, and is no longer available, thus the connection to this application is refused. In the system log (the `/var/log/syslog` file) the following value was displayed:

Mar 30 07:07:09 ip-10-146-187-162 kernel: [57668.318737] Out of memory: Kill process 6520 (java) score 167 or sacrifice child

This verified that the error occurring was an actual out-of-memory error, which resulted in killing a running Tomcat process, in this example process 6520.
Table 7.6: Proprietary instance test plan results reaching maximum allocated memory

The red-colored rows indicate the tests that resulted in out-of-memory errors, i.e. the total available heap space for the JVM was consumed on the instance. The error that’s is logged is the “Connection refused” error. When the operating system cannot allocate more memory, it kills a running process, in this case a Tomcat process. This makes a particular application running on Tomcat unavailable. A resource trying to create a connection with the unavailable application, results in a “Connection refused” error.

At approximately 36000 requests the server can no longer handle incoming requests resulting in the mentioned error. This number of requests is considered the peak of requests.

Using the table above the results are plotted in the following graph:

Figure 7.5: Proprietary server scenario test results graph

Using this graph it can be noticed that as x gets large, the y value approaches a value between 100 and 300 milliseconds approximately. From this graph it can be seen from approximately 125000 requests the graph starts to stabilize. Based on this only these values, which did not result in out-of-memory errors, are extracted with corresponding response time values, as can be seen in the following table:
Table 7.7: Proprietary server scenario test results sample

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Overall average response time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12500</td>
<td>264</td>
</tr>
<tr>
<td>15000</td>
<td>230</td>
</tr>
<tr>
<td>17500</td>
<td>202</td>
</tr>
<tr>
<td>20000</td>
<td>179</td>
</tr>
<tr>
<td>22500</td>
<td>198</td>
</tr>
<tr>
<td>25000</td>
<td>150</td>
</tr>
<tr>
<td>27500</td>
<td>138</td>
</tr>
<tr>
<td>30000</td>
<td>130</td>
</tr>
<tr>
<td>32500</td>
<td>217</td>
</tr>
<tr>
<td>35000</td>
<td>117</td>
</tr>
</tbody>
</table>

The results in table 7.7 are used in section 7.2 for testing the in section 1.2 stated hypotheses where using a proprietary server is compared with using auto-scaling technology and load balancing.

In this scenario using two proprietary servers involves starting up the Tomcat instances with the deployed application on the second instance that is running in standby mode. Once the first server fails, in this case because of out of memory, Corosync Pacemaker starts up the second instance in the cluster-like environment. The time required for the applications to be up and running is the time required for all Tomcat instances to start up.

7.1.5 Performance test in the cloud using Amazon Auto Scaling service

As described in section 6.4.5.1 the Auto Scaling scenario was setup. Using the jmeter-ec2 tool JMeter was installed on the load generator instance and the performance test plan, described in section 7.1.2, was executed, sending requests to a single instance behind the load balancer. The JMeter test plan used in this scenario is illustrated in the following figure, using the test.piloba.com DNS name as the DNS name for the Elastic Load Balancer:
As the memory usage of this instance reached 30% of the total instance memory, scaling must take place, adding one additional instance to the auto-scaling group. The 30% threshold was determined based on performance tests that were carried, and is explained in the sub-section that follows table 7.8. To make the scaling of instances possible two CloudWatch alarms were created based on the Memory usage metric, as described in section 6.4.5.1. One alarm that watches over the scale-in metric, and one alarm that watches over the scale-out metric, that were both specified when creating the auto scaling policy. In this project the `sms-sim-memory-high-alarm` and `sms-sim-memory-low-alarm` alarms were created as can be seen in the following figure.

![Figure 7.7: Auto Scaling CloudWatch alarms](image)

The `sms-sim-memory-high-alarm`, as seen in figure 7.8, consists of a threshold “UsedMemoryPercent is greater than or equal to 30 for 1 minute”, which means that if the memory usage of the instance is equal to 30% of the total instance memory, or remains above the threshold of 30% of the total instance memory, for 1 minute long, the scale-out auto scaling policy will be triggered by the alarm adding an instance to the group of instances behind the load balancer.

![Figure 7.8: CloudWatch sms-sim-memory-high-alarm details](image)
The sms-sim-memory-low-alarm, as seen in figure 7.9, checks if the memory usage of the instance is equal to 30\% of the total instance memory, or remains below the threshold of 30\% of the total instance memory, for 1 minute long, the scale-in auto scaling policy will be triggered by the alarm to scale-in, removing an instance from the group of instances behind the load balancer. As this alarm is fired when the memory usage is below 30\%, when an instance is started up, this alarm fires and its state remains in the ALARM state.

Figure 7.9: CloudWatch sms-sim-memory-low-alarm details

The following table shows the results of the executed tests:

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Ramp-up Period (seconds)</th>
<th>Loop count</th>
<th>Overall average response time (milliseconds)</th>
<th>Transactions per second (tps)</th>
<th>Errors</th>
<th>Error Rate (in %)</th>
<th>Memory usage instance</th>
<th>Memory usage auto-scaled instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2500</td>
<td>60</td>
<td>1</td>
<td>1144</td>
<td>39.3</td>
<td>0</td>
<td>0</td>
<td>4578</td>
<td>1108</td>
</tr>
<tr>
<td>5000</td>
<td>120</td>
<td>1</td>
<td>621</td>
<td>39.7</td>
<td>0</td>
<td>0</td>
<td>8852</td>
<td>1104</td>
</tr>
<tr>
<td>7500</td>
<td>180</td>
<td>1</td>
<td>427</td>
<td>38.3</td>
<td>0</td>
<td>0</td>
<td>9979</td>
<td>1122</td>
</tr>
<tr>
<td>10000</td>
<td>240</td>
<td>1</td>
<td>342</td>
<td>38.3</td>
<td>0</td>
<td>0</td>
<td>11376</td>
<td>1105</td>
</tr>
<tr>
<td>12500</td>
<td>300</td>
<td>1</td>
<td>245</td>
<td>38.1</td>
<td>0</td>
<td>0</td>
<td>11659</td>
<td>1138</td>
</tr>
<tr>
<td>15000</td>
<td>360</td>
<td>1</td>
<td>242</td>
<td>37.7</td>
<td>0</td>
<td>0</td>
<td>12123</td>
<td>2485</td>
</tr>
<tr>
<td>17500</td>
<td>420</td>
<td>1</td>
<td>235</td>
<td>37.2</td>
<td>0</td>
<td>0</td>
<td>12289</td>
<td>1166</td>
</tr>
<tr>
<td>20000</td>
<td>480</td>
<td>1</td>
<td>192</td>
<td>36.0</td>
<td>0</td>
<td>0</td>
<td>12437</td>
<td>1151</td>
</tr>
<tr>
<td>22500</td>
<td>540</td>
<td>1</td>
<td>159</td>
<td>35.9</td>
<td>0</td>
<td>0</td>
<td>11256</td>
<td>5323</td>
</tr>
<tr>
<td>25000</td>
<td>600</td>
<td>1</td>
<td>171</td>
<td>35.2</td>
<td>0</td>
<td>0</td>
<td>12932</td>
<td>2645</td>
</tr>
<tr>
<td>27500</td>
<td>660</td>
<td>1</td>
<td>141</td>
<td>34.4</td>
<td>0</td>
<td>0</td>
<td>11256</td>
<td>5567</td>
</tr>
<tr>
<td>30000</td>
<td>720</td>
<td>1</td>
<td>133</td>
<td>33.6</td>
<td>0</td>
<td>0</td>
<td>12086</td>
<td>4161</td>
</tr>
<tr>
<td>32500</td>
<td>780</td>
<td>1</td>
<td>156</td>
<td>32.9</td>
<td>0</td>
<td>0</td>
<td>11271</td>
<td>5471</td>
</tr>
<tr>
<td>35000</td>
<td>840</td>
<td>1</td>
<td>126</td>
<td>32.0</td>
<td>0</td>
<td>0</td>
<td>11891</td>
<td>4633</td>
</tr>
<tr>
<td>37500</td>
<td>900</td>
<td>1</td>
<td>153</td>
<td>30.4</td>
<td>0</td>
<td>0</td>
<td>11331</td>
<td>5621</td>
</tr>
<tr>
<td>40000</td>
<td>960</td>
<td>1</td>
<td>121</td>
<td>30.3</td>
<td>0</td>
<td>0</td>
<td>12001</td>
<td>5367</td>
</tr>
<tr>
<td>42500</td>
<td>1020</td>
<td>1</td>
<td>109</td>
<td>29.6</td>
<td>0</td>
<td>0</td>
<td>11384</td>
<td>6212</td>
</tr>
<tr>
<td>45000</td>
<td>1080</td>
<td>1</td>
<td>117</td>
<td>29.4</td>
<td>0</td>
<td>0</td>
<td>12522</td>
<td>5685</td>
</tr>
<tr>
<td>47500</td>
<td>1140</td>
<td>1</td>
<td>166</td>
<td>28.0</td>
<td>0</td>
<td>0</td>
<td>12106</td>
<td>6457</td>
</tr>
<tr>
<td>50000</td>
<td>1200</td>
<td>1</td>
<td>103</td>
<td>26.7</td>
<td>0</td>
<td>0</td>
<td>12520</td>
<td>5840</td>
</tr>
<tr>
<td>52500</td>
<td>1260</td>
<td>1</td>
<td>99</td>
<td>26.0</td>
<td>0</td>
<td>0</td>
<td>12567</td>
<td>6054</td>
</tr>
<tr>
<td>55000</td>
<td>1320</td>
<td>1</td>
<td>118</td>
<td>26.7</td>
<td>0</td>
<td>0</td>
<td>13045</td>
<td>6374</td>
</tr>
</tbody>
</table>

Table 7.8: Auto Scaling test plan results reaching maximum allocated memory
From 2500 requests to 20000 requests scaling takes place, adding one instance to which the load balancer must distribute requests. When this instance is added it is registered with the load balancer, and starts up. While starting up and passing through all status checks carried out by Amazon, this instance remains in the OutOfService state behind the load balancer. When all status checks finish successfully the instance is in the InService state behind the load balancer, and can start receiving requests. This whole process takes time, while the first instance keeps receiving requests, and thus the memory usage keeps increasing. For this reason the 30% threshold was chosen, to have enough time for the booting and status checks of the newly booted instance to be ready, before the messaging solution on the first instance crashes because of an out-of-memory error. An example of this booting and status checks process is given in the following:

- At 20.51 the test is started.
- At 20.53 the low alarm’s state changes from ALARM to OK, which means that the system’s memory usage reached 30%.
- At 20.55 the high alarm is fired and a new instance is registered with the load balancer and started up. The status of the instance is OutOfService for the load balancer, and the instance’s status-check state remains in Initializing.
- At 21:02 the new instance behind the load balancer’s state changes to InService and starts receiving requests.

It takes approximately 7 minutes for a new instance to start up and be ready to receive requests. During these 7 minutes the first instance keeps receiving requests and must be able to handle and process them.

The results in the previous table are plotted in the following graph based on the number of threads and the response time measured:

Figure 7.10: Auto Scaling test plan results response time graph
Using this graph it can be noticed that as $x$ gets large, the $y$ value approaches a value between 100 and 250 milliseconds approximately. From this graph it can be seen from approximately 12500 requests the graph starts to stabilize. Based on this only these values are extracted with corresponding response time values, as can be seen in the following table:

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Overall average response time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12500</td>
<td>245</td>
</tr>
<tr>
<td>15000</td>
<td>242</td>
</tr>
<tr>
<td>17500</td>
<td>235</td>
</tr>
<tr>
<td>20000</td>
<td>192</td>
</tr>
<tr>
<td>22500</td>
<td>159</td>
</tr>
<tr>
<td>25000</td>
<td>171</td>
</tr>
<tr>
<td>27500</td>
<td>141</td>
</tr>
<tr>
<td>30000</td>
<td>133</td>
</tr>
<tr>
<td>32500</td>
<td>156</td>
</tr>
<tr>
<td>35000</td>
<td>126</td>
</tr>
<tr>
<td>37500</td>
<td>153</td>
</tr>
<tr>
<td>40000</td>
<td>121</td>
</tr>
<tr>
<td>42500</td>
<td>109</td>
</tr>
<tr>
<td>45000</td>
<td>117</td>
</tr>
<tr>
<td>47500</td>
<td>166</td>
</tr>
<tr>
<td>50000</td>
<td>103</td>
</tr>
<tr>
<td>52500</td>
<td>99</td>
</tr>
<tr>
<td>55000</td>
<td>118</td>
</tr>
</tbody>
</table>

*Table 7.9: Auto Scaling scenario test results sample*

The results in table 7.9 are used in section 7.2 for testing the in section 1.2 stated hypotheses where using auto-scaling technology is compared with using a proprietary server and load balancing.

### 7.1.6 Performance test in the cloud using Amazon Elastic Load Balancing service

In this scenario two methods were tested to make load balancing performance testing possible. The first method includes the use of the Elastic Load Balancing method provided by Amazon AWS, while the second method includes a separate Amazon EC2 instance with Apache web server installed that acts as the load balancer. The choice was made to use a separate load balancer as more control is provided to the user to manage the load balancer. During tests it was noticed that at certain moments no requests were sent from the JMeter client to the load balancer. At that point it could not be figured out where the problem was, and it was opted to use a custom created load balancer on a separate instance using apache2. These two methods are described in the following sections.

#### 7.1.6.1 Elastic Load Balancing

As described in section 6.4.6.1 the scenario using Elastic Load Balancing was setup. Using the jmeter-ec2 tool JMeter was installed on the load generator instance and the performance test plan, described in section 7.1.2, was executed, distributing requests to two instances behind the load balancer. The JMeter test plan used in this scenario is illustrated in the following figure:
Figure 7.11: JMeter test plan Elastic Load Balancing scenario

The tests results are listed in the following table:

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Ramp-up Period (seconds)</th>
<th>Loop count</th>
<th>Overall average response time (milliseconds)</th>
<th>Transactions per second (tps)</th>
<th>Errors</th>
<th>Error Rate (in %)</th>
<th>Memory usage instance 1</th>
<th>Memory usage instance 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2500</td>
<td>60</td>
<td>1</td>
<td>465</td>
<td>39.4</td>
<td>0</td>
<td>0</td>
<td>2903</td>
<td>2885</td>
</tr>
<tr>
<td>5000</td>
<td>120</td>
<td>1</td>
<td>248</td>
<td>39.8</td>
<td>0</td>
<td>0</td>
<td>5053</td>
<td>5012</td>
</tr>
<tr>
<td>7500</td>
<td>180</td>
<td>1</td>
<td>159</td>
<td>39.1</td>
<td>0</td>
<td>0</td>
<td>5410</td>
<td>5425</td>
</tr>
<tr>
<td>10000</td>
<td>240</td>
<td>1</td>
<td>144</td>
<td>38.4</td>
<td>0</td>
<td>0</td>
<td>5990</td>
<td>5924</td>
</tr>
<tr>
<td>12500</td>
<td>300</td>
<td>1</td>
<td>101</td>
<td>37.9</td>
<td>0</td>
<td>0</td>
<td>6248</td>
<td>6214</td>
</tr>
<tr>
<td>15000</td>
<td>360</td>
<td>1</td>
<td>97</td>
<td>37.4</td>
<td>0</td>
<td>0</td>
<td>6410</td>
<td>6421</td>
</tr>
<tr>
<td>17500</td>
<td>420</td>
<td>1</td>
<td>93</td>
<td>36.9</td>
<td>0</td>
<td>0</td>
<td>6708</td>
<td>6658</td>
</tr>
<tr>
<td>20000</td>
<td>480</td>
<td>1</td>
<td>77</td>
<td>36.5</td>
<td>0</td>
<td>0</td>
<td>6783</td>
<td>6730</td>
</tr>
<tr>
<td>22500</td>
<td>540</td>
<td>1</td>
<td>69</td>
<td>36.1</td>
<td>0</td>
<td>0</td>
<td>7012</td>
<td>6887</td>
</tr>
<tr>
<td>25000</td>
<td>600</td>
<td>1</td>
<td>61</td>
<td>35.2</td>
<td>0</td>
<td>0</td>
<td>7068</td>
<td>7101</td>
</tr>
<tr>
<td>27500</td>
<td>660</td>
<td>1</td>
<td>67</td>
<td>34.6</td>
<td>0</td>
<td>0</td>
<td>7283</td>
<td>7215</td>
</tr>
<tr>
<td>30000</td>
<td>720</td>
<td>1</td>
<td>58</td>
<td>33.8</td>
<td>0</td>
<td>0</td>
<td>7463</td>
<td>7245</td>
</tr>
<tr>
<td>32500</td>
<td>780</td>
<td>1</td>
<td>61</td>
<td>32.6</td>
<td>0</td>
<td>0</td>
<td>7414</td>
<td>7458</td>
</tr>
<tr>
<td>35000</td>
<td>840</td>
<td>1</td>
<td>60</td>
<td>32.3</td>
<td>0</td>
<td>0</td>
<td>7593</td>
<td>7412</td>
</tr>
<tr>
<td>37500</td>
<td>900</td>
<td>1</td>
<td>60</td>
<td>31.1</td>
<td>0</td>
<td>0</td>
<td>7693</td>
<td>7454</td>
</tr>
<tr>
<td>40000</td>
<td>960</td>
<td>1</td>
<td>68</td>
<td>30.4</td>
<td>0</td>
<td>0</td>
<td>7568</td>
<td>7831</td>
</tr>
<tr>
<td>42500</td>
<td>1020</td>
<td>1</td>
<td>53</td>
<td>29.8</td>
<td>0</td>
<td>0</td>
<td>7911</td>
<td>7950</td>
</tr>
<tr>
<td>45000</td>
<td>1080</td>
<td>1</td>
<td>68</td>
<td>29.2</td>
<td>0</td>
<td>0</td>
<td>8470</td>
<td>8063</td>
</tr>
<tr>
<td>47500</td>
<td>1140</td>
<td>1</td>
<td>80</td>
<td>28.1</td>
<td>0</td>
<td>0</td>
<td>8490</td>
<td>8506</td>
</tr>
<tr>
<td>50000</td>
<td>1200</td>
<td>1</td>
<td>62</td>
<td>27.8</td>
<td>0</td>
<td>0</td>
<td>8487</td>
<td>8383</td>
</tr>
<tr>
<td>52500</td>
<td>1260</td>
<td>1</td>
<td>68</td>
<td>27.4</td>
<td>0</td>
<td>0</td>
<td>9169</td>
<td>8501</td>
</tr>
<tr>
<td>55000</td>
<td>1320</td>
<td>1</td>
<td>72</td>
<td>26.6</td>
<td>0</td>
<td>0</td>
<td>8811</td>
<td>9173</td>
</tr>
</tbody>
</table>

Table 7.10: Amazon Elastic Load Balancer test results

The results in the previous table are plotted in the following graph based on the number of threads and the response time measured:
Using this graph it can also be noticed that as \( x \) gets large, the \( y \) value approaches a value between 50 and 100 milliseconds approximately. From this graph it can be seen from approximately 12500 requests that the graph starts to steady. Based on this, only these values are extracted with corresponding response time values, as can be seen in the following table:

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Overall average response time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12500</td>
<td>101</td>
</tr>
<tr>
<td>15000</td>
<td>97</td>
</tr>
<tr>
<td>17500</td>
<td>93</td>
</tr>
<tr>
<td>20000</td>
<td>77</td>
</tr>
<tr>
<td>22500</td>
<td>69</td>
</tr>
<tr>
<td>25000</td>
<td>61</td>
</tr>
<tr>
<td>27500</td>
<td>67</td>
</tr>
<tr>
<td>30000</td>
<td>58</td>
</tr>
<tr>
<td>32500</td>
<td>61</td>
</tr>
<tr>
<td>35000</td>
<td>60</td>
</tr>
<tr>
<td>37500</td>
<td>60</td>
</tr>
<tr>
<td>40000</td>
<td>68</td>
</tr>
<tr>
<td>42500</td>
<td>53</td>
</tr>
<tr>
<td>45000</td>
<td>68</td>
</tr>
<tr>
<td>47500</td>
<td>80</td>
</tr>
<tr>
<td>50000</td>
<td>62</td>
</tr>
<tr>
<td>52500</td>
<td>68</td>
</tr>
<tr>
<td>55000</td>
<td>72</td>
</tr>
</tbody>
</table>

Table 7.11: Elastic Load Balancer scenario test results sample

The results in table 7.11 are used in section 7.2 for testing the in section 1.2 stated hypotheses where using load balancing is compared with using a proprietary server and auto-scaling technology.
7.1.6.2 Custom load balancer using Apache webserver

The second method tested was using a separate Amazon EC2 instance, with Apache2 installed as the load balancer. Apache is well known as a web server on Linux systems. Web Servers are used to serve web pages that are requested by client computers. These web pages are requested using Uniform Resource Locators (URLs) that point to the web server using the Fully Qualified Domain Name (FQDN) and a path to the required resource. In this HTTP is used as the protocol to simulate a messaging solution to send text messages from point to point to the other. HTTP is the most commonly used protocol to transfer web pages. Apache can be configured to act as a proxy forwarding requests to defined resources, in this case the two Amazon EC2 instances behind the load balancer. This way a load balancer is created using a single Amazon EC2 instance. The installation was done by updating the /etc/apache2/sites-available/default configuration file with the following contents to make the round-robin distribution between the two instances behind the load balancer possible.

```xml
<Proxy balancer://smsgw>
  BalancerMember http://10.146.228.151:8080/ keepalive=On disablereuse=On max=100 ttl=3600 retry=0 timeout=3600
  BalancerMember http://10.146.237.36:8080/ keepalive=On disablereuse=On max=100 ttl=3600 retry=0 timeout=3600
  ProxySet lbmethod=byrequests
</Proxy>

ProxyPass /performance-test balancer://smsgw
```

The given /performance-test value in this configuration file is used in JMeter to be able to connect with the instances behind the load balancer. The instances behind the load balancer in this example have the IP addresses 10.146.228.151 and 10.146.237.36. Requests are sent from the JMeter test plan to the address of the EC2 instance containing the Apache2 installation with the configured default port number (80).

The performance test using the separate EC2 instance for the load balancer is as follows, where the DNS name of the load balancer instance is used and the configured path as described above:

![Figure 7.13: Custom load balancer performance test plan](image)

The following table shows the results of tests executed using the custom load balancer:

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Ramp-up Period (seconds)</th>
<th>Loop count</th>
<th>Overall average response time (milliseconds)</th>
<th>Transactions per second (tps)</th>
<th>Errors</th>
<th>Error Rate (in %)</th>
<th>Total memory usage instance 1</th>
<th>Total memory usage instance 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2500</td>
<td>60</td>
<td>1</td>
<td>755</td>
<td>39.2</td>
<td>0</td>
<td>0</td>
<td>3830</td>
<td>3956</td>
</tr>
<tr>
<td>5000</td>
<td>120</td>
<td>1</td>
<td>232</td>
<td>39.8</td>
<td>0</td>
<td>0</td>
<td>4358</td>
<td>4467</td>
</tr>
<tr>
<td>7500</td>
<td>180</td>
<td>1</td>
<td>156</td>
<td>39.2</td>
<td>0</td>
<td>0</td>
<td>5711</td>
<td>5688</td>
</tr>
<tr>
<td>10000</td>
<td>240</td>
<td>1</td>
<td>121</td>
<td>38.5</td>
<td>0</td>
<td>0</td>
<td>5958</td>
<td>5914</td>
</tr>
<tr>
<td>12500</td>
<td>300</td>
<td>1</td>
<td>80</td>
<td>38.2</td>
<td>0</td>
<td>0</td>
<td>6145</td>
<td>6121</td>
</tr>
<tr>
<td>15000</td>
<td>360</td>
<td>1</td>
<td>84</td>
<td>37.3</td>
<td>0</td>
<td>0</td>
<td>6405</td>
<td>6379</td>
</tr>
<tr>
<td>17500</td>
<td>420</td>
<td>1</td>
<td>68</td>
<td>36.9</td>
<td>0</td>
<td>0</td>
<td>6651</td>
<td>6658</td>
</tr>
<tr>
<td>20000</td>
<td>480</td>
<td>1</td>
<td>62</td>
<td>36.4</td>
<td>0</td>
<td>0</td>
<td>6693</td>
<td>6758</td>
</tr>
<tr>
<td>22500</td>
<td>540</td>
<td>1</td>
<td>61</td>
<td>35.9</td>
<td>0</td>
<td>0</td>
<td>6925</td>
<td>6923</td>
</tr>
<tr>
<td>25000</td>
<td>600</td>
<td>1</td>
<td>51</td>
<td>35.2</td>
<td>0</td>
<td>0</td>
<td>7083</td>
<td>7143</td>
</tr>
<tr>
<td>27500</td>
<td>660</td>
<td>1</td>
<td>54</td>
<td>34.4</td>
<td>0</td>
<td>0</td>
<td>7270</td>
<td>7243</td>
</tr>
<tr>
<td>30000</td>
<td>720</td>
<td>1</td>
<td>55</td>
<td>33.8</td>
<td>0</td>
<td>0</td>
<td>7389</td>
<td>7325</td>
</tr>
<tr>
<td>32500</td>
<td>780</td>
<td>1</td>
<td>52</td>
<td>32.7</td>
<td>0</td>
<td>0</td>
<td>7435</td>
<td>7406</td>
</tr>
<tr>
<td>35000</td>
<td>840</td>
<td>1</td>
<td>49</td>
<td>32.1</td>
<td>0</td>
<td>0</td>
<td>7517</td>
<td>7576</td>
</tr>
<tr>
<td>37500</td>
<td>900</td>
<td>1</td>
<td>61</td>
<td>31.0</td>
<td>0</td>
<td>0</td>
<td>7574</td>
<td>7460</td>
</tr>
<tr>
<td>40000</td>
<td>960</td>
<td>1</td>
<td>48</td>
<td>30.3</td>
<td>0</td>
<td>0</td>
<td>7742</td>
<td>7767</td>
</tr>
<tr>
<td>42500</td>
<td>1020</td>
<td>1</td>
<td>50</td>
<td>29.3</td>
<td>0</td>
<td>0</td>
<td>7826</td>
<td>7846</td>
</tr>
<tr>
<td>45000</td>
<td>1080</td>
<td>1</td>
<td>49</td>
<td>29.1</td>
<td>0</td>
<td>0</td>
<td>8145</td>
<td>8132</td>
</tr>
<tr>
<td>47500</td>
<td>1140</td>
<td>1</td>
<td>51</td>
<td>28.4</td>
<td>0</td>
<td>0</td>
<td>8387</td>
<td>8453</td>
</tr>
<tr>
<td>50000</td>
<td>1200</td>
<td>1</td>
<td>49</td>
<td>27.7</td>
<td>0</td>
<td>0</td>
<td>8404</td>
<td>8403</td>
</tr>
<tr>
<td>52500</td>
<td>1260</td>
<td>1</td>
<td>46</td>
<td>27.1</td>
<td>0</td>
<td>0</td>
<td>8697</td>
<td>8634</td>
</tr>
<tr>
<td>55000</td>
<td>1320</td>
<td>1</td>
<td>50</td>
<td>26.9</td>
<td>0</td>
<td>0</td>
<td>8813</td>
<td>8759</td>
</tr>
</tbody>
</table>

Table 7.12: Custom load balancer test results

The results in the previous table are plotted in the following graph based on the number of threads and the response time measured:

![Graph showing overall average response time](image)

Figure 7.14: Custom load balancer response time test results plotted
Using this graph it can also be noticed that as $x$ gets large, the $y$ value approaches a value between 45 and 85 milliseconds approximately. From this graph it can be seen from approximately 12500 requests that the graph starts to steady. Based on this, only these values are extracted with corresponding response time values, as can be seen in the following table:

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Overall average response time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12500</td>
<td>80</td>
</tr>
<tr>
<td>15000</td>
<td>84</td>
</tr>
<tr>
<td>17500</td>
<td>68</td>
</tr>
<tr>
<td>20000</td>
<td>62</td>
</tr>
<tr>
<td>22500</td>
<td>61</td>
</tr>
<tr>
<td>25000</td>
<td>51</td>
</tr>
<tr>
<td>27500</td>
<td>54</td>
</tr>
<tr>
<td>30000</td>
<td>55</td>
</tr>
<tr>
<td>32500</td>
<td>52</td>
</tr>
<tr>
<td>35000</td>
<td>49</td>
</tr>
<tr>
<td>37500</td>
<td>61</td>
</tr>
<tr>
<td>40000</td>
<td>48</td>
</tr>
<tr>
<td>42500</td>
<td>50</td>
</tr>
<tr>
<td>45000</td>
<td>49</td>
</tr>
<tr>
<td>47500</td>
<td>51</td>
</tr>
<tr>
<td>50000</td>
<td>49</td>
</tr>
<tr>
<td>52500</td>
<td>46</td>
</tr>
<tr>
<td>55000</td>
<td>50</td>
</tr>
</tbody>
</table>

*Table 7.13: Custom load balancer scenario test results sample*

The results in table 7.13 are used in section 7.2 for testing the in section 1.2 stated hypotheses where using load balancing is compared with using a proprietary server and auto-scaling technology.

### 7.1.7 Performance tests evaluation

During the execution of the performance tests proper configuration of kernel parameters was required to acquire the expected accurate results. After carrying out a vast number of tests in all three scenarios it was noted that the error “Cannot assign requested address” was continuously thrown at almost the same time the memory reached its maximum using *htop* tool to monitor the memory usage. This resulted in the confusion that an out-of-memory error occurred, or at least that the “Cannot assign requested address” was actually caused because there was no more memory. This was not the case as more research was done and the tests were more thoroughly analyzed. This led to a lot of time being used in the execution of tests, until the right results, those documented in this thesis, were encountered.

### 7.2 Comparison of results

In this section initially the availability of the tested HTTP-based messaging service is evaluated and compared between the three tested scenarios. Finally the average response time results in each scenario are evaluated and compared to each other, this way determining which configuration performs faster under high loads, i.e. the stated hypotheses in section 1.2 are tested. The used methodologies to evaluate both the availability and performance of the messaging service are described in the following sub-section.
7.2.1 Comparison methodology

7.2.1.1 Availability comparison methodology
To determine the availability of messaging services the collected test results (in sections 7.1.4, 7.1.5 and 7.1.6) are compared in tabular form, this way making it possible to observe in which scenario the tests start failing, this way indicating that the service is no longer available. These results are based on the limited number of tests carried out. Performing more tests could lead to more accurate results for all three scenarios.

7.2.1.2 Performance comparison methodology
To determine if the means of the collected response time results in each scenario, in the previous sections 7.1.4, 7.1.5 and 7.1.6, are significantly different from each other, the t-test statistical method is used. The t-test is a statistical method used to test stated hypotheses. According to Statistically Significant Consulting LLC (2014), with the statistics t-test a p-value can be determined that indicates how likely test results could have been gotten by chance, if in fact the null hypothesis were true. The null hypothesis means that no relationship exists between variables. By convention, if there is less than 5% chance of getting the observed differences by chance, the null hypothesis is rejected and a statistically significant difference between the two groups has been found. According to WikiBooks (2013), the t-test is the most powerful parametric test for calculating the significance of a small sample mean. In this case this statistical method is a good alternative to test the stated hypotheses. In each scenario the same number of requests is being sent to the testing environment (i.e. the Amazon AWS), but under different conditions, i.e. using a different server configuration for each scenario. Hence, the paired sample t-test is used. The paired sample t-test is used determine whether there is a significant difference between the average values of the same measurement made under two different conditions (The Statistics Glossary, 2014).

7.2.2 Comparison of test results

7.2.2.1 Availability comparison
The following table shows the results obtained in sections 7.1.4, 7.1.5 and 7.1.6 for all three scenarios respectively:

<table>
<thead>
<tr>
<th>#</th>
<th>Proprietary (ms)</th>
<th>Auto-scaling (ms)</th>
<th>Amazon ELB Load balancing (ms)</th>
<th>Apache2 Load balancing (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1478</td>
<td>1144</td>
<td>465</td>
<td>755</td>
</tr>
<tr>
<td>2</td>
<td>591</td>
<td>621</td>
<td>248</td>
<td>232</td>
</tr>
<tr>
<td>3</td>
<td>706</td>
<td>427</td>
<td>159</td>
<td>156</td>
</tr>
<tr>
<td>4</td>
<td>461</td>
<td>342</td>
<td>144</td>
<td>121</td>
</tr>
<tr>
<td>5</td>
<td>264</td>
<td>245</td>
<td>101</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>230</td>
<td>242</td>
<td>97</td>
<td>84</td>
</tr>
<tr>
<td>7</td>
<td>202</td>
<td>235</td>
<td>93</td>
<td>68</td>
</tr>
<tr>
<td>8</td>
<td>179</td>
<td>192</td>
<td>77</td>
<td>62</td>
</tr>
<tr>
<td>9</td>
<td>198</td>
<td>159</td>
<td>69</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td>150</td>
<td>171</td>
<td>61</td>
<td>51</td>
</tr>
<tr>
<td>11</td>
<td>138</td>
<td>141</td>
<td>67</td>
<td>54</td>
</tr>
<tr>
<td>12</td>
<td>130</td>
<td>133</td>
<td>58</td>
<td>55</td>
</tr>
<tr>
<td>13</td>
<td>217</td>
<td>156</td>
<td>61</td>
<td>52</td>
</tr>
<tr>
<td>14</td>
<td>117</td>
<td>126</td>
<td>60</td>
<td>49</td>
</tr>
<tr>
<td>15</td>
<td>98</td>
<td>153</td>
<td>60</td>
<td>61</td>
</tr>
<tr>
<td>16</td>
<td>115</td>
<td>121</td>
<td>68</td>
<td>48</td>
</tr>
<tr>
<td>17</td>
<td>145</td>
<td>109</td>
<td>53</td>
<td>50</td>
</tr>
</tbody>
</table>
In this table it can be observed that the availability of the messaging service is not preserved when using a proprietary (or dedicated) server, while it is preserved in all three remaining cases. Performing a larger number of tests, which requires more time investment, may lead to different results for the three remaining cases. All the following three hypotheses can be concluded based on this result:

- The availability of services at event-based peaks of traffic is not preserved using a dedicated server, when the capacity of the infrastructure is reached.
- The availability of services at event-based peaks of traffic is preserved using auto-scaling technology, when the capacity of the infrastructure is reached.
- The availability of services at event-based peaks of traffic is preserved using load-balancing technology, when the capacity of the infrastructure is reached.

7.2.2.2 Performance comparison

In sections 7.1.4, 7.1.5 and 7.1.6 the test results for all three scenarios were described with corresponding results. The following three scenarios are being compared to each other:

1. SMS messaging solution deployed on a proprietary server,
2. SMS messaging solution deployed in the Amazon AWS cloud-computing platform using AutoScaling,
3. SMS messaging solution deployed using load balancing.
   • In this scenario two configurations were used:
     i. Amazon’s Elastic Load Balancing feature
     ii. Apache2 web server as the load balancer

Scenario 1 vs. scenario 2

The following table shows all the obtained response time results for the proprietary and load balancing scenarios and is followed by the procedure for testing the hypothesis given below using SPSS predictive analytics software (SPSS Software, 2014). In this table of the 22 tests carried out, 8 failed due to out-of-memory-errors as was described in section 7.1.4. Hence, only 14 results are considered useful to test the hypothesis in this section.
The following table shows only the results that are used in this section. As was described in section 7.1.4, only the results that did not result in out-of-memory errors are taken into consideration. From these results, based on the graph in figure 7.5 only those values that lie in the range where the graph stabilizes (10 test results) are used for testing the hypothesis with the matching auto-scaling results, as only these are considered the relevant values.

<table>
<thead>
<tr>
<th>#</th>
<th>Proprietary (ms)</th>
<th>Auto-scaling (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>179</td>
<td>192</td>
</tr>
<tr>
<td>9</td>
<td>198</td>
<td>159</td>
</tr>
<tr>
<td>10</td>
<td>150</td>
<td>171</td>
</tr>
<tr>
<td>11</td>
<td>138</td>
<td>141</td>
</tr>
<tr>
<td>12</td>
<td>130</td>
<td>133</td>
</tr>
<tr>
<td>13</td>
<td>217</td>
<td>156</td>
</tr>
<tr>
<td>14</td>
<td>117</td>
<td>126</td>
</tr>
<tr>
<td>15</td>
<td>98</td>
<td>153</td>
</tr>
<tr>
<td>16</td>
<td>115</td>
<td>121</td>
</tr>
<tr>
<td>17</td>
<td>145</td>
<td>109</td>
</tr>
<tr>
<td>18</td>
<td>97</td>
<td>117</td>
</tr>
<tr>
<td>19</td>
<td>548</td>
<td>166</td>
</tr>
<tr>
<td>20</td>
<td>163</td>
<td>103</td>
</tr>
<tr>
<td>21</td>
<td>188</td>
<td>99</td>
</tr>
<tr>
<td>22</td>
<td>92</td>
<td>118</td>
</tr>
</tbody>
</table>

Table 7.15: All test results response time proprietary vs. auto-scaling

In this section it is determined if the response time for a subscriber’s request for service improves when using auto-scaling technology instead of using a proprietary server. The hypothesis is:

- The response time for a subscriber’s request for service improves when using auto-scaling technology instead of a dedicated server.

- The null hypothesis and alternative hypothesis are defined as follows:
  \[ H_0: \mu_1 = \mu_2, \text{ i.e. } d = \mu_1 - \mu_2 = 0 \] where \( d \) is the mean value of the difference.
  \[ H_1: d > 0 \]

Where \( \mu_1 \) is the mean response time for a subscriber’s request for service when using a proprietary server, and \( \mu_2 \) is the mean response time for a subscriber’s request for service when using auto-scaling technology. \( \mu_2 \) has smaller value than \( \mu_1 \), i.e. improved response time for a subscriber’s request for service when using auto-scaling technology compared to using proprietary servers, thus their difference is greater than zero.
• This test has one-tailed test because the hypothesis involves the phrase "improves", i.e. an improvement is expected. One-tailed tests expect the effect to be in a certain direction.
• The α level is: \( \alpha = .05 \)

In SPSS when running the test the output viewer will appear with the results of the t-test, which consists of three main parts: descriptive statistics, the correlation between the pair of variables, and inferential statistics. To determine if the null hypothesis can be rejected the third part is used. The other two parts provide additional information if required.

The descriptive statistics figure (figure 7...) gives the mean, the number of tests that were carried out, the standard deviation from the mean and the standard error of the mean for each group.

**Paired Samples Statistics**

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>N</th>
<th>Std. Deviation</th>
<th>Std. Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proprietary</td>
<td>182.5000</td>
<td>10</td>
<td>44.0792</td>
<td>15.20398</td>
</tr>
<tr>
<td>AutoScaling</td>
<td>180.0000</td>
<td>10</td>
<td>45.92506</td>
<td>14.52278</td>
</tr>
</tbody>
</table>

**Figure 7.15: Descriptive statistics t-test proprietary vs. auto-scaling**

The next part of the output gives the correlation between the pair of variables and the p value for the correlation coefficient.

**Paired Samples Correlations**

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Correlation</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proprietary &amp; AutoScaling</td>
<td>10</td>
<td>.813</td>
<td>.004</td>
</tr>
</tbody>
</table>

**Figure 7.16: Correlation output of t-test proprietary vs. auto-scaling**

The last part of the output gives the inferential statistics, including the difference of the two means, the difference in their standard deviations from the mean and the difference of the standard errors of the means. Also the observed or calculated t value is given, followed by the degrees of freedom associated with the t-test and the significance "Sig. (2-tailed)", which gives the two-tailed p value associated with the test.

**Paired Samples Test**

<table>
<thead>
<tr>
<th></th>
<th>Paired Differences</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Std. Deviation</td>
<td>Std. Error Mean</td>
<td>95% Confidence Interval of the Difference</td>
<td>t</td>
<td>df</td>
<td>Sig. (2-tailed)</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 7.17: Inferential statistics t-test proprietary vs. auto-scaling**

The corresponding degrees-of-freedom df is 9 and the two-tailed p-value is .790. The observed or calculated t-test value (t_{obs}) is 0.274. Because t_{obs} is positive, significance is found in the positive one-tailed t-tests. The one-tailed t-test in the negative direction would not be significant, because \( \alpha \) was placed in the wrong tail. The difference \( d \) was predicted to be positive before the test was conducted and the t_{obs} value is in the same direction. For this reason this p-value \( p = .790 \) is divided in half, i.e. \( p = .790 / 2 = .395 \). p = .395, which is not less than or equal to \( .05 / 2 = .025 \). So, we fail to reject the null hypothesis \( H_0 \). This implies that there is insufficient evidence to conclude that the response time for a subscriber’s request for service improves when using auto-scaling technology instead of a proprietary server.

**Scenario 1 vs. scenario 3**
In this case two different configurations are used to carry out the tests comparing with the proprietary server scenario, one using the Amazon Elastic Load Balancer and the other using Apache2 load balancer. This section elaborates the testing of the hypothesis based on each configuration.

- **Using Amazon Elastic Load Balancer**

The following table shows the obtained response time results for the proprietary server and load balancing scenarios, using Amazon Elastic Load Balancing, and is followed by the procedure for testing the hypothesis given below using SPSS. In this table of the 22 tests carried out, 8 failed due to out-of-memory-errors as was described in section 7.1.4. Hence, only 14 results are considered useful to test the hypothesis in this section.

<table>
<thead>
<tr>
<th>#</th>
<th>Proprietary (ms)</th>
<th>Amazon ELB Load balancing (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1478</td>
<td>465</td>
</tr>
<tr>
<td>2</td>
<td>591</td>
<td>248</td>
</tr>
<tr>
<td>3</td>
<td>706</td>
<td>159</td>
</tr>
<tr>
<td>4</td>
<td>461</td>
<td>144</td>
</tr>
<tr>
<td>5</td>
<td>264</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>230</td>
<td>97</td>
</tr>
<tr>
<td>7</td>
<td>202</td>
<td>93</td>
</tr>
<tr>
<td>8</td>
<td>179</td>
<td>77</td>
</tr>
<tr>
<td>9</td>
<td>198</td>
<td>69</td>
</tr>
<tr>
<td>10</td>
<td>150</td>
<td>61</td>
</tr>
<tr>
<td>11</td>
<td>138</td>
<td>67</td>
</tr>
<tr>
<td>12</td>
<td>130</td>
<td>58</td>
</tr>
<tr>
<td>13</td>
<td>217</td>
<td>61</td>
</tr>
<tr>
<td>14</td>
<td>117</td>
<td>60</td>
</tr>
<tr>
<td>15</td>
<td>98</td>
<td>60</td>
</tr>
<tr>
<td>16</td>
<td>115</td>
<td>68</td>
</tr>
<tr>
<td>17</td>
<td>145</td>
<td>53</td>
</tr>
<tr>
<td>18</td>
<td>97</td>
<td>68</td>
</tr>
<tr>
<td>19</td>
<td>548</td>
<td>80</td>
</tr>
<tr>
<td>20</td>
<td>163</td>
<td>62</td>
</tr>
<tr>
<td>21</td>
<td>188</td>
<td>68</td>
</tr>
<tr>
<td>22</td>
<td>92</td>
<td>72</td>
</tr>
</tbody>
</table>

*Table 7.17: All test results response time proprietary vs. Amazon ELB load balancing*

The following table shows only the results that are used in this section. As was described in section 7.1.4, only the results that did not result in out-of-memory errors are taken into consideration. From these results, based on the graph in figure 7.5 only those values that lie in the range where the graph stabilizes (10 test results) are used for testing the hypothesis with the matching Amazon Elastic Load Balancing results, as only these are considered the relevant values.

<table>
<thead>
<tr>
<th>#</th>
<th>Proprietary (ms)</th>
<th>Amazon ELB Load balancing (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>264</td>
<td>101</td>
</tr>
<tr>
<td>2</td>
<td>230</td>
<td>97</td>
</tr>
<tr>
<td>3</td>
<td>202</td>
<td>93</td>
</tr>
<tr>
<td>4</td>
<td>179</td>
<td>77</td>
</tr>
<tr>
<td>5</td>
<td>198</td>
<td>69</td>
</tr>
<tr>
<td>6</td>
<td>150</td>
<td>61</td>
</tr>
<tr>
<td>7</td>
<td>138</td>
<td>67</td>
</tr>
</tbody>
</table>
Table 7.18: Correct test results response time proprietary vs. Amazon ELB load balancing

<table>
<thead>
<tr>
<th>#</th>
<th>Proprietary (ms)</th>
<th>Amazon ELB Load balancing (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>130</td>
<td>58</td>
</tr>
<tr>
<td>9</td>
<td>217</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td>117</td>
<td>60</td>
</tr>
</tbody>
</table>

In this section it is determined if the response time for a subscriber’s request for service improves when using load balancing instead of using a proprietary server. The hypothesis is:

- The response time for a subscriber’s request for service improves when using load balancing instead of a dedicated server.

- The null hypothesis and alternative hypothesis are defined as follows:
  
  \[ H_0: \mu_1 = \mu_3, \text{ i.e. } d = \mu_1 - \mu_3 = 0 \]  
  
  \[ H_a: d > 0 \]

  Where \( \mu_1 \) is the mean response time for a subscriber’s request for service when using a proprietary server, and \( \mu_3 \) is the mean response time for a subscriber’s request for service when using load balancing. \( \mu_3 \) has smaller value than \( \mu_1 \), i.e. improved response time for a subscriber’s request for service when using load balancing compared to using proprietary servers, thus their difference is greater than zero.

- This test has one-tailed test because the hypothesis involves the phrase “improves”, i.e. an improvement is expected. One-tailed tests expect the effect to be in a certain direction.

- The \( \alpha \) level is: \( \alpha = .05 \)

The mean, the number of tests that were carried out, the standard deviation from the mean and the standard error of the mean are shown in the following figure for this test.

![Figure 7.18: Descriptive statistics t-test proprietary vs. Amazon Elastic Load Balancing](image1.png)

The following figure gives the correlation between the pair of variables and the p value for the correlation coefficient.

![Figure 7.19: Correlation output of t-test proprietary vs. Amazon Elastic Load Balancing](image2.png)

The difference of the two means, the difference in their standard deviations from the mean, the difference of the standard errors of the means, the observed or calculated t-value, followed by the degrees of freedom associated with the t-test and the significance "Sig. (2-tailed)", which gives the two-tailed p value associated with the test, are shown in the following figure.

![Figure 7.20: Inferential statistics t-test proprietary vs. Amazon Elastic Load Balancing](image3.png)
The corresponding degrees-of-freedom \( df \) is 9 and the two-tailed \( p \)-value is .000. According to Hallstone (2013), the \( p \)-value is not really equal to zero as given by SPSS. It is just smaller than .0001, thus \( p < .0001 \). For a two-tailed test the null hypothesis could already be rejected, because \( p < .0001 < .05 \). The observed or calculated t-test value \( (t_{\text{OBS}}) \) is 9.339. Because \( t_{\text{OBS}} \) is positive, significance is found in the positive one-tailed t-tests. The one-tailed t-test in the negative direction would not be significant, because \( \alpha \) was placed in the wrong tail. The difference \( d \) was predicted to be positive before the test was conducted and the \( t_{\text{OBS}} \) value is in the same direction. For this reason this \( p \)-value \( p < .0001 \) divided in half, i.e. \( p < .0001 \) divided by 2.

\[
\frac{.0001}{2} = .00005
\]

\( p < .00005 \), which is less than \( \frac{.05}{2} = .025 \).

Hence, the null hypothesis \( H_0 \) is rejected, and the alternative hypothesis \( H_1 \) is concluded.

- **Using Apache2 Load Balancer**

In this section the same t-test is carried out using a proprietary server and load balancing, but in this case using the Apache2 load balancer, instead of the Amazon Elastic Load Balancer. The following table shows the obtained response time results for the proprietary server and load balancing scenarios, using Apache 2 load balancer, and is followed by the procedure for testing the hypothesis given below using SPSS. In this table of the 22 tests carried out, 8 failed due to out-of-memory-errors as was described in section 7.1.4. Hence, only 14 results are considered useful to test the hypothesis in this section.

<table>
<thead>
<tr>
<th>#</th>
<th>Proprietary (ms)</th>
<th>Apache2 Load balancing (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1478</td>
<td>755</td>
</tr>
<tr>
<td>2</td>
<td>591</td>
<td>232</td>
</tr>
<tr>
<td>3</td>
<td>706</td>
<td>156</td>
</tr>
<tr>
<td>4</td>
<td>461</td>
<td>121</td>
</tr>
<tr>
<td>5</td>
<td>264</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>230</td>
<td>84</td>
</tr>
<tr>
<td>7</td>
<td>202</td>
<td>68</td>
</tr>
<tr>
<td>8</td>
<td>179</td>
<td>62</td>
</tr>
<tr>
<td>9</td>
<td>198</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td>150</td>
<td>51</td>
</tr>
<tr>
<td>11</td>
<td>138</td>
<td>54</td>
</tr>
<tr>
<td>12</td>
<td>130</td>
<td>55</td>
</tr>
<tr>
<td>13</td>
<td>217</td>
<td>52</td>
</tr>
<tr>
<td>14</td>
<td>117</td>
<td>49</td>
</tr>
<tr>
<td>15</td>
<td>98</td>
<td>61</td>
</tr>
<tr>
<td>16</td>
<td>115</td>
<td>48</td>
</tr>
<tr>
<td>17</td>
<td>145</td>
<td>50</td>
</tr>
<tr>
<td>18</td>
<td>97</td>
<td>49</td>
</tr>
<tr>
<td>19</td>
<td>548</td>
<td>51</td>
</tr>
<tr>
<td>20</td>
<td>163</td>
<td>49</td>
</tr>
<tr>
<td>21</td>
<td>188</td>
<td>46</td>
</tr>
<tr>
<td>22</td>
<td>92</td>
<td>50</td>
</tr>
</tbody>
</table>

*Table 7.19: All test results response time proprietary vs. Apache2 load balancing*

The following table shows only the results that are used in this section. As was described in section 7.1.4, only the results that did not result in out-of-memory errors are taken into consideration. From these results, based on the graph in figure 7.5 only those values that lie in the range where the graph
stabilizes are used (10 results) for testing the hypothesis with the matching Apache2 load balancing results, as only these are considered the relevant values.

<table>
<thead>
<tr>
<th>#</th>
<th>Proprietary (ms)</th>
<th>Apache2 Load balancing (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>264</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>230</td>
<td>84</td>
</tr>
<tr>
<td>3</td>
<td>202</td>
<td>68</td>
</tr>
<tr>
<td>4</td>
<td>179</td>
<td>62</td>
</tr>
<tr>
<td>5</td>
<td>198</td>
<td>61</td>
</tr>
<tr>
<td>6</td>
<td>150</td>
<td>51</td>
</tr>
<tr>
<td>7</td>
<td>138</td>
<td>54</td>
</tr>
<tr>
<td>8</td>
<td>130</td>
<td>55</td>
</tr>
<tr>
<td>9</td>
<td>217</td>
<td>52</td>
</tr>
<tr>
<td>10</td>
<td>117</td>
<td>49</td>
</tr>
</tbody>
</table>

Table 7.20: Correct test results response time proprietary vs. Apache2 load balancing

In this section it is determined if the response time for a subscriber’s request for service improves when using load balancing instead of using a proprietary server. The hypothesis is:

- The response time for a subscriber’s request for service improves when using load balancing instead of a dedicated server.

  • The null hypothesis and alternative hypothesis are defined as follows:
    \( H_0: \mu_1 = \mu_3, \) i.e. \( d = \mu_1 - \mu_3 = 0 \) where \( d \) is the mean value of the difference.
    \( H_1: d > 0 \)
    Where \( \mu_1 \) is the mean response time for a subscriber’s request for service when using a proprietary server, and \( \mu_3 \) is the mean response time for a subscriber’s request for service when using load balancing. \( \mu_3 \) has smaller value than \( \mu_1 \), i.e. improved response time for a subscriber’s request for service when using load balancing compared to using proprietary servers, thus their difference is greater than zero.
  
  • This test has one-tailed test because the hypothesis involves the phrase "improves", i.e. an improvement is expected. One-tailed tests expect the effect to be in a certain direction.

  • The \( \alpha \) level is: \( \alpha = .05 \)

The mean, the number of tests that were carried out, the standard deviation from the mean and the standard error of the mean are shown in the following figure for this test.

![Figure 7.21: Descriptive statistics t-test proprietary vs. Apache2 load balancer](image)

The following figure gives the correlation between the pair of variables and the \( p \) value for the correlation coefficient.

![Figure 7.22: Correlation output of t-test proprietary vs. Apache2 load balancer](image)
The difference of the two means, the difference in their standard deviations from the mean, the difference of the standard errors of the means, the observed or calculated t-value, followed by the degrees of freedom associated with the t-test and the significance “Sig. (2-tailed)”, which gives the two-tailed p value associated with the test, are shown in the following figure.

![Figure 7.23: Inferential statistics t-test proprietary vs. Apache2 load balancer](image)

The corresponding degrees-of-freedom df is 9 and the two-tailed p-value is .000. The p-value is not really equal to zero as given by SPSS. It is just smaller than .0001, thus $p < .0001$. For a two-tailed test the null hypothesis could already be rejected, because $p < .0001 < .05$. The observed or calculated t-test value ($t_{obs}$) is 9.782. Because $t_{obs}$ is positive, significance is found in the positive one-tailed t-tests. The one-tailed t-test in the negative direction would not be significant, because $\alpha$ was placed in the wrong tail. The difference $d$ was predicted to be positive before the test was conducted and the $t_{obs}$ value is in the same direction.

For this reason this $p$-value $p < .0001$ is divided in half, i.e. $p < .0001$ divided by 2.

$$\frac{.0001}{2} = .00005$$

$p < .00005$, which is less than $\frac{.05}{2} = .025$.

Hence, the null hypothesis $H_0$ is rejected, and the alternative hypothesis $H_1$ is concluded.

**Scenario 2 vs. scenario 3**

In this case also two different configurations are used to carry out the tests comparing with the auto-scaling scenario, one using the Amazon Elastic Load Balancer and the other using Apache2 load balancer. This section elaborates the testing of the hypothesis based on each configuration.

- **Using Amazon Elastic Load Balancer**

The following table shows the obtained response time results for the auto-scaling and load balancing scenarios, using in this case the Amazon Elastic load Balancer, and is followed by the procedure for testing the hypothesis given below using SPSS.

<table>
<thead>
<tr>
<th>#</th>
<th>Auto-scaling (ms)</th>
<th>Amazon ELB Load balancing (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1144</td>
<td>465</td>
</tr>
<tr>
<td>2</td>
<td>621</td>
<td>248</td>
</tr>
<tr>
<td>3</td>
<td>427</td>
<td>159</td>
</tr>
<tr>
<td>4</td>
<td>342</td>
<td>144</td>
</tr>
<tr>
<td>5</td>
<td>245</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>242</td>
<td>97</td>
</tr>
<tr>
<td>7</td>
<td>235</td>
<td>93</td>
</tr>
<tr>
<td>8</td>
<td>192</td>
<td>77</td>
</tr>
<tr>
<td>9</td>
<td>159</td>
<td>69</td>
</tr>
<tr>
<td>10</td>
<td>171</td>
<td>61</td>
</tr>
<tr>
<td>11</td>
<td>141</td>
<td>67</td>
</tr>
<tr>
<td>12</td>
<td>133</td>
<td>58</td>
</tr>
<tr>
<td>13</td>
<td>156</td>
<td>61</td>
</tr>
<tr>
<td>14</td>
<td>126</td>
<td>60</td>
</tr>
</tbody>
</table>
In the following table only the results that are used in this section are shown. As described in section 7.1.5, based on the graph in figure 7.10 these include only the results that lie in the range where the graph stabilizes. These results are used (18 results) for testing the hypothesis with the matching Elastic Load Balancing results, as only these are considered the relevant values.

### Table 7.21: Test results response time auto-scaling vs. Amazon Elastic Load Balancing

<table>
<thead>
<tr>
<th>#</th>
<th>Auto-scaling (ms)</th>
<th>Amazon ELB Load balancing (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>153</td>
<td>60</td>
</tr>
<tr>
<td>16</td>
<td>121</td>
<td>68</td>
</tr>
<tr>
<td>17</td>
<td>109</td>
<td>53</td>
</tr>
<tr>
<td>18</td>
<td>117</td>
<td>68</td>
</tr>
<tr>
<td>19</td>
<td>166</td>
<td>80</td>
</tr>
<tr>
<td>20</td>
<td>103</td>
<td>62</td>
</tr>
<tr>
<td>21</td>
<td>99</td>
<td>68</td>
</tr>
<tr>
<td>22</td>
<td>118</td>
<td>72</td>
</tr>
</tbody>
</table>

In this section it is determined if the response time for a subscriber’s request for service improves when using load balancing instead of using auto-scaling technology. The hypothesis is:

- The response time for a subscriber’s request for service improves when using load balancing instead of auto-scaling technology.

The null hypothesis and alternative hypothesis are defined as follows:

- $H_0$: $\mu_2 = \mu_3$, i.e. $d = \mu_2 - \mu_3 = 0$ where $d$ is the mean value of the difference.
- $H_1$: $d > 0$

Where $\mu_2$ is the mean response time for a subscriber’s request for service when using auto-scaling technology, and $\mu_3$ is the mean response time for a subscriber’s request for service when using load balancing. $\mu_3$ has smaller value than $\mu_2$, i.e. improved response time for a subscriber’s request for service when using load balancing compared to using auto-scaling technology, thus their difference is greater than zero.
• This test has one-tailed test because the hypothesis involves the phrase "improves", i.e. an improvement is expected. One-tailed tests expect the effect to be in a certain direction.
• The α level is: α = .05

In SPSS when running the test the output viewer will appear with the results of the t-test, which consists of three main parts: descriptive statistics, the correlation between the pair of variables, and inferential statistics. To determine if the null hypothesis can be rejected the third part is used. The other two parts provide additional information if required.

Figure 7.24 gives the mean, the number of tests that were carried out, the standard deviation from the mean and the standard error of the mean for the auto-scaling vs. load balancing group.

<table>
<thead>
<tr>
<th>Paired Samples Statistics</th>
<th>Mean</th>
<th>N</th>
<th>Std. Deviation</th>
<th>Std. Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoScaling</td>
<td>154.7278</td>
<td>18</td>
<td>46.82955</td>
<td>11.03783</td>
</tr>
<tr>
<td>AmazonElasticLoadbalanc er</td>
<td>70.8333</td>
<td>18</td>
<td>11.80643</td>
<td>3.25421</td>
</tr>
</tbody>
</table>

**Figure 7.24: Descriptive statistics t-test auto-scaling vs. Amazon Elastic Load Balancing**

Figure 7.25 gives the correlation between the pair of variables and the p-value for the correlation coefficient.

<table>
<thead>
<tr>
<th>Paired Samples Correlations</th>
<th>N</th>
<th>Correlation</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoScaling &amp; AmazonElasticLoadbalancer</td>
<td>18</td>
<td>.816</td>
<td>.000</td>
</tr>
</tbody>
</table>

**Figure 7.25: Correlation output of t-test auto-scaling vs. Amazon Elastic Load Balancing**

Figure 7.26 gives the inferential statistics, including the difference of the two means, the difference in their standard deviations from the mean and the difference of the standard errors of the means. Also the observed or calculated t value is given, followed by the degrees of freedom associated with the t-test and the significance "Sig. (2-tailed)", which gives the two-tailed p value associated with the test.

<table>
<thead>
<tr>
<th>Paired Samples Test</th>
<th>Paired Differences</th>
<th>95% Confidence Interval of the Difference</th>
<th>t</th>
<th>df</th>
<th>Sig. (2-tailed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoScaling - AmazonElasticLoadbalancer</td>
<td>83.94444</td>
<td>36.08890</td>
<td>8.50623</td>
<td>65.99786</td>
<td>101.89103</td>
</tr>
</tbody>
</table>

**Figure 7.26: Inferential statistics t-test auto-scaling vs. Amazon Elastic Load Balancing**

The corresponding degrees-of-freedom df is 17 and the two-tailed p-value is .000. The p-value is not really equal to zero as given by SPSS. It is just smaller than .0001, thus p < .0001. For a two-tailed test the null hypothesis could already be rejected, because p < .0001 < .05. The observed or calculated t-test value (tobs) is 9.869. Because tob is positive, significance is found in the positive one-tailed t-tests. The one-tailed t-test in the negative direction would not be significant, because α was placed in the wrong tail. The difference d was predicted to be positive before the test was conducted and the tob value is in the same direction. For this reason this p-value p < .0001 is divided in half, i.e. p < .00005 divided by 2.

\[
\frac{.0001}{2} = .00005,\ p < .00005,\text{ which is less than } .025 = \frac{.05}{2}.
\]

Hence, the null hypothesis H0 is rejected, and the alternative hypothesis H1 is concluded.
- Using Apache2 web server as load balancer

In the following table the obtained response time results are found for the auto-scaling and load balancing scenarios, using the Apache 2 web server as the load balancer, and is followed by the procedure for testing the hypothesis given below using SPSS.

<table>
<thead>
<tr>
<th>#</th>
<th>Auto-scaling (ms)</th>
<th>Apache2 load balancing (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1144</td>
<td>755</td>
</tr>
<tr>
<td>2</td>
<td>621</td>
<td>232</td>
</tr>
<tr>
<td>3</td>
<td>427</td>
<td>156</td>
</tr>
<tr>
<td>4</td>
<td>342</td>
<td>121</td>
</tr>
<tr>
<td>5</td>
<td>245</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>242</td>
<td>84</td>
</tr>
<tr>
<td>7</td>
<td>235</td>
<td>68</td>
</tr>
<tr>
<td>8</td>
<td>192</td>
<td>62</td>
</tr>
<tr>
<td>9</td>
<td>159</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td>171</td>
<td>51</td>
</tr>
<tr>
<td>11</td>
<td>141</td>
<td>54</td>
</tr>
<tr>
<td>12</td>
<td>133</td>
<td>55</td>
</tr>
<tr>
<td>13</td>
<td>156</td>
<td>52</td>
</tr>
<tr>
<td>14</td>
<td>126</td>
<td>49</td>
</tr>
<tr>
<td>15</td>
<td>153</td>
<td>61</td>
</tr>
<tr>
<td>16</td>
<td>121</td>
<td>48</td>
</tr>
<tr>
<td>17</td>
<td>109</td>
<td>50</td>
</tr>
<tr>
<td>18</td>
<td>117</td>
<td>49</td>
</tr>
<tr>
<td>19</td>
<td>166</td>
<td>51</td>
</tr>
<tr>
<td>20</td>
<td>103</td>
<td>49</td>
</tr>
<tr>
<td>21</td>
<td>99</td>
<td>46</td>
</tr>
<tr>
<td>22</td>
<td>118</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 7.23: Test results response time auto-scaling vs. Apache2 load balancing

The following table shows only the results that are used in this section. As was described in section 7.1.5, only the results based on the graph in figure 7.10 that lie in the range where the graph stabilizes are used (18 results) for testing the hypothesis with the matching Apache2 load balancer results, as only these are considered the relevant values.

<table>
<thead>
<tr>
<th>#</th>
<th>Auto-scaling (ms)</th>
<th>Apache2 load balancing (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>245</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>242</td>
<td>84</td>
</tr>
<tr>
<td>3</td>
<td>235</td>
<td>68</td>
</tr>
<tr>
<td>4</td>
<td>192</td>
<td>62</td>
</tr>
<tr>
<td>5</td>
<td>159</td>
<td>61</td>
</tr>
<tr>
<td>6</td>
<td>171</td>
<td>51</td>
</tr>
<tr>
<td>7</td>
<td>141</td>
<td>54</td>
</tr>
<tr>
<td>8</td>
<td>133</td>
<td>55</td>
</tr>
<tr>
<td>9</td>
<td>156</td>
<td>52</td>
</tr>
<tr>
<td>10</td>
<td>126</td>
<td>49</td>
</tr>
<tr>
<td>11</td>
<td>153</td>
<td>61</td>
</tr>
<tr>
<td>12</td>
<td>121</td>
<td>48</td>
</tr>
<tr>
<td>13</td>
<td>109</td>
<td>50</td>
</tr>
<tr>
<td>14</td>
<td>117</td>
<td>49</td>
</tr>
<tr>
<td>15</td>
<td>166</td>
<td>51</td>
</tr>
<tr>
<td>16</td>
<td>103</td>
<td>49</td>
</tr>
<tr>
<td>17</td>
<td>99</td>
<td>46</td>
</tr>
<tr>
<td>18</td>
<td>118</td>
<td>50</td>
</tr>
</tbody>
</table>

Table 7.24: Used test results response time auto-scaling vs. Apache2 load balancing
In this section it is determined if the response time for a subscriber’s request for service improves when using load balancing instead of using auto-scaling technology.

- The response time for a subscriber’s request for service improves when using load balancing instead of auto-scaling technology.

- The null hypothesis and alternative hypothesis are defined as follows:
  \( H_0: \mu_2 = \mu_3, \) i.e. \( d = \mu_2 - \mu_3 = 0 \) where \( d \) is the mean value of the difference.
  \( H_1: \ d > 0 \)
  Where \( \mu_2 \) is the mean response time for a subscriber’s request for service when using auto-scaling technology, and \( \mu_3 \) is the mean response time for a subscriber’s request for service when using load balancing. \( \mu_3 \) has smaller value than \( \mu_2 \), i.e. improved response time for a subscriber’s request for service when using load balancing compared to using auto-scaling technology, thus their difference is greater than zero.

- This test has one-tailed test because the hypothesis involves the phrase "improves", i.e. an improvement is expected. One-tailed tests expect the effect to be in a certain direction.

- The \( \alpha \) level is: \( \alpha = .05 \)

In SPSS when running the test the output viewer will appear with the results of the t-test, which consists of three main parts: descriptive statistics, the correlation between the pair of variables, and inferential statistics. To determine if the null hypothesis can be rejected the third part is used. The other two parts provide additional information if required.

Figure 7.27 gives the mean, the number of tests that were carried out, the standard deviation from the mean and the standard error of the mean for the auto-scaling vs. load balancing group.

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>N</th>
<th>Std Deviation</th>
<th>Std. Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair 1 AutoScaling</td>
<td>154.7778</td>
<td>18</td>
<td>46.82955</td>
<td>11.01785</td>
</tr>
<tr>
<td>Apache2LoadBalancer</td>
<td>56.8667</td>
<td>18</td>
<td>10.94371</td>
<td>2.57946</td>
</tr>
</tbody>
</table>

*Figure 7.27: Descriptive statistics t-test auto-scaling vs. Apache2 load balancer*

Figure 7.28 gives the correlation between the pair of variables and the p-value for the correlation coefficient.

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Correlation</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair 1 AutoScaling &amp;</td>
<td>18</td>
<td>.903</td>
<td>.000</td>
</tr>
<tr>
<td>Apache2LoadBalancer</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 7.28: Correlation output for t-test auto-scaling vs. Apache2 load balancer*

Figure 7.29 gives the inferential statistics, including the difference of the two means, the difference in their standard deviations from the mean and the difference of the standard errors of the means. Also the observed or calculated t value is given, followed by the degrees of freedom associated with the t-test and the significance "Sig. (2-tailed)", which gives the two-tailed \( p \) value associated with the test.

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Paired Differences</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>Std Deviation</td>
<td>Std Error Mean</td>
<td>95% Confidence Interval of the Difference</td>
<td>t</td>
<td>df</td>
<td>Sig. (2- tailed)</td>
</tr>
<tr>
<td>Pair 1 AutoScaling -</td>
<td>98.1111</td>
<td>37.24938</td>
<td>6.77976</td>
<td>79.58743 - 116.63479</td>
<td>11.175</td>
<td>17</td>
<td>.000</td>
</tr>
<tr>
<td>Apache2LoadBalancer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 7.29: Inferential statistics t-test auto-scaling vs. Apache2 load balancer*
The corresponding degrees-of-freedom \( df \) is 17 and the two-tailed \( p \)-value is .000. The \( p \)-value is not really equal to zero as given by SPSS. It is just smaller than .0001, thus \( p < .0001 \). For a two-tailed test the null hypothesis could already be rejected, because \( p < .0001 < .05 \). The observed or calculated \( t \)-test value (tobs) is 11.75. Because tobs is positive, significance is found in the positive one-tailed \( t \)-tests. The one-tailed \( t \)-test in the negative direction would not be significant, because \( \alpha \) was placed in the wrong tail. The difference was predicted to be positive before the test was conducted and the tobs value is in the same direction. For this reason this \( p \)-value \( p < .0001 \) is divided in half, i.e. \( p < .0001 \) divided by 2.

\[
\frac{.0001}{2} = .00005, \ p < .00005, \text{ which is less than } \frac{.05}{2} = .025. \text{ Hence, the null hypothesis } H_0 \text{ is rejected, and the alternative hypothesis } H_1 \text{ is concluded.}
\]

**Comparing Amazon Elastic Load Balancer with Apache2 load balancer**

For this comparison the following hypothesis is defined:

- The response time for a subscriber’s request for service improves when using Apache2 load balancer instead of the Amazon Elastic Load Balancer.

The following table shows the obtained response time results when using the Amazon Elastic Load Balancer and the Apache2 load balancer and is followed by the procedure for testing the above stated hypothesis using SPSS.

<table>
<thead>
<tr>
<th>#</th>
<th>Amazon Elastic Load Balancer (ms)</th>
<th>Apache2 load balancer (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>465</td>
<td>755</td>
</tr>
<tr>
<td>2</td>
<td>248</td>
<td>232</td>
</tr>
<tr>
<td>3</td>
<td>159</td>
<td>156</td>
</tr>
<tr>
<td>4</td>
<td>144</td>
<td>121</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>97</td>
<td>84</td>
</tr>
<tr>
<td>7</td>
<td>93</td>
<td>68</td>
</tr>
<tr>
<td>8</td>
<td>77</td>
<td>62</td>
</tr>
<tr>
<td>9</td>
<td>69</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td>61</td>
<td>51</td>
</tr>
<tr>
<td>11</td>
<td>67</td>
<td>54</td>
</tr>
<tr>
<td>12</td>
<td>58</td>
<td>55</td>
</tr>
<tr>
<td>13</td>
<td>61</td>
<td>52</td>
</tr>
<tr>
<td>14</td>
<td>60</td>
<td>49</td>
</tr>
<tr>
<td>15</td>
<td>60</td>
<td>61</td>
</tr>
<tr>
<td>16</td>
<td>68</td>
<td>48</td>
</tr>
<tr>
<td>17</td>
<td>53</td>
<td>50</td>
</tr>
<tr>
<td>18</td>
<td>68</td>
<td>49</td>
</tr>
<tr>
<td>19</td>
<td>80</td>
<td>51</td>
</tr>
<tr>
<td>20</td>
<td>62</td>
<td>49</td>
</tr>
<tr>
<td>21</td>
<td>68</td>
<td>46</td>
</tr>
<tr>
<td>22</td>
<td>72</td>
<td>50</td>
</tr>
</tbody>
</table>

*Table 7.25: Test results response time Amazon Elastic Load Blancer vs. Apache2 load balancer*

The following table shows only the results that are used in this section. As was described in sections 7.1.6.1 and 7.1.6.2, only the results based on the graphs in figure 7.12 and figure 7.14 that lie in the range where these graph stabilize are used (18 results) for testing the hypothesis stated above, as only these are considered the relevant values.
In this section it is determined if the response time for a subscriber’s request for service improves when using Apache2 web server as the load balancer instead of using the Amazon Elastic Load Balancer.

- The response time for a subscriber’s request for service improves when using Apache2 load balancer instead of the Amazon Elastic Load Balancer.

• The null hypothesis and alternative hypothesis are defined as follows:
  
  \[ H_0: \mu_{3A} = \mu_{3B}, \text{ i.e. } d = \mu_{3A} - \mu_{3B} = 0 \]
  
  \[ H_1: d > 0 \]

  Where \( \mu_{3A} \) is the mean response time for a subscriber’s request for service when using the Amazon Elastic Load Balancer, and \( \mu_{3B} \) is the mean response time for a subscriber’s request for service when using Apache2 load balancer. \( \mu_{3B} \) has smaller value than \( \mu_{3A} \), i.e. improved response time for a subscriber’s request for service when using Apache2 load balancer compared to using Amazon Elastic Load Balancer, thus their difference is greater than zero.

• This test has one-tailed test because the hypothesis involves the phrase "improves", i.e. an improvement is expected. One-tailed tests expect the effect to be in a certain direction.

• The \( \alpha \) level is: \( \alpha = .05 \)

In SPSS when running the test the output viewer will appear with the results of the t-test, which consists of three main parts: descriptive statistics, the correlation between the pair of variables, and inferential statistics. To determine if the null hypothesis can be rejected the third part is used. The other two parts provide additional information if required.

The following figure gives the mean, the number of tests that were carried out, the standard deviation from the mean and the standard error of the mean for the Amazon Elastic Load Blancer vs. Apache2 load balancer group.
**Paired Samples Statistics**

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>N</th>
<th>Std. Deviation</th>
<th>Std. Error Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair 1</td>
<td>70.833</td>
<td>18</td>
<td>13.8064</td>
<td>3.25421</td>
</tr>
<tr>
<td>Apache2 LoadBalancer</td>
<td>56.667</td>
<td>18</td>
<td>10.94371</td>
<td>2.57946</td>
</tr>
</tbody>
</table>

Figure 7.30: Descriptive statistics t-test Amazon Elastic Load Balancer vs. Apache2 load balancer

Figure 7.31 gives the correlation between the pair of variables and the p-value for the correlation coefficient.

**Paired Samples Correlations**

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Correlation</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair 1</td>
<td>18</td>
<td>.805</td>
<td>.000</td>
</tr>
</tbody>
</table>

Figure 7.31: Correlation output for t-test Amazon Elastic Load Balancer vs. Apache2 load balancer

Figure 7.32 gives the inferential statistics, including the difference of the two means, the difference in their standard deviations from the mean and the difference of the standard errors of the means. Also the observed or calculated t value is given, followed by the degrees of freedom associated with the t-test and the significance "Sig. (2-tailed)", which gives the two-tailed $p$ value associated with the test.

**Paired Samples Test**

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>Std. Deviation</th>
<th>Std Error Mean</th>
<th>95% Confidence Interval of the Difference</th>
<th>Lower</th>
<th>Upper</th>
<th>t</th>
<th>df</th>
<th>Sig. (2-tailed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair 1</td>
<td>14.1667</td>
<td>8.19674</td>
<td>1.93058</td>
<td>10.09351</td>
<td>18.23983</td>
<td></td>
<td>7.338</td>
<td>17</td>
<td>.000</td>
</tr>
</tbody>
</table>

Figure 7.32: Inferential statistics t-test Amazon Elastic Load Balancer vs. Apache2 load balancer

The corresponding degrees-of-freedom $df$ is 17 and the two-tailed $p$-value is .000. The $p$-value is not really equal to zero as given by SPSS. It is just smaller than .0001, thus $p < .0001$. For a two-tailed test the null hypothesis could already be rejected, because $p < .0001 < .05$. The observed or calculated t-test value ($t_{obs}$) is 7.338. Because $t_{obs}$ is positive, significance is found in the positive one-tailed t-tests. The one-tailed t-test in the negative direction would not be significant, because $\alpha$ was placed in the wrong tail. The difference $d$ was predicted to be positive before the test was conducted and the $t_{obs}$ value is in the same direction. For this reason this $p$-value $p < .0001$ is divided in half, i.e. $p < .00005$ divided by 2.

$$p \div 2 = .00005, p < .00005,$$ which is less than $\frac{.05}{2} = .025$. Hence, the null hypothesis $H_0$ is rejected, and the alternative hypothesis $H_1$ is concluded.

### 7.2.3 Evaluation of comparison of results

The method used to determine the availability of the messaging service using any of three scenarios was straightforward, as only in the first scenario at some point no more memory was available for the server instance to handle more requests. In remaining scenarios more resources are made available, improving this way the availability of the messaging service compared to using a dedicated server. Concerning the comparison of the performance of the three scenarios, using the t-test statistical method the stated hypotheses could be statistically tested, this way determining if the found test results were gotten by chance or not.
Based on the above test results a proprietary server can process a maximum number of requests, until a second instance is made ready, e.g. using Corosync Pacemaker, to further process more requests. In the mean time when the first instance has failed, no requests are processed till the second instance is up. Based on the configuration used in auto scaling, and the created CloudWatch alarms, in the Auto Scaling scenario the response time may be reduced by using a different configuration. The booting of the second instance when scaling out took time, leaving the first instance as the only instance handling requests for a long time. Booting the second server more quickly may reduce the response time as more instances are used at the same time by the load balancer. The advantage of this scenario is that an unlimited number of requests can be handled as more and more instances are added to the auto-scaling group. Concerning the load balancing scenario, each added instance must be created separately and registerd with the load balancer, while with auto-scaling this happens automatically. Using a proprietary server also an additional instance must be bought and configured with Corosync Pacemaker to enable the automatically booting of servers when one server goes down.

7.3 Comparison of scenarios based on costs involved

In this section an indication is given of the costs involved in implementing each scenario described in this thesis. These costs are calculated on a yearly basis and serve as an instrument to consider when implementing an infrastructure for highly available services. In the previous sections in this chapter an indication was given of the response time of an HTTP-based messaging solution deployed in all three listed scenarios (see section 1.2). The response time gives an impression of which scenario may be more adequate when there is high need for highly available services. Still a cost analysis is required when considering which scenario to use in practice, as the costs implied in each scenario may vary drastically. This section provides this financial aspect for all three scenarios to give an idea of the costs involved, using the described m3.xlarge instance type. The three scenarios are evaluated in the following sections based on their cost per year and cost in years, followed by an evaluation of these costs and an evaluation of performance and costs combined.

7.3.1 Amazon Cloud costs comparison

The three scenarios are evaluated in the following sections based on their cost per year and cost in years.

- Proprietary server solution

The infrastructure that is generally used by a customer owning proprietary servers, consists of two dedicated servers, one acting as the main production server, while the other is running as a back-up server in case the first server goes down. In case the main production server goes down, using Corosync Pacemaker the second server is made available by starting up all required Tomcat instances running the needed applications. The cost indication for this scenario is based on three pricing models for three different ways of acquiring single servers from Amazon. The first cost indication is based on the use of a two Amazon On-Demand EC2 instances paying for compute capacity by the hour with no long-term commitments.
On demand EC2 instance

<table>
<thead>
<tr>
<th>Product</th>
<th>Hours a Day</th>
<th>Days a Month</th>
<th>Hours a Month</th>
<th>Quantity</th>
<th>Price</th>
<th>Price a Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC2</td>
<td>24</td>
<td>30</td>
<td>720</td>
<td>2</td>
<td>$0.450 per hour</td>
<td>$648.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total a month: $648.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total a year: $7776.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total in 3 years: $23328.00</td>
</tr>
</tbody>
</table>

Table 7.27: Cost indication On-Demand EC2 instance

The second cost indication is based on the use of Amazon EC2 Dedicated On-Demand instances paying for compute capacity by the hour with no long-term commitments. Dedicated Instance pricing has two components:
- an hourly per instance usage fee and
- a dedicated per region fee

**Dedicated On-Demand instance per region fee**

<table>
<thead>
<tr>
<th></th>
<th>Hours a Day</th>
<th>Days a Month</th>
<th>Hours a Month</th>
<th>Quantity</th>
<th>Price</th>
<th>Total for three years</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC2</td>
<td>24</td>
<td>30</td>
<td>720</td>
<td>2</td>
<td>$0.495 per hour</td>
<td>$712.80</td>
</tr>
<tr>
<td>Dedicated per region fee</td>
<td>24</td>
<td>30</td>
<td>720</td>
<td>1</td>
<td>$2.00 per hour</td>
<td>$1,440.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total a month: $2,152.80</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total a year: $25,833.60</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total in 3 years: $77500.80</td>
</tr>
</tbody>
</table>

Table 7.28: Cost indication Dedicated On-Demand instance

The third cost indication is based on the use of an Amazon Dedicated Heavy Utilization Reserved Instance, which runs for a complete year or three years.

**Dedicated Reserved Instance Heavy 1-Year Term:**

<table>
<thead>
<tr>
<th></th>
<th>Hours a Day</th>
<th>Days a Month</th>
<th>Hours a Month</th>
<th>Number of Months</th>
<th>Quantity</th>
<th>Price</th>
<th>Total for one Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upfront</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td>$1,266</td>
<td>$ 2,532.00</td>
</tr>
<tr>
<td>EC2</td>
<td>24</td>
<td>30</td>
<td>720</td>
<td>12</td>
<td>2</td>
<td>$0.105 per hour</td>
<td>$ 1,814.40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total a year: $ 4346.40</td>
</tr>
</tbody>
</table>

Table 7.29: Cost indication Dedicated Reserved instance 1 year

**Dedicated Reserved Instance Heavy 3-Year Term:**

<table>
<thead>
<tr>
<th></th>
<th>Hours a Day</th>
<th>Days a Month</th>
<th>Hours a Month</th>
<th>Number of Months</th>
<th>Quantity</th>
<th>Price</th>
<th>Total for three years</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upfront</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td>$1,922</td>
<td>$3,844.00</td>
</tr>
<tr>
<td>EC2</td>
<td>24</td>
<td>30</td>
<td>720</td>
<td>36</td>
<td>2</td>
<td>$0.086 per hour</td>
<td>$4,458.24</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total in 3 years: $8,302.24</td>
</tr>
</tbody>
</table>

Table 7.30: Cost indication Dedicated Reserved Instance 3 years
- **Amazon AWS Auto Scaling solution**

The pricing estimation for the auto-scaling scenario is hourly-based and calculated using a single Elastic Load Balancer. The price is calculated for each hour the Elastic Load Balancer is running and for each GB of data transferred through the Elastic Load Balancer. For this estimation it is assumed the Elastic Load Balancer transferred 20 GB of data over a 30 day-period. In addition the price for a single EC2 instance is calculated for a single month of usage. It is assumed that auto-scaling occurs three times a day for a period of 2 hours at a time on average, which implies that an extra EC2 instance is used for 6 hours a day, totaling 6 x 30 = 180 hours a month. Thus one m3.xlarge instance is used the whole month totaling 24 x 30 = 720 hours and one m3.xlarge instance 180 hours. The total price per month using auto-scaling in this example scenario is shown in the following table:

### Auto Scaling: 1 ELB + 1 EC2 + auto-scaled EC2

<table>
<thead>
<tr>
<th>Product</th>
<th>Hours a Day</th>
<th>Days a Month</th>
<th>Hours a Month</th>
<th>Quantity</th>
<th>Price</th>
<th>Price a Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELB</td>
<td>24</td>
<td>30</td>
<td>720</td>
<td>1</td>
<td>$0.025 per hour</td>
<td>$18.00</td>
</tr>
<tr>
<td>Data transfer</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td>$0.008 per GB</td>
<td>$0.16</td>
</tr>
<tr>
<td>EC2</td>
<td>24</td>
<td>30</td>
<td>720</td>
<td>1</td>
<td>$0.450 per hour</td>
<td>$324.00</td>
</tr>
<tr>
<td>EC2</td>
<td>24</td>
<td>7.5</td>
<td>180</td>
<td>1</td>
<td>$0.450 per hour</td>
<td>$81.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th><strong>Total a month</strong></th>
<th><strong>Price a Month</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$423.16</td>
<td>$423.16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th><strong>Total a year</strong></th>
<th><strong>Price a Year</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$5077.92</td>
<td>$5077.92</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th><strong>Total in 3 years</strong></th>
<th><strong>Price in 3 years</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$15233.76</td>
<td>$15233.76</td>
</tr>
</tbody>
</table>

Table 7.31: Cost indication Auto Scaling

- **Amazon Elastic Load Balancing**

The pricing estimation for the load-balancing scenario is hourly-based and calculated using a single Elastic Load Balancer. The price is calculated for each hour the Elastic Load Balancer is running and for each GB of data transferred through the Elastic Load Balancer. For this estimation it is assumed the Elastic Load Balancer transferred 20 GB of data over a 30 day-period. In addition the price for two EC2 instances is calculated for a single month of usage.

### 1 ELB + 2 EC2 instances

<table>
<thead>
<tr>
<th>Product</th>
<th>Hours a Day</th>
<th>Days a Month</th>
<th>Hours a Month</th>
<th>Quantity</th>
<th>Price</th>
<th>Price a Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELB</td>
<td>24</td>
<td>30</td>
<td>720</td>
<td>1</td>
<td>$0.025 per hour</td>
<td>$18.00</td>
</tr>
<tr>
<td>Data transfer</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td>$0.008 per GB</td>
<td>$0.16</td>
</tr>
<tr>
<td>EC2</td>
<td>24</td>
<td>30</td>
<td>720</td>
<td>2</td>
<td>$0.450 per hour</td>
<td>$648.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th><strong>Total a month</strong></th>
<th><strong>Price a Month</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$666.16</td>
<td>$666.16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th><strong>Total a year</strong></th>
<th><strong>Price a Year</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$7993.92</td>
<td>$7993.92</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th><strong>Total in 3 years</strong></th>
<th><strong>Price in 3 years</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$23981.76</td>
<td>$23981.76</td>
</tr>
</tbody>
</table>

Table 7.32: Cost indication Elastic Load Balancing

Also using a separate instance for the load balancer is possible as used in this thesis project, totaling three EC2 instances. The costs involved with this type are:
3 EC2 instances (including an EC2 for a load balancer)

<table>
<thead>
<tr>
<th>Product</th>
<th>Hours a Day</th>
<th>Days a Month</th>
<th>Hours a Month</th>
<th>Quantity</th>
<th>Price</th>
<th>Price a month</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC2</td>
<td>24</td>
<td>30</td>
<td>720</td>
<td>3</td>
<td>$0.450 per hour</td>
<td>$972.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total a month</td>
<td>$972.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total a year</td>
<td>$11664.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total in 3 years</td>
<td>$34992.00</td>
</tr>
</tbody>
</table>

Table 7.33: Cost indication custom load balancer

7.3.2 Evaluation of costs involved

The cost indications provided are based on the same infrastructure that was used to carry out the performance tests. Based on the results acquired from the performance tests, including the response time and maximum number of requests that each scenario can process, combining them with the costs involved, users or future users of the Cloud can get a good impression of what the given infrastructure configurations can provide concerning performance and costs involved to make these possible and implement them in the real world. On a longer-term basis, i.e. 3 years, the prices vary drastically. A comparison of the costs involved in 3 years is shown in the following table, sorted by ascending price:

<table>
<thead>
<tr>
<th>Product</th>
<th>Cost</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved Instance Heavy 3-Year Term</td>
<td>$8,302.24</td>
<td>Proprietary</td>
</tr>
<tr>
<td>Auto Scaling</td>
<td>$15,233.76</td>
<td>Auto Scaling</td>
</tr>
<tr>
<td>On demand EC2 instance</td>
<td>$23,328.00</td>
<td>Proprietary</td>
</tr>
<tr>
<td>Load balancing with ELB and 2 EC2 instances</td>
<td>$23,981.76</td>
<td>Load balancing</td>
</tr>
<tr>
<td>Load balancing with 3 EC2 instances</td>
<td>$34,992.00</td>
<td>Load balancing</td>
</tr>
<tr>
<td>Dedicated On-Demand instance per region fee</td>
<td>$77,500.80</td>
<td>Proprietary</td>
</tr>
</tbody>
</table>

Table 7.34: Amazon cost comparison

7.3.3 Evaluation of performance and costs involved

Based on the performance results and costs given in the previous tables, it can be stated that advantages in costs may have disadvantages in performance of the system, i.e. higher response times. The more one is willing to pay for the environment, the better the response could and should be, as more resources are available. The purpose is to find the best combination of applications performing well with the adequate investment for each business. For businesses requiring services to run 24/7, but not requiring highly available services, or a business that does not experience high workload at peak hours, the Reserved Instance Heavy 3-Year Term is a great option, as the services are always available for a relatively good price. Also this instance type fits well for businesses that require a separate dedicated server, where the hardware is only used by this business, and no other. For businesses requiring high availability of service, and that experience dynamically changing high workloads at different peak hours weekly of monthly, the Auto Scaling would fit better. However, as auto scaling feature may reduce costs by dynamically adding and removing server instances when required, this reducing the costs of investing in proper infrastructure, in some cases auto scaling actually may increase costs because it is not always possible to determine whether a usage spike is legitimate or it is the result of a distributed denial of service (DDoS) attack, when servers scale to address increased traffic. This creates a cost without the benefit of real traffic, and because the
cloud’s resources are so great that the service never actually goes down, a continuous rise in costs results. The solution is a smarter monitoring tool that can distinguish between real traffic and an attack (Janssen, 2014).

Businesses that do not experience high workloads at peak hours, but require their service to be highly available, i.e. low response times, and do not want to invest in their own infrastructure, the load balancing solution is the best fit, as it performs really well and EC2 instances can be added and removed (manually) as required. The collected response time results provide insight to users and future cloud users about the capabilities of the performance- and stress-tested configurations, i.e. the three scenarios using the m3.xlarge instance type.

7.4 Issues and recommendations for improvement

In the following sections the problems encountered during the implementation of the messaging service, or the execution of the tests are listed. Also some recommendations for improvement are given.

7.4.1 Issues

The issues listed here include technical implementation issues that required research to find a proper solution.

- An issue encountered during the performance tests execution was the occurrence of Tomcat connection timeouts. The Tomcat instance configuration, as shown here, must be adapted for each application in the corresponding server.xml file.

```xml
<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="120000"
maxConnections="-1" maxThreads="15000" maxHttpHeaderSize="16384"
redirectPort="8443" />
```

- During the execution of performance test at a certain point requests were not anymore received and processed by the instances behind the load balancer. The problem is described in section 7.1.3.4 and concerns the open files limits on the Linux system. This limits the number of requests that the server can receive and process, limiting this way the performance tests that can be carried out on selected instance types.

7.4.2 Recommendations

A recommendation includes doing research about combining Amazon’s (or other Cloud providers) AutoScaling feature with the Amazon’s Elastic Load Balancer and carrying out intensive tests to measure its performance and its availability. Using load balancing resulted in this thesis as the alternative with the best performance (smaller response time), and combining this option with the auto scaling feature may imply that the unlimited number of requests that can be handled by using auto scaling (keeps scaling out as needed) is combined with the best performance alternative.

Another recommendation includes carrying out the tests using Amazon’s Simple Queue Service (SQS) in case scaling using a much bigger amount of traffic is required. The Amazon SQS is a message queuing service that makes it possible to store incoming requests that still need to be processed by the application. It is used to decouple components in application so they run independently.
PART III

Conclusions and recommendations
Chapter 8
Conclusions

In this thesis the IP Multimedia Subsystem architectural framework was researched with the main purpose of designing and implementing a mobile messaging solution, this way creating high messaging traffic. The OpenNebula open source cloud solution was also researched with the purpose to design and implement a private cloud solution using proprietary infrastructure. These solutions were abandoned because of the lack of proper resources, and the direction of the thesis project changed focusing on an HTTP messaging solution deployed on the Amazon AWS Cloud. The research goal of this thesis is to determine to what extent implementing messaging services on the Amazon Cloud improves the availability of mobile messaging services provided by telecom operators and reduce costs. This thesis provides a description of the performance tests that were carried out using a custom implemented HTTP-based messaging solution in the Amazon Cloud to determine the above-mentioned.

8.1 Evaluation of used messaging technologies

This section will synthesize the findings to answer the following research question:

**RQ1**  How are mobile messaging services implemented using the HTTP protocol?

At the start of this thesis project the IP Multimedia Subsystem architectural framework was researched, including its definition, characteristics, benefits, security issues and possible open source implementations. After this technology was abandoned as a suitable solution in this thesis project, the HTTP protocol was used to implement a Java-based messaging service using the Eclipse IDE, Java Enterprise Edition, Apache Maven, Apache Tomcat, the Spring Web MVC framework and the Apache HTTPComponents API to make the sending and receiving of HTTP requests and responses possible. SMS traffic is generated by using URLs, i.e. an SMS sent by a user is represented as the following example HTTP URL, which is sent and subsequently received by the Spring servlet DispatcherServlet:


This request is sent to activate an example service Blackberry using the keyword BB_ON for the subscriber with phone number 59995209247. The request is sent to the messaging-gateway application. Using the pattern /rest/* in the request the Spring servlet DispatcherServlet handles the request and delegates it to the RequestProtocolHandler controller, which processes the request, forwards it to other components in the message flow, or corresponding response. This way a messaging solution was developed and made possible that generates messaging traffic. This solution was afterwards deployed in the Cloud, which is described in the next section. For the detailed description of the HTTP messaging service design and implementation, please refer to sections 6.2 and 6.3.

8.2 Evaluation of used cloud solution

This section will synthesize the findings to answer the following research question:

**RQ2**  How can mobile messaging services be deployed in the Amazon Cloud?
In the beginning the OpenNebula open source cloud solution was researched and no successful implementation was achieved due to the lack of the proper hardware, i.e. servers that support virtualization. Hence, the decision was made to abandon the open source private cloud implementation and use a popular public cloud solution, with a good reputation. In this case, the company where the thesis is executed started using the Amazon Cloud at more or less the same time when executing this project. This made it easier to start using and get approval by the company to finance the use of this solution. The Amazon AWS Platform is a powerful Cloud solution, in the sense that Amazon is one of the biggest providers and is generally considered the market leader in terms of computing capacity.

Using Amazon’s web-based user interface all three scenarios were implemented and configured, i.e. the dedicated server, auto-scaling and load balancing scenarios. Server instances (Amazon EC2 instances) were created using Amazon Machine Images and appropriate instance types, including type and size of space, memory and processor. AMIs are templates that contain a software configuration, including an operating system that define the operating environment. For the dedicated server scenario a single Amazon EC2 instance was used for the execution of the performance tests. The auto-scaling scenario implies a web service that provides the ability to automatically launch or terminate Amazon EC2 instances based on user-defined policies, health status checks, and schedules. With Auto Scaling, EC2 instances are categorized into Auto Scaling groups for the purposes of instance scaling and management. With auto scaling groups the minimum, maximum or desired number of running EC2 instances the auto scaling group must have, can be defined. An auto-scaling group uses a launch configuration to launch EC2 instances. With the Auto Scaling policy, which is attached to the already created Auto Scaling group, metrics can be defined for scaling in and scaling out. Auto scaling on EC2 is based on triggers from a CloudWatch alarm. CloudWatch is a web service that provides monitoring for AWS cloud resources. This alarm watches over the scale-in and sale-out metrics that were specified by the Auto Scaling policy. Concerning the load-balancing scenario, the Amazon Elastic Load Balancing solution was used, which distributes incoming application traffic across multiple Amazon EC2 instances. These three infrastructures were used to deploy the HTTP-based messaging service using Tomcat web server on the EC2 instances. Apache JMeter, an open source application designed to load test functional behavior and measure performance, was used in combination with the JMeter-EC2 shell script, which automates running Apache JMeter on Amazon EC2 instances or any other server, to carry out the load tests to test the availability of and performance in all three scenarios. This way using this platform the required tests were carried out without problems concerning the Amazon Cloud itself.

For the detailed description of the tools used, the design and the implementation of the cloud infrastructure using Amazon Web Services, and the deployment of the messaging service in the Cloud, please refer to section 6.4, 6.5 and 6.6, respectively. The following sub-section summarizes the issues encountered.

**Issues encountered and solutions**

This section will synthesize the findings to answer the following research question:

**RQ3** Which issues exist and should be tackled when providing mobile messaging services in the Amazon cloud?

Problems that were encountered during the implementation of the Cloud infrastructure included low-level Operating System kernel problems, concerning kernel parameters limiting the number of TCP/IP connections that can be opened. Also the Java Virtual Machine heap size needed to be adapted. The JVM heap size determines how often and how long the VM spends collecting garbage (WebLogic
Server Performance and Tuning, 2014). These parameters needed to be adapted to make the proper execution of performance tests possible in the Amazon Cloud.

Another encountered issue during the implementation and testing of the Auto Scaling web service was that the documentation of Amazon Auto Scaling did not clearly indicate to create a load balancer to route incoming requests to the newly created instance. After scaling takes place and an additional instance is added to the auto-scaling group, incoming requests are routed to both instances behind the load balancer based on round robin distribution.

Another issue involved after extensive testing that the messaging service did not stress the processor, but instead kept consuming memory till all memory was out, causing tomcat instances to be killed by the operating system. Hence, the available CloudWatch alarm based on the CPU utilization metric could not be used. Instead a CloudWatch alarm that is associated with a memory-utilization metric was required. By default, CloudWatch does not collect data about memory usage on an EC2 instance. Therefore, a custom metric was setup to monitor memory usage. This metric is subsequently attached to an alarm, and the auto scaling policy is based on that alarm. For an elaboration of the issues and recommendations, please refer to section 6.7.

8.3 Evaluation of availability

Based on the results acquired by executing the performance tests in all three defined scenarios, these scenarios were compared with each other to draw conclusions about the availability in each scenario.

The following was determined:

• **Scenario 1: Using a dedicated server**

  *Stated hypothesis*
  The availability of services at event-based peaks of traffic is not preserved using a dedicated server, when the capacity of the infrastructure is reached.

  Based on the listed test results in tabular form, this hypothesis is supported, because at approximately 36000 requests sent (which in this case is considered the peak of requests), the server can no longer handle incoming requests, resulting in errors.

• **Scenario 2: Using auto-scaling technology**

  *Stated hypothesis*
  The availability of services at event-based peaks of traffic is preserved using auto-scaling technology, when the capacity of the infrastructure is reached.

  Based on the listed test results in tabular form, this hypothesis is supported, because at the same maximum number of requests and higher, still the requests are successfully processed with no errors occurring.

• **Scenario 3: Using load balancing**

  *Stated hypothesis*
  The availability of services at event-based peaks of traffic is preserved using load-balancing technology, when the capacity of the infrastructure is reached.
Based on the listed test results in tabular form, this hypothesis is supported, because at the same maximum number of requests and higher, still the requests are successfully processed with no errors occurring.

**Availability conclusion**

Based on these previous results the following research questions with corresponding answers are listed.

**RQ4**  *Is the availability of mobile messaging services preserved using a dedicated server, when the capacity of the infrastructure is reached?*

Based on the previous results it can be concluded that the availability of mobile messaging services is *not preserved* using a dedicated server, when the capacity of the infrastructure is reached.

**RQ5**  *Is the availability of mobile messaging services preserved using auto-scaling technology, when the capacity of the infrastructure is reached?*

Based on the previous results it can be concluded that the availability of mobile messaging services is *preserved* using auto-scaling technology, when the capacity of the infrastructure is reached.

**RQ6**  *Is the availability of mobile messaging services preserved using load balancing, when the capacity of the infrastructure is reached?*

Based on the previous results it can be concluded that the availability of mobile messaging services is *preserved* using load balancing, when the capacity of the infrastructure is reached.

### 8.4 Evaluation of performance

Based on the results acquired by executing the performance tests in all three defined scenarios, these scenarios were compared with each other to draw conclusions about the performance in each scenario. The following was determined:

- **Scenario 1: Using a dedicated server**

  **Stated hypothesis**
  
  The response time for a subscriber’s request for service improves when using auto-scaling technology instead of a dedicated server.

  Based on the listed test results and performing the paired sample t-test on the dedicated server and auto-scaling population, the stated hypothesis cannot be accepted. We fail to reject the stated null hypothesis, which states that there is no difference between the response times in the two populations, and conclude that there is insufficient evidence to suggest that the response time for a subscriber’s request for service improves when using auto-scaling technology instead of a dedicated server.

- **Scenario 2: Using auto-scaling technology**

  **Stated hypothesis**
  
  The response time for a subscriber’s request for service improves when using load balancing instead of a dedicated server.

  Based on the listed test results and performing the paired sample t-test on the dedicated server and load balancing population, using both the Amazon Elastic Load Balancer and the Apache2 web server...
as the load balancer, the stated hypothesis is supported. The stated null hypothesis is rejected as the resulting one-tailed p-value using SPSS is positive and thus in the same direction as the stated hypothesis, and is smaller than 0.025 (0.05 / 2), which means that there is less than 2.5% chance of getting the observed differences by chance. Thus, the stated hypothesis is concluded.

• **Scenario 3: Using load balancing**

*Stated hypothesis*

The response time for a subscriber’s request for service improves when using load balancing instead of auto-scaling technology.

Based on the listed test results and performing the paired sample t-test on the auto-scaling and load balancing population, using both the Amazon Elastic Load Balancer and the Apache2 web server as the load balancer, the stated hypothesis is supported. The stated null hypothesis is rejected as the resulting one-tailed p-value using SPSS is positive and thus in the same direction as the stated hypothesis, and is smaller than 0.025 (0.05 / 2), which means that there is less than 2.5% chance of getting the observed differences by chance. Thus, the stated hypothesis is concluded.

**Performance conclusion**

Based on these previous results the following research questions with corresponding answers are listed.

**RQ7** *How is the response time for a subscriber’s request for service influenced using auto-scaling technology compared to using a dedicated server?*

Based on the previous results it cannot be concluded that the response time for a subscriber’s request for service improves when using auto-scaling technology compared to using a dedicated server. Based on the result of the carried out statistical t-test, it can be concluded that there is insufficient evidence to suggest that the response time for a subscriber’s request for service improves when using auto-scaling technology instead of a dedicated server.

**RQ8** *How is the response time for a subscriber’s request for service influenced using load balancing compared to using a dedicated server?*

Based on the previous results it can be concluded that the response time for a subscriber’s request for service improves when using load balancing compared to using a dedicated server.

**RQ9** *How is the response time for a subscriber’s request for service influenced using auto-scaling technology compared to using load balancing?*

Based on the previous results it can be concluded that the response time for a subscriber’s request for service improves when using load balancing compared to using auto-scaling technology.

**8.5 Evaluation of costs involved using Amazon AWS**

This section will synthesize the findings to answer the last three stated research questions. Based on Table 7.34 the following answers can be provided to the research questions.

**RQ10** *What are the costs involved in deploying messaging services in the Amazon Cloud on a dedicated server?*
The costs involved in deploying services in the Amazon Cloud on a dedicated server for a period of 3 years, include the following three options:

<table>
<thead>
<tr>
<th>Product</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved Instance Heavy 3-Year Term</td>
<td>$8,302.24</td>
</tr>
<tr>
<td>On demand EC2 instance</td>
<td>$23,328.00</td>
</tr>
<tr>
<td>Dedicated On-Demand instance per region fee</td>
<td>$77,500.80</td>
</tr>
</tbody>
</table>

**RQ11** *What are the costs involved in deploying messaging services in the Amazon Cloud using Auto Scaling?*

The costs involved in deploying services in the Amazon Cloud using auto-scaling technology for a period of 3 years are:

<table>
<thead>
<tr>
<th>Product</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Scaling</td>
<td>$15,233.76</td>
</tr>
</tbody>
</table>

**RQ12** *What are the costs involved in deploying messaging services in the Amazon Cloud using load-balancing technology?*

The costs involved in deploying services in the Amazon Cloud using load balancing for a period of 3 years, include the following two options:

<table>
<thead>
<tr>
<th>Product</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load balancing with ELB and 2 EC2 instances</td>
<td>$23,981.76</td>
</tr>
<tr>
<td>Load balancing with 3 EC2 instances</td>
<td>$34,992.00</td>
</tr>
</tbody>
</table>

Based on the results acquired from the performance tests, to test the availability and the response time of requests in each scenario, combined with the costs involved, users or future users of the Cloud can get a good impression of what the given infrastructure configurations can provide concerning performance and costs involved to make these possible and implement them in the real world. Based on the performance results and costs involved, it can be stated that advantages in costs may have disadvantages in performance of the system, i.e. higher response times. The more one is willing to pay for the environment, the better the response could and should be, as more resources are available. The purpose is to find the best combination of applications performing well with the adequate investment for each business. For an elaboration of the costs involved and the evaluation of these costs, please refer to section 7.3

### 8.6 Overall conclusion

This section synthesizes the findings to answer the main problem statement:

*To what extent will implementing messaging services on the Amazon Cloud improve the availability of mobile messaging services provided by telecom operators and reduce infrastructure costs?*

Using cloud computing Auto Scaling technology it was proven that the availability of services is preserved with high workloads at the particular request peak. When the memory usage of an instance reaches its maximum value, services running on webserver instances are killed by the operating system. Using Auto Scaling this can be prevented creating an alarm that watches over the memory utilization metric, and fires an alarm when the memory usage threshold is reached. Hence, an
additional instance is added behind the load balancer, this way enabling the load balancer to distribute the load between more instances.

It can be concluded that availability of services is improved when using the Amazon Cloud, or a cloud service in general when using the auto-scaling feature, as this technology ensures that new server instances are seamlessly added to the existing pool of server instances during demand peaks, and decreased during demand drops. Concerning the costs involved, businesses requiring high availability of service, and that experience dynamically changing high workloads at different peak hours weekly or monthly, the auto scaling feature is an adequate option to reduce costs as server instances are dynamically added and dropped to fit the required demand, using a pay-per-use cost model.

8.7 Concluding remarks

Based on the cost analysis that was performed and the described performance analysis, businesses, including services providers, can get insight into the costs involved in the setup of a cloud infrastructure provided by Amazon. Amazon provides a more secure Cloud solution, namely the Virtual Private Cloud, in this thesis the Dedicated On-Demand instance or the Dedicated Reserved Instance, which is a secure, private and isolated section of the AWS cloud. In case security is highly required, sensitive data or applications can be deployed using a VPC.

8.8 Contribution

This thesis serves as a base for future work in the area of Cloud computing and messaging services, using both the IMS architectural framework and the HTTP protocol. Based on the acquired test results future work can be realized by carrying out similar tests using more features provided by Amazon AWS, but not only limited to Amazon. Also other Cloud solutions, both commercial and open sources solutions can be researched and tested to provide more insight in their capabilities and possibly optimizing existing solutions with new findings.
Chapter 9
Recommendations

Recommendations for achieving high availability of mobile messaging services include:

1. The Amazon Auto Scaling service provides the capability of configuring an auto-scaling group, auto-scaling policies and alarms that trigger policies to scale-in or scale-out instances. It is important to analyze and test the scaling feature, so that the added resource, i.e. the extra instance starts running just in time, just when required. Required CloudWatch alarms need to be configured in such a way that the added resource starts functioning at the right moment. Based on tests carried out, this added instance takes time to startup and be checked by Amazon to be ready for production.

2. Businesses that require high availability of services and good performance (low response time), but that do not experience dynamically changing workloads at peak hours, are better off using the Elastic Load Balancing service provided by Amazon, as this solution guarantees, based on executed tests, relatively low response times.

3. Businesses that provide memory intensive (messaging) services can implement the script as described in section 6.4.5.1, that used the Memory Utilization metric. This way scaling is made possible based on memory usage, as this feature is not yet implemented by Amazon.

9.1 Limitations and future work

During the execution of this technical project complications were encountered for the IP Multimedia Subsystem topic, including a lack of required knowledge in the areas of telecommunication and network infrastructure design and implementation. Also there was a lack of proper resources, i.e. proper servers, for the design and implementation of IP Multimedia Subsystem architecture in the Cloud using open source solutions. For the realization of the open source IMS messaging solution in the Cloud more research has to be carried out in the areas of telecommunications, networking and network infrastructure design and implementation. If research has to be done in the field of Cloud computing with the purpose to design and implement a proper private cloud using open source solutions, it is recommended to make sure beforehand that the proper infrastructure is available, i.e. the required servers with the required features.

Concerning the implementation of the HTTP messaging solution in the Cloud, due to time constraints the maximum number of the two described load balancing scenarios could not be verified based on tests. To more effectively test the load-balancing scenario, the Linux kernel should be more optimized, making it possible that enough connections can be established, i.e. enabling a much larger number of requests to be received by the tested machine.

Although a lot of time and effort was invested in doing this research and carrying out the described tests, there exist the limitation that the sample of tests carried out is too small. The larger the sample the more accurate the results are, even though the t-test is considered the most powerful parametric test for calculating the significance of a small sample mean, as is the case in this thesis.
Bibliography


http://cloudstack.apache.org/

http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html


Archives.OpenNebula.org (2014). *Planning the Installation 3.2*. Retrieved from:
http://archives.opennebula.org/documentation:rel3.2.bck:plan


http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4343398


http://www.businessinsider.com/10-most-important-in-cloud-computing-2013-4?op=1


http://www.computerworld.com/s/article/9180941/Building_a_private_cloud_Get_ready_for_a_bumpy_ride


https://www.cert.org/insider-threat/

http://www.ietf.org/rfc/rfc3015.txt

http://www.ebizq.net/blogs/saasweek/2008/03/distinguishing_cloud_computing/

http://www.ijitee.org/attachments/File/v2i3/C0453022313.pdf

Dialogic Corporation (2009). The Architecture and Benefits of IMS. Retrieved from:

https://www.ietf.org/rfc/rfc2246.txt


http://www.facweb.iitkgp.ernet.in/~pallab/mob_com/Ericsson_Intro_to_IMS.pdf


ETSI TC-SMG (1996). TS 03.02: Digital cellular telecommunications system (Phase 2+); Network architecture. Retrieved from:
http://www.etsi.org/deliver/etsi_gts/03/0302/05/01.00_60/gsmts_0302v050100p.pdf


### Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2G</td>
<td>Second Generation</td>
</tr>
<tr>
<td>3G</td>
<td>Third Generation</td>
</tr>
<tr>
<td>3GPP</td>
<td>Third Generation Partnership Project</td>
</tr>
<tr>
<td>AAA</td>
<td>Authentication, Authorization and Accounting</td>
</tr>
<tr>
<td>ADSL</td>
<td>Asymmetric Digital Subscriber Line</td>
</tr>
<tr>
<td>AH</td>
<td>Authentication Header</td>
</tr>
<tr>
<td>AKA</td>
<td>Authentication Key Agreement</td>
</tr>
<tr>
<td>Amazon EC2</td>
<td>Amazon Elastic Compute Cloud</td>
</tr>
<tr>
<td>AMI</td>
<td>Amazon Machine Image</td>
</tr>
<tr>
<td>ANSI-136</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ARN</td>
<td>Amazon Resource Name</td>
</tr>
<tr>
<td>ARPU</td>
<td>Average Revenue Per User</td>
</tr>
<tr>
<td>AS</td>
<td>Application Server</td>
</tr>
<tr>
<td>AUC</td>
<td>Authentication Center</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>BGCF</td>
<td>Breakout Gateway Control Function</td>
</tr>
<tr>
<td>BICC</td>
<td>Bearer Independent Call Control</td>
</tr>
<tr>
<td>BSC</td>
<td>Base Station Controller</td>
</tr>
<tr>
<td>BSS</td>
<td>Base Station Subsystem</td>
</tr>
<tr>
<td>BSS-</td>
<td>Business support system</td>
</tr>
<tr>
<td>BTS</td>
<td>Base Transceiver Station</td>
</tr>
<tr>
<td>CaaS</td>
<td>Communication as a Service</td>
</tr>
<tr>
<td>CAMEL</td>
<td>Customized Applications for Mobile network Enhanced Logic</td>
</tr>
<tr>
<td>CDMA</td>
<td>Code Division Multiple Access</td>
</tr>
<tr>
<td>CGI</td>
<td>Common Gateway Interface</td>
</tr>
<tr>
<td>CIMD</td>
<td>Computer Interface to Message Distribution</td>
</tr>
<tr>
<td>CLI</td>
<td>Command Line Interface</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CSA</td>
<td>Cloud Security Alliance</td>
</tr>
<tr>
<td>CSCF</td>
<td>Call/Session Control Function</td>
</tr>
<tr>
<td>CSI</td>
<td>Customer Subscriber Integrator</td>
</tr>
<tr>
<td>CTS</td>
<td>Converged Telephony Server</td>
</tr>
<tr>
<td>Daas</td>
<td>Data-Storage as a Service</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>DDoS</td>
<td>Distributed Denial of Service</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>DSL</td>
<td>Digital Subscriber Line</td>
</tr>
<tr>
<td>EC2</td>
<td>Elastic Compute Cloud</td>
</tr>
<tr>
<td>EDGE</td>
<td>Enhanced Data rates for GSM Evolution</td>
</tr>
<tr>
<td>EIR</td>
<td>Equipment Identity Register</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ESME</td>
<td>External Short Message Entity</td>
</tr>
<tr>
<td>ESP</td>
<td>Encapsulation Security Payload</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standards Institute</td>
</tr>
<tr>
<td>GGSN</td>
<td>Gateway GPRS support node</td>
</tr>
<tr>
<td>GMSC</td>
<td>Gateway Mobile Switching Center</td>
</tr>
<tr>
<td>GPRS</td>
<td>General Packet Radio Service</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile</td>
</tr>
<tr>
<td>GSN</td>
<td>GPRS Support Node</td>
</tr>
<tr>
<td>HLR</td>
<td>Home Location Register</td>
</tr>
<tr>
<td>HSCSD</td>
<td>High Speed Circuit Switched Data</td>
</tr>
<tr>
<td>HSS</td>
<td>Home Subscriber Server</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
</tr>
<tr>
<td>ICP</td>
<td>IMS Client Platform</td>
</tr>
<tr>
<td>I-CSCF</td>
<td>Interrogating Call/Session Control Function</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>iDEN</td>
<td>Integrated Digital Enhanced Network</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IMEI</td>
<td>International Mobile Equipment Identity</td>
</tr>
<tr>
<td>IMPI</td>
<td>IP Multimedia Private identity</td>
</tr>
<tr>
<td>IMPU</td>
<td>IP Multimedia Public Identity</td>
</tr>
<tr>
<td>IMS</td>
<td>IP Multimedia Subsystem</td>
</tr>
<tr>
<td>IMS-ALG</td>
<td>IMS Application Layer Gateway</td>
</tr>
<tr>
<td>IMSI</td>
<td>International Mobile Subscriber Identity</td>
</tr>
<tr>
<td>IM-SSF</td>
<td>IP Multimedia Service Switching Function</td>
</tr>
<tr>
<td>IMT-2000</td>
<td>International Mobile Telecommunications-2000</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IP-CAN</td>
<td>IP Connectivity Access Network</td>
</tr>
<tr>
<td>IPsec ESP</td>
<td>Internet Protocol Security Encapsulation Security Payload</td>
</tr>
<tr>
<td>IS-95</td>
<td>Interim Standard 95</td>
</tr>
<tr>
<td>ISIM</td>
<td>IMS Subscriber Identity Module</td>
</tr>
<tr>
<td>Java EE (J2E)</td>
<td>Java Enterprise Edition</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>LA</td>
<td>Location Area</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MEGACO</td>
<td>Media Gateway Control</td>
</tr>
<tr>
<td>MGCF</td>
<td>Media Gateway Control Function</td>
</tr>
<tr>
<td>MGW</td>
<td>Media Gateway</td>
</tr>
<tr>
<td>MIME</td>
<td>Multipurpose Internet Mail Extensions</td>
</tr>
<tr>
<td>MMS</td>
<td>Multimedia Messaging Service</td>
</tr>
<tr>
<td>MRF</td>
<td>Media Resource Function</td>
</tr>
<tr>
<td>MRFC</td>
<td>Media Resource Function Controller</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MRFP</td>
<td>Media Resource Function Processor</td>
</tr>
<tr>
<td>MS</td>
<td>Mobile Station</td>
</tr>
<tr>
<td>MSC</td>
<td>Mobile Switching Center</td>
</tr>
<tr>
<td>MSISDN</td>
<td>Mobile Subscriber Integrated Services Digital Network-number</td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller</td>
</tr>
<tr>
<td>NAI</td>
<td>Network Access Identifier</td>
</tr>
<tr>
<td>NAS</td>
<td>Network Access Server</td>
</tr>
<tr>
<td>NAS</td>
<td>Network-Attached Storage</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address (and Port) Translation</td>
</tr>
<tr>
<td>NGN</td>
<td>Next Generation Networking</td>
</tr>
<tr>
<td>OCCI</td>
<td>Open Cloud Computing Interface</td>
</tr>
<tr>
<td>OMA</td>
<td>Open Mobile Alliance</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>OSA</td>
<td>Open Service Access</td>
</tr>
<tr>
<td>OSA-GW</td>
<td>Open Services Access - Gateway</td>
</tr>
<tr>
<td>OSA-SCS</td>
<td>Open Service Access-Service Capability Server</td>
</tr>
<tr>
<td>OSS</td>
<td>Operational support system</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>PCM</td>
<td>Pulse Code Modulation</td>
</tr>
<tr>
<td>P-CSCF</td>
<td>Proxy Call/Session Control Function</td>
</tr>
<tr>
<td>PDC</td>
<td>Personal Digital Cellular</td>
</tr>
<tr>
<td>PDP</td>
<td>Packet Data Protocol</td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol Data Unit</td>
</tr>
<tr>
<td>PGM</td>
<td>Presence, Group &amp; Data Management</td>
</tr>
<tr>
<td>PLMN</td>
<td>Public Land Mobile Network</td>
</tr>
<tr>
<td>POM</td>
<td>Project Object Model</td>
</tr>
<tr>
<td>PSTN</td>
<td>Public Switched Telephone Network</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RADIUS</td>
<td>Remote Authentication Dial-In User Service</td>
</tr>
<tr>
<td>RDS</td>
<td>Relational Database Services</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RI</td>
<td>Reserved Instance</td>
</tr>
<tr>
<td>RIM</td>
<td>Research in Motion</td>
</tr>
<tr>
<td>ROI</td>
<td>Return On Investment</td>
</tr>
<tr>
<td>RTCP</td>
<td>RTP Control Protocol</td>
</tr>
<tr>
<td>RTP</td>
<td>Real-time Transport Protocol</td>
</tr>
<tr>
<td>S3</td>
<td>Simple Storage Service</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SAN</td>
<td>Storage Area Network</td>
</tr>
<tr>
<td>SCP</td>
<td>Secure-Copy</td>
</tr>
<tr>
<td>S-CSCF</td>
<td>Serving Call/Session Control Function</td>
</tr>
<tr>
<td>SEG</td>
<td>Security Gateway</td>
</tr>
<tr>
<td>SER</td>
<td>SIP Express Router</td>
</tr>
<tr>
<td>SGSN</td>
<td>Serving GPRS Support Node</td>
</tr>
<tr>
<td>SGW</td>
<td>Signaling Gateway Function</td>
</tr>
<tr>
<td>SIM</td>
<td>Subscriber Identity Module</td>
</tr>
</tbody>
</table>
SIP  Session Initiation Protocol
SIP-AS  Session Initiation Protocol-Application Server
SLA  Service Level Agreement
SLF  Subscription Locator Function
SLF4J  Simple Logging Façade for Java
SMPP  Short Message Peer-to-Peer
SMS  Short Message Service
SMSC  Short Message Service Center
SMTP  Simple Mail Transfer Protocol
SOAP  Simple Object Access Protocol
SQS  Simple Queue Service
SSH  Secure Shell
SSL  Secure Socket Layer
SVN  Sub Version
TAS  Telephony Application Server
TCP  Transfer Control Protocol
TCP/IP  Transfer Control Protocol/ Internet Protocol
TDMA  Time Division Multiple Access
THIG  Topology Hiding Interface Gateway
TISPAN  Telecommunications and Internet converged Services and Protocols for Advanced Networking
TLS  Transfer Layer Security
TMSI  Temporal Mobile Subscriber Identity
TrGW  Transition Gateway
TTS  Text-To-Speech
UDP  User Datagram Protocol
UE  User Equipment
UI  User Interface
UICC  Universal Integrated Circuit Card
UMTS  Universal Mobile Telecommunications System
UNAB  Unified Network Address Book
URI  Uniform Resource Identifier
URL  Uniform Resource Locator
User Agent  UA
USSD  Unstructured Supplementary Services Data
vCPU  Virtual CPU
VLR  Visited Location Register
VM  Virtual Machines
VMM  Virtual Machine Monitor
VoIP  Voice over IP
VPC  Virtual Private Cloud
VPS  Voice Processing System
W3C  World Wide Web Consortium
WAP  Wireless Application Protocol
WAR  Web application ARchive
W-CDMA  Wideband-Code Division Multiple Access
WiMAX  Worldwide Interoperability for Microwave Access
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XML-RPC</td>
<td>eXtensible Markup Language-Remote Procedure Call</td>
</tr>
<tr>
<td>XP</td>
<td>eXtreme Programming</td>
</tr>
</tbody>
</table>
## Appendix A: Clusters, Grids and Clouds characteristics

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Clusters</th>
<th>Grids</th>
<th>Clouds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population</td>
<td>Commodity computers</td>
<td>High-end computers (servers, clusters)</td>
<td>Commodity computers and high-end servers and network attached storage</td>
</tr>
<tr>
<td>Size / Scalability</td>
<td>100s</td>
<td>100s</td>
<td>100s to 100s</td>
</tr>
<tr>
<td>Node Operating System (OS)</td>
<td>One of the standard OSs (Linux, Windows)</td>
<td>Any standard OS (dominated by Unix)</td>
<td>A hypervisor (VM) on which multiple OSs run</td>
</tr>
<tr>
<td>Ownership</td>
<td>Single</td>
<td>Multiple</td>
<td>Single</td>
</tr>
<tr>
<td>Interconnection Network / Speed</td>
<td>Dedicated, high-end with low latency and high bandwidth</td>
<td>Mostly Internet with high latency and low bandwidth</td>
<td>Dedicated, high-end with low latency and high bandwidth</td>
</tr>
<tr>
<td>Security/Privacy</td>
<td>Traditional login/password-based. Medium level of privacy – depends on user privileges.</td>
<td>Public/private key pair based authentication and mapping a user to an account. Limited support for privacy.</td>
<td>Each user/application is provided with a virtual machine. High security/privacy is guaranteed. Support for setting per-file access control list (ACL)</td>
</tr>
<tr>
<td>Discovery</td>
<td>Membership services</td>
<td>Centralised indexing and decentralised info services</td>
<td>Membership services</td>
</tr>
<tr>
<td>Service Negotiation</td>
<td>Limited</td>
<td>Yes, SLA based</td>
<td>Yes, SLA based</td>
</tr>
<tr>
<td>User Management</td>
<td>Centralised</td>
<td>Decentralised and also Virtual Organization (VO)-based</td>
<td>Centralised or can be delegated to third party</td>
</tr>
<tr>
<td>Resource Management</td>
<td>Centralised</td>
<td>Distributed</td>
<td>Centralised/Distributed</td>
</tr>
<tr>
<td>Allocation / Scheduling</td>
<td>Centralised</td>
<td>Decentralised</td>
<td>Both centralised/decentralised</td>
</tr>
<tr>
<td>Standards / Interoperability</td>
<td>Virtual Interface Architecture (VIA)-based</td>
<td>Some Open Grid Forum standards</td>
<td>Web Services (SOAP and REST)</td>
</tr>
<tr>
<td>Single System Image</td>
<td>Yes</td>
<td>No</td>
<td>Yes, but optional</td>
</tr>
<tr>
<td>Capacity</td>
<td>Stable and guaranteed</td>
<td>Varies, but high</td>
<td>Provisioned on demand</td>
</tr>
<tr>
<td>Failure Management (Self-healing)</td>
<td>Limited (often failed tasks/applications are restarted).</td>
<td>Limited (often failed tasks/applications are restarted).</td>
<td>Strong support for failover and content replication. VMs can be easily migrated from one node to other.</td>
</tr>
<tr>
<td>Pricing of Services</td>
<td>Limited, not open market</td>
<td>Dominated by public good or privately assigned</td>
<td>Utility pricing, discounted for larger customers</td>
</tr>
<tr>
<td>Internetworking</td>
<td>Multi-clustering within an Organization</td>
<td>Limited adoption, but being explored through research efforts such as Gridbus InterGrid</td>
<td>High potential, third party solution providers can loosely tie together services of different Clouds</td>
</tr>
<tr>
<td>Application Drivers</td>
<td>Science, business, enterprise computing, data centers</td>
<td>Collaborative scientific and high throughput computing applications</td>
<td>Dynamically provisioned legacy and web applications, content delivery</td>
</tr>
<tr>
<td>Potential for Building 3rd Party or Value-added Solutions</td>
<td>Limited due to rigid architecture</td>
<td>Limited due to strong orientation for scientific computing</td>
<td>High potential – can create new services by dynamically provisioning of compute, storage, and application services and offer as their own isolated or composite Cloud services to users</td>
</tr>
</tbody>
</table>
Appendix B: Open IMS Core installation instructions

Overview
This page is supposed to help you jump-start the Open IMS Core. In the SVN distribution, the whole things is pre-configured and pre-provisioned with a set of minimal data that should be enough for a simple "Alice-calls-Bob" trial.

Quick Install
If you already went once through the installation procedure, this section will help refresh your memory on what steps you are supposed to follow. If this is your first time, skip over it for now and start with Step 1.

mkdir /opt/OpenIMSCore
cd /opt/OpenIMSCore
mkdir FHoSS
svn checkout http://svn.berlios.de/svnroot/repos/openimscore/FHoSS/trunk FHoSS
mkdir ser_ims
svn checkout http://svn.berlios.de/svnroot/repos/openimscore/ser_ims/trunk ser_ims
cd FHoSS
ant compile deploy
cd ..
cd ser_ims
make install-libs all
cd ..
mysql -u root -p < FHoSS/scripts/hss_db.sql
mysql -u root -p < FHoSS/scripts/userdata.sql
mysql -u root -p < ser_ims/cfg/icscf.sql
cp ser_ims/cfg/*.cfg .
cp ser_ims/cfg/*.xml .
cp ser_ims/cfg/*.sh .

Step 1: Prerequisites
  • Hardware requirements
    o A current Linux desktop class machine should be enough
    o If you want to get ultimate performance:
      ▪ Add several Gigabytes of RAM
      ▪ Have as many CPUs/Cores as
      ▪ Gigabit Ethernet would help
  • Network access
    o A current Linux desktop class machine should be enough
    o Inter-domain NAT is not something we are interested in, so a public IP address would be great
    o Controllable DNS server if you don’t want to have one on your Linux box
  • Software requirements
    o ~100 MBytes of disk space to be on the safe side
    o GCC3/4, make, JDK1.5, ant
    o MySQL installed and started (or other DBMS if you can deal with it)
    o bison, flex
    o libxml2 (> 2.6), libmysql - both with development
Linux kernel 2.6 and ipsec-tools (setkey) if you want to use IPSec security
- curl and libcurl4-gnutls-dev for the LoST interface of the E-CSCF
- Optional: openssl if you would like to enable the TLS security
- bind installed and running (or other name server if you can deal with it)
- Browser on the box or that can connect to the box (for user provisioning)

**Note:** we consider that you have all this installed, configured and running. **Note:** we assume for now that you want to install the whole thing on just 1 box.

### Step 2: Get the Source Code

- **Where?** - Fresh code at [http://svn.berlios.de/svnroot/repos/openimscore](http://svn.berlios.de/svnroot/repos/openimscore) (you will need to have Subversion installed). On this page you can find a lot more information about the sources.

- **What?**
  - The CSCFs: ser_ims/trunk
  - The HSS: FHoSS/trunk

- **How?** - The source code is pre-configured to work from a standard file path:
  - Create /opt/OpenIMSCore and go there

```
mkdir /opt/OpenIMSCore
cd /opt/OpenIMSCore
```

- Create a new directory ser_ims and checkout the CSCFs there:

```
mkdir ser_ims
svn checkout http://svn.berlios.de/svnroot/repos/openimscore/ser_ims/trunk
```

- Create a new directory FHoSS and checkout the HSS there:

```
mkdir FHoSS
svn checkout http://svn.berlios.de/svnroot/repos/openimscore/FHoSS/trunk
```

- If you would prefer another path, be prepared to edit the configuration files!

### Step 3: Compile

- **ser_ims**
  - New!!! Do "make install-libs all" in ser_ims

```
cd ser_ims
make install-libs all
```

- If something breaks, you probably don't have all the prerequisites.

- **FHoSS**
  - If you don't have a JDK >=1.5, get one before proceeding
  - Make sure, that the JDK version that you are using is >= 1.5!!!

```
# java -version java version "1.5.0_07" Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_07-b03) Java HotSpot(TM) Client VM (build 1.5.0_07-b03, mixed mode)
```

It is often the case that users have just installed a 1.5 JDK but they are still using their old JDK installation! If you see lots of errors, recheck this before posting a bug report!

- Do "ant compile deploy" in FHoSS New!!! "ant gen" is not needed any more!!!

```
cd FHoSS
ant compile ant deploy
```

- While you wait for the compilation to finish, you can go ahead and perform Step 4.

### Step 4: Configure the Environment

- **Notes:**
  - All the installation examples configured to work only on the local loopback and the default domain configured as "open-ims.test".
  - The MySQL access rights are set only for local access
  - We recommend that you try it first like this and then do your changes:

```
Replace 127.0.0.1 where required with your IP address
```
Replace the home domain (open-ims.test) with your own one

Change the database passwords
For this operation the ser_ims/cfg/configurator.sh might help you.

- **DNS**
  - A sample DNS zone file can be found in ser_ims/cfg/open-ims.dnszone
  - Copy it to your bind configuration directory
  - Edit named.conf and insert the file there (Would be great to also add reverse DNS entries)
  - Restart the name server
  - Test that the names are resolvable (don't forget about /etc/resolv.conf pointing to your new DNS server!)

- **MySQL**
  - Run the SQL dumps (mysql -u root -p -h localhost < dump.sql): **New!!!** "hssdb.sql" was replaced by "hss_db.sql" !!!
    
    mysql -u root -p -h localhost < ser_ims/cfg/icscf.sql
    mysql -u root -p -h localhost < FHoSS/scripts/hss.db.sql
    mysql -u root -p -h localhost < FHoSS/scripts/userdata.sql
  - Check if the databases are in there and accessible

---

**Step 5: Configure the IMS Core**

- By now you should have MySQL and DNS working

- **CSCFs**
  - Copy the following files to /opt/OpenMSCore or another location comfortable for you: pcscf.cfg, pcscf.sh, icscf.cfg, icscf.xml, icscf.sh, scscf.cfg, scscf.xml, scscf.sh,
  
    cp ser_ims/cfg/*.cfg .
    cp ser_ims/cfg/*.xml .
    cp ser_ims/cfg/*.sh .

- **FHoSS**
  - Take a look at the configuration files in FHoSS/deploy/ (available after Step 3 completes)
  - Edit these files to your own preferences (don't forget to update the DNS zone file accordingly and restart the name server)

---

**Step 6: Start the components**

- **CSCFs**
  - Start pcscf.sh, icscf.sh and scscf.sh
  - All these should run in parallel.
  - We love debugging, so by default they would stay in foreground.
  - By default you should see periodically log messages with the content of the registrar and with the opened diameter links

- **FHoSS**
  - Start FHoSS/deploy/startup.sh
  - If the previous step fails, check that you have the JAVA_HOME environment variable correctly exported and/or modify the script that you just tried to start.
  - Check the web interface on [http://localhost:8080/](http://localhost:8080/)
  - Check if the Diameter Peers are connecting to each other. You can see this in the console of FHoSS or in that of I/S-CSCF

---

**Step 7: Configure Subscribers**

- **FHoSS**
  - By default, FHoSS comes provisioned with a couple of sample users:

```plaintext
  alice@open-ims.test
```
• bob@open-ims.test
  o Use these or insert new ones.
• Create a Subscription
• Create a Private Identity
• Create a Public Identity
• Link them
• SIP-to-IMS Gateway
  o The SIP-to-IMS Gateway is now obsolete and was dropped from the project. See the Annex and FAQ for information on how you can use the Open IMS Core with SIP clients capable of only MD5 authentication.
• IMS User Endpoint Configuration
  o Provision with your own UE data or use one of the default users
  o Alice:
    § Private Identity: alice@open-ims.test
    § Secret Key: alice
    § OP: 0x00...0
    § AMF: 0x00...0
    § Use of Anonimity Key: enable
    § Public Identity: sip:alice@open-ims.test
    § Realm: open-ims.test
    § Strict Outbound Proxy: sip:pcscf.open-ims.test:4060
      o Bob: similar
• SIP User Endpoint Configuration
  o Alice:
    § User part of the SIP URI: alice
    § Host part of the SIP URI/Domain/realm: open-ims.test
    § Password: alice
    § Strict Outbound Proxy: sip:pcscf.open-ims.test:4060
• !!! Make sure that your SIP client does REGISTER sip:open-ims.test and not REGISTER sip:pcscf.open-ims.test:4060
• !!! Read the Annex and FAQ related to using MD5-only clients with the Open IMS Core
  o Bob: similar

Step 8: Test!
• This is the last step. You should have all installed and configured by now
• Registration uses all components and as such, it is a good test if all is up & running
• Use Wireshark to see what’s going on:
  o Monitor ports 4060, 5060 and 6060 for SIP traffic
  o Monitor ports 3868, 3869 and 3870 for Diameter traffic

Annex A - DNS HOWTO
A lot of users seem to have difficulties when setting up a DNS server. Although it is not our purpose to teach you this, here is a summary of this process. But be aware that this does not mean that we are offering any further support for it and you don't have to read the DNS manual. If you think that /etc/hosts would be enough, you are wrong as it can not help you with special DNS queries like NAPTR and SRV.

So how do you get a DNS server up and running?
• Get the bind (or often called named) package installed on your distribution
• Make sure you are root
• Locate named.conf (could be in /etc or /etc/bind or /etc/named)
• Edit that file according to your needs. Here are some things that you need in there:
  options { ... forward first; forwarders {
    [THE_IP_ADDRESS_OF_YOUR_UPSTREAM_DNS_SERVER]; ... ]; ... };
  zone "open-ims.test" IN { type master; file "pri/open-ims.dnszone"; notify no; };
On some distributions this file includes other files so be sure to dig through those also. THE_IP_ADDRESS_OF_YOUR_UPSTREAM_DNS_SERVER can be found in /etc/resolv.conf.
• Now copy the file /opt/OpenIMSCore/ser_ims/cfg/open-ims.dnszone to where you configure it above
  (pri/open-ims.dnszone)
cp /opt/OpenIMSCore/ser_ims/cfg/open-ims.dnszone /var/bind/pri/chown -R named:named /var/bind/pri/open-ims.dnszone
• Then start or re-start the DNS server (remember that these configuration files are not monitored for changes so you will have to send a SIGHUP or do a restart to reload them).
/etc/init.d/named restart
• You should now test if it works. In the response look if you got the correct answer.
dig @127.0.0.1 pcsf.open-ims.test
• To actually use it, you would need to configure it as a DNS server for your machine. Here is how your /etc/resolv.conf file should look like:
# cat /etc/resolv.conf
nameserver 127.0.0.1
search open-ims.test
dig
• Remember that utilities like the DHCP-Client overwrite this file be default!

Annex B - SIP Clients How-To

The old SIP2IMS module that was performing MDS-to-AKA authentication translation has been deleted as being obsolete as the core can perform MDS authentication too. Another reason is that it’s functionality was seriously flawed by the fact that it was a SIP proxy rather than a full B2BUA.

So how do you get a client registered?

• First provision the users (or use the default sip:alice@open-ims.test or sip:bob@open-ims.test). Make sure than in the HSS provisioning interface in the private identity configuration, you allow the use of Digest-MDS for the respective users
• Then you have two options for the S-CSCF to trigger an MDS authentication
  1. Modify the client to send a parameter "algorithm=MDS" in the Authorization header in the first unauthorized REGISTER.
  2. Or modify the scsf.cfg and enable the MDS authorization as the default authentication method instead of AKAv1-MDS.
• Next just make sure that the client is using the P-CSCF address (sip:pcscf.open-ims.test:4060 by default) as strict outbound proxy and the REGISTER Request-URI is "sip:open-ims.test" (or your own domain name) and not "sip:pcscf.open-ims.test:4060" as many SIP clients fail here.

Annex C - Changing the Domain Name and IP-Address of configuration files

OpenIMSCore is preconfigured to work with the domain "open-ims.test" and ip address of default loopback device, "127.0.0.1". Following the installation instructions above without changing the configuration files would set up this environment successfully. For some reasons (like testing roaming, communicating with other external entities), you would need to change the domain name. In that case you could use a configuration script(configurator.sh) which is stored under directory /opt/OpenIMSCore/ser_ims/cfg/
Does it only change *.cfg files?

No, it can also reconfigure *.xml, *.sql and FHoSS configuration files(*.properties).

How do I make use of configurator.sh?

- You can directly execute it and fetch the files to be changed as arguments. See the example below:

  ```bash
  # pwd /opt/OpenIMSCore
  # ser_ims/cfg/configurator.sh ser_ims/cfg/scscf.cfg
  ser_ims/cfg/icscf.xml
  FHoSS/deploy/hss_db.sql
  FHoSS/deploy/hss.properties
  ```

- Then copy the files to the /opt/OpenIMSCore directory

- Note that you have to backup your configuration files that you want to reconfigure