UNIVERSITAT POLITÈCNICA DE CATALUNYA

Analysis and Design of the Integration of Cloud Solutions into a Customized PaaS

by

Xavi Magrinyà

Director: Teemu Heikkonen
Examiner: Jordi Guitart

A thesis submitted in partial fulfillment for the degree of Master of Science

in the
Facultat d’Informàtica de Barcelona
Departament d’Arquitectura de Computadors

May 2014
Declaration of Authorship

I, XAVI MAGRINYÀ, declare that this thesis titled, ‘ANALYSIS AND DESIGN OF THE INTEGRATION OF CLOUD SOLUTIONS INTO A CUSTOMIZED PAAS’ and the work presented in it are my own. I confirm that:

■ This work was done wholly or mainly while in candidature for a research degree at this University.

■ Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.

■ Where I have consulted the published work of others, this is always clearly attributed.

■ Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.

■ I have acknowledged all main sources of help.

■ Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

________________________________________________________

Date: ___________________________________________________
“Education is the most powerful weapon which you can use to change the world.”

Nelson Mandela
Abstract

Facultat d’Informàtica de Barcelona
Departament d’Arquitectura de Computadors

Master of Science

by Xavi Magrinyà

Cloud computing has become one of the most trendy options in the software industry. Having a solid base where to build cloud solutions is necessary to enable growth in the companies. In the presented research, we aim to investigate the advantages and disadvantages of using a new customized PaaS where to run a set of cloud solutions. After describing the properties of service-oriented architecture, we analyze the viability of fitting our current cloud solutions into a customized platform as a service. For this purpose we provide an analysis of the platforms, a comparison between them, a collection of requirements to meet and a proof of concept. Then, we redesign the current architecture to take advantage of service-oriented architecture and provide some results of the redesign. Finally, we conclude that this architecture enables modularity, reusability and scalability and the needed redesign does not imply major changes in the software.
Acknowledgements

This thesis wouldn’t have been possible without the help of some people. Starting with my thesis’ directors Teemu Heikkonen and Timo Rihtniemi, I would like to thank them their support and dedication. I would like to thank also Jordi Guitart for his help and mentoring, in addition to the helpful feedback provided and willingness of support from the first day. Thanks also to people like Joni Moilanen, Alexey Otavin, Juhani Haapanen and Kim Nymberg, who helped me providing valuable information. Also thanks to Siva Soundarapandian, CPaaS architect, for his valuable time and the online conversations from India. And last but not least, thanks to Vesa Pirilä for his mentoring and support along this project.
## Contents

Declaration of Authorship ................................................................. i  

Abstract ............................................................................................. iii  

Acknowledgements ............................................................................. iv  

List of Figures .................................................................................... viii  

List of Tables ..................................................................................... ix  

Abbreviations ...................................................................................... x  

1 Introduction .................................................................................... 1  
1.1 Overview ...................................................................................... 1  
  1.1.1 Projects’ frame ........................................................................ 1  
  1.1.2 Collaborative tools and their role ............................................ 1  
  1.1.3 The current status ................................................................... 2  
1.2 Research problem .......................................................................... 2  
  1.2.1 Reasons to change to another platform .................................. 3  
1.3 The research question ................................................................... 3  
1.4 Research process ........................................................................... 4  
  1.4.1 Task list .................................................................................. 4  
1.5 Thesis planning .............................................................................. 5  
1.6 Structure of the thesis ................................................................... 9  

2 Background theory ........................................................................ 10  
2.1 Cloud computing .......................................................................... 10  
  2.1.1 Comparing IaaS and PaaS ......................................................... 11  
2.2 Service Oriented Architecture ...................................................... 12  
  2.2.1 API Manager .......................................................................... 13  
  2.2.2 Identity Server ........................................................................ 14  
  2.2.3 Enterprise Service Bus ............................................................ 14  
    2.2.3.1 ESB Internals .................................................................... 15  
  2.2.4 Application Server .................................................................. 17  
2.3 Message-oriented Middleware ..................................................... 18
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.1 Java Message Service</td>
<td>19</td>
</tr>
<tr>
<td>2.3.1.1 JMS Properties</td>
<td>20</td>
</tr>
<tr>
<td>2.3.1.2 Distribution models</td>
<td>21</td>
</tr>
<tr>
<td>2.3.2 Advanced Message Queuing Protocol</td>
<td>22</td>
</tr>
<tr>
<td>2.3.2.1 AMQP Properties</td>
<td>23</td>
</tr>
<tr>
<td>2.3.2.2 Distribution models</td>
<td>23</td>
</tr>
<tr>
<td>3 Scenario description</td>
<td>26</td>
</tr>
<tr>
<td>3.1 The requirements</td>
<td>26</td>
</tr>
<tr>
<td>3.2 Current architecture</td>
<td>28</td>
</tr>
<tr>
<td>3.2.1 Architectural components</td>
<td>29</td>
</tr>
<tr>
<td>3.3 CPaaS architecture</td>
<td>30</td>
</tr>
<tr>
<td>3.3.1 Components</td>
<td>30</td>
</tr>
<tr>
<td>3.3.1.1 API Manager</td>
<td>32</td>
</tr>
<tr>
<td>3.3.1.2 Identity Server</td>
<td>33</td>
</tr>
<tr>
<td>3.3.1.3 Enterprise Serial Bus</td>
<td>34</td>
</tr>
<tr>
<td>3.3.1.4 Message Broker</td>
<td>34</td>
</tr>
<tr>
<td>3.3.1.5 Application Deployment Service</td>
<td>35</td>
</tr>
<tr>
<td>3.3.2 Scalability</td>
<td>35</td>
</tr>
<tr>
<td>4 Analysis of the integration</td>
<td>36</td>
</tr>
<tr>
<td>4.1 Analysis of the integration with the customized PaaS</td>
<td>36</td>
</tr>
<tr>
<td>4.1.1 Common points between the platforms</td>
<td>37</td>
</tr>
<tr>
<td>4.1.2 Potential difficulties and differences</td>
<td>37</td>
</tr>
<tr>
<td>4.1.2.1 Databases slightly different</td>
<td>38</td>
</tr>
<tr>
<td>4.1.2.2 Different BLOB storage</td>
<td>38</td>
</tr>
<tr>
<td>4.1.2.3 Usage of AMQP instead of the current protocol</td>
<td>39</td>
</tr>
<tr>
<td>4.1.3 Comparison table between the options</td>
<td>39</td>
</tr>
<tr>
<td>4.1.4 Proof of concept</td>
<td>40</td>
</tr>
<tr>
<td>4.1.4.1 Use cases</td>
<td>41</td>
</tr>
<tr>
<td>4.1.4.2 Architecture overview</td>
<td>42</td>
</tr>
<tr>
<td>4.1.4.3 How it works</td>
<td>43</td>
</tr>
<tr>
<td>4.2 Viability of the project</td>
<td>43</td>
</tr>
<tr>
<td>5 Redesign of the architecture</td>
<td>45</td>
</tr>
<tr>
<td>5.1 Architecture redesign</td>
<td>45</td>
</tr>
<tr>
<td>5.1.1 Service-oriented Architecture</td>
<td>45</td>
</tr>
<tr>
<td>5.1.2 Services</td>
<td>46</td>
</tr>
<tr>
<td>5.1.3 Application changes</td>
<td>47</td>
</tr>
<tr>
<td>5.1.4 BLOB storage</td>
<td>47</td>
</tr>
<tr>
<td>5.2 Results of the redesign</td>
<td>48</td>
</tr>
<tr>
<td>6 Planning</td>
<td>50</td>
</tr>
<tr>
<td>6.1 Final planning</td>
<td>50</td>
</tr>
<tr>
<td>6.2 Problems encountered</td>
<td>53</td>
</tr>
<tr>
<td>6.3 Costs of the project</td>
<td>53</td>
</tr>
<tr>
<td>7 Conclusions</td>
<td>55</td>
</tr>
</tbody>
</table>
Contents

7.1 Analysis of the results ......................................... 55
7.2 Future work .................................................... 56

A Project Definition .............................................. 57
A.1 Introduction .................................................. 57
  A.1.1 Project’s frame .......................................... 57
  A.1.2 Project proposal ......................................... 57
A.2 Understanding the why ........................................ 58
A.3 Scope of the project .......................................... 58
A.4 Deliverables ................................................... 58
A.5 Possible problems to overcome .............................. 59
A.6 Specific conditions ........................................... 59

Bibliography .......................................................... 60
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparison of the different models offered by cloud computing providers</td>
<td>12</td>
</tr>
<tr>
<td>2.2</td>
<td>Example of how service consumers communicate to the ESB using a common protocol, and the ESB translates the request to the corresponding protocol to communicate with the service</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>Sequence of mediators</td>
<td>16</td>
</tr>
<tr>
<td>2.4</td>
<td>Message brokers like ActiveMQ allow interoperability between JMS and other messaging APIs</td>
<td>20</td>
</tr>
<tr>
<td>2.5</td>
<td>The two types of distribution models: queues and topics</td>
<td>22</td>
</tr>
<tr>
<td>2.6</td>
<td>Direct exchange routing</td>
<td>24</td>
</tr>
<tr>
<td>2.7</td>
<td>Fanout exchange routing</td>
<td>24</td>
</tr>
<tr>
<td>3.1</td>
<td>Schema of the current architecture</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>CPaaS architectural overview</td>
<td>32</td>
</tr>
<tr>
<td>3.3</td>
<td>Example of an API Store with an example API uploaded</td>
<td>33</td>
</tr>
<tr>
<td>4.1</td>
<td>Use cases of the prototype</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>Prototype architecture overview</td>
<td>42</td>
</tr>
<tr>
<td>5.1</td>
<td>External BLOB storage scheme</td>
<td>48</td>
</tr>
<tr>
<td>5.2</td>
<td>Final scheme of CPaaS and the interaction between its components</td>
<td>49</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Comparison table between the current platforms and CPaaS. . . . . . . . 40

6.1 Expenses of the project (amounts in €) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMPQ</td>
<td>Advanced Message Queuing Protocol</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BLOB</td>
<td>Binary Large Object</td>
</tr>
<tr>
<td>CPaaS</td>
<td>Customized Platform as a Service</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DDS</td>
<td>Data Distribution Service</td>
</tr>
<tr>
<td>ELB</td>
<td>Elastic Load Balancer</td>
</tr>
<tr>
<td>ESB</td>
<td>Enterprise Service Bus</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
</tr>
<tr>
<td>IMAP</td>
<td>Internet Message Access Protocol</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>J2EE (Java EE)</td>
<td>Java (2) Enterprise Edition</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Message Protocol</td>
</tr>
<tr>
<td>JSON</td>
<td>Javascript Object Notation</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>MOM</td>
<td>Message-oriented Middleware</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>POP</td>
<td>Post Office Protocol</td>
</tr>
<tr>
<td>ROI</td>
<td>Return on Investment</td>
</tr>
<tr>
<td>SAML2</td>
<td>Security Assertion Markup Language 2.0</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SSO</td>
<td>Single Sign On</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>STOMP</td>
<td>Simple Text Oriented Messaging Protocol</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XMPP</td>
<td>Extensible Messaging and Presence Protocol</td>
</tr>
</tbody>
</table>
Dedicated to my parents for their support
Chapter 1

Introduction

1.1 Overview

1.1.1 Projects’ frame

The company which I’m working for has been developing software for over 20 years. This software is oriented to professionals who need a specialized tool for big projects, where quality has an important role. This software improves the processes used in these projects and optimizes the way professionals work. The size and amount of projects has grown during the past years. In each of these projects there are dozens of professionals from the same sector working together and constantly sharing information. Since the usage of the applications provided has raised in the past years, the company has grown substantially and the software is currently developed by few hundreds of developers around the world. This, adding the fact that these products have been in the market for over 20 years, creates a large code base of over 7 million lines of code.

However, software evolves and as new technologies appear also new features can be implemented to make the customer’s life easier. Additionally new needs pop up as the projects where the software is used get bigger and bigger. It is then when collaborative work appears and starts to have an important role.

1.1.2 Collaborative tools and their role

Projects that involve many people working on the same data have been traditionally a headache for workers and managers. A large amount of manual work is needed in order to merge all the changes into the same place. This manual work has been causing huge delays and inefficiency in many projects and it is a main concern for companies.
幸运的是，随着云计算技术和协作工具的发展，项目中使用这些工具并优化时间花费的项目大幅增加。使用我们公司软件的公司也需要使用这些协作工具来优化项目中的时间花费。

1.1.3 当前状况

目前，我所在的公司为客户提供协作工具，以满足客户的需求。这些协作工具共享自己的数据模型，可以同时被多个用户使用和更新。然而，这些工具尚未完全完成，因为需要克服许多挑战。这些工具被托管在不同的云PaaS上，这些PaaS提供了适合每个应用的服务。一些因素，如员工在不同平台上的经验、提供的功能的适用性和价格，决定了使用这些PaaS的决定。

然后，另一家公司进入了这一领域。我所在公司的公司所有者是一家大型跨国公司，最近为他们的所有子公司开发了一个定制的PaaS，称为CPaaS。这个新的PaaS使用了许多提供商，并在它们之上建立了一层，根据任务和提供的服务选择了最合适的。这意味着更好的可用性，不受任何公司的影响，并具有更好的价格。更好的可用性意味着如果其中一个提供者的服务中断，它会使用可用的提供者。它也是非依赖性的，因为它没有绑定到任何特定的平台，所以可以在广泛的IaaS提供商中使用。如果某些提供商关闭其服务，还有其他提供商可以提供这些服务。这也意味着有来自不同提供商的不同价格，并且可以选择最便宜的。由于这个项目很大，这些提供商也提供了折扣。所有这些因素使CPaaS成为替换当前实施的合适候选。

1.2 研究问题

我所在的公司正在考虑使用自定义的PaaS来实施服务，但这是可行的吗？可以获得什么，有哪些优势，可能遇到的问题以及主要威胁是什么？采用这个新平台可能意味着需要对当前在封闭平台上执行的服务器代码重写。基本上，问题是要找出该解决方案是否符合公司的目的以及如何实施。这是一个复杂的工作，因为当前的代码使用了许多不同的技术，这些技术可能与现有的技术不同。
platform-specific and should guarantee exactly the same behaviour if another technology is used.

1.2.1 Reasons to change to another platform

There are many reasons to change to another platform. Summing up, and as said previously, it is much better to use platform-independent components rather than platform-specific components because it does not tie the company to pay to any specific provider for their services. If the solution is platform-independent it could be even ran in local machines hosted by the same company. Otherwise, hosting all the services in a closed platform could cost a lot of money for the company if the provider decides to increase the prices or even shut down the service.

There are other reasons as well. This customized PaaS is shared also with other venture companies that belong to the same multinational owner, so this means that applications of general purpose like authentication, authorization, file storage, structured and unstructured data services and others could be used across all the companies.

1.3 The research question

The research is mainly about analyzing the customized PaaS, comparing it to the currently used and redesigning the current projects in order to fit them into this customized PaaS. For this, the first step is to take a deep look into the current implementation and what are the requirements. Once the analysis is done, we can analyze also the proposed solution and see which are the advantages and drawbacks and compare them. After that, if the proposed solution meets the requirements we can proceed and redesign the current architecture.

So here we are looking for a concrete question: Is it possible to integrate our current cloud solutions into a customized PaaS and how? In other words: is this customized PaaS a good alternative to our cloud solutions? If it is so, what can we do to integrate these cloud solutions into the customized PaaS?

So in this research there will be mainly two clearly different questions to answer. I’m going to refer to them as Q1 and Q2 from now on.

Q1: Is it feasible to integrate our cloud solutions into CPaaS?

In order to answer Q1 there should be a deep analysis of the current services, the technologies used and the requirements of the projects. It needs a good understanding
of the software, the architecture and the protocols and technologies used. Then we also need to know what are the features that CPaaS provides and see if they match the requirements of the projects. Once we know all this information, we can determine if it is possible to fit our cloud solutions into CPaaS.

**Q2: How can these solutions be integrated into CPaaS?**

After knowing if it is possible the integration, Q2 aims to solve the way this could be implemented. What changes in the current architecture do we need? In order to solve this question we should deeply analyze the software architecture, redesign it and think about the proper way to implement all the cloud solutions not only as a standalone projects but also as a part of the application ecosystem with other projects.

### 1.4 Research process

In order to solve these two questions we are going to use a research methodology consisting in few steps that we are going to follow. These steps will allow us to go through a detailed process that will enable us to eventually find out a result and extract the conclusions from it.

#### 1.4.1 Task list

Now let’s describe the steps that we are going to follow in order to understand the process. This will be very helpful to get the methodology used for this research.

1. **Gather information:** First there is going to be an investigation about the software underneath the proposed platform, in order to understand its functionality and what the different middleware pieces do.

2. **Documentation:** After that, there is going to be a background theory explaining the different components and middleware used. This is one of the most important steps because it is very important to have a very good understanding of the different components, what do they do, and how do they work. If this step is vague and the information is not clear, it can lead to a chain of misunderstandings.

3. **Analysis:** After knowing the theory, we are going to analyze the case study in our company. Here we will explain what is the concrete case in the company, what are the needs and the solution proposed, the CPaaS. To do that the best way is gathering the requirements that will give us a good understanding of how the system works, and then see if it matches what’s offered in CPaaS.
4. **Design and prototyping:** Once analyzed the viability of the project, there’s going to be an architectural redesign of how it should be done along with a proof of concept. This step will require some practical and a hands-on prototype that would serve as a proof of concept. If the project is not viable, there is going to be some reasons why it is not possible and an explanation of what would be needed to make it viable.

5. **Conclusions:** Eventually, there will be results of the redesign and the conclusions that we can take after it. Here we will look at the results, extract information of them and also provide some future work that can be done.

### 1.5 Thesis planning

Here there is a first picture with a reasonable timing and the different estimations of the different tasks that there will be and an approximate duration of them. There are also some milestones that will serve as a checkpoints.

The fist step would be to get all the information available, have a really good understanding about what the thesis is about, how all the technology underneath works and how the proof of concept is going to be done. These first tasks could take up to 2 weeks because we need to gather information from different sources and there is a lot of documentation to read about.

After that, there will be an analysis of the information. This means processing the information and providing some results about the viability of porting our cloud solutions into CPaaS. It means also setting up some of the middleware used in CPaaS to see how it actually works. We foresee this to last for around a month because we don’t know how complex will be to analyze this and to setup CPaaS components locally. This will also clarify some doubts that might remain from the information retrieval process.

Once this is done, there will be a redesign of the architecture that it’s supposed to last over a week and will change the overall picture of the current software architecture. If the information previously fetched is clear enough, this process should be pretty straightforward.

After the redesign process, we will dedicate a bit less of two months to implement a proof of concept that demonstrates that the redesign makes sense and it is possible to use CPaaS as a platform to deploy our current cloud solutions.

In parallel to all these processes, this document should be written. For this purpose we will schedule some checkpoints when the different chapters should be finished. This will
make us easily switch from a task to another, prevent spending too many days doing
the same thing, avoid monotony and provide flexibility.
<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
<th>Predecessors</th>
<th>Milestone Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Analysis</td>
<td>0 days</td>
<td>Fri 21/3/14</td>
<td>Fri 22/3/14</td>
<td>Magdysa Xavi</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Redesign</td>
<td>0 days</td>
<td>Sun 3/4/14</td>
<td>Sun 5/4/14</td>
<td>Magdysa Xavi</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Results</td>
<td>0 days</td>
<td>Sun 20/4/14</td>
<td>Sun 20/4/14</td>
<td>Magdysa Xavi</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Conclude</td>
<td>0 days</td>
<td>Sun 4/5/14</td>
<td>Sun 4/5/14</td>
<td>Magdysa Xavi</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Finish up</td>
<td>0 days</td>
<td>Fri 23/5/14</td>
<td>Fri 23/5/14</td>
<td>Magdysa Xavi</td>
<td></td>
</tr>
</tbody>
</table>


1.6 Structure of the thesis

This thesis involves a lot of analysis and software design. The thesis starts with an introduction to the problem to understand why it is important to research about this cloud platform. It is also important to ask the correct questions to have a clear scope of the thesis, what should be resolved and what is out of the scope of the thesis.

After the introduction, some background theory is needed in order to understand the topic and get a basic knowledge of the technology underneath the cloud platforms. We provide also information about what SOA is, what is it meant for and what are its properties. There is also a description of what MOM is, how it works and an explanation of some different standards used.

After the basic information needed to understand the concepts, we will focus on the case study: the cloud software of the company. In this chapter we will describe how the current implementation of the cloud solutions work digging deep into the details of the key parts of the project, the ones that are platform-specific and are the most problematic. In this chapter there won’t be any explanation about what the software does internally and there won’t be any change in that part since the core of the software should remain the same to preserve the functionality.

In the following chapter there will be an explanation of how the CPaaS works, what are the features and the restrictions. As this platform is still being implemented, we should consider also features that are not currently available but will be in a near future. This chapter will mainly explain not only the features but the overall architecture and the properties of why this can be a good platform to use.

Once the analysis is done and there is a clear understanding of the situation of the project and the solution offered, there will be a redesign of the architecture to fit the current software, along with the needed modifications to support the CPaaS, into the new platform. This redesign has to take in count many things like the integration of future projects, communication between applications and re-utilization of some of the features by other applications.

Finally, some conclusions are given to answer to Q1 and Q2. These conclusions include a summary of the results and the reasoning after knowing them.
Chapter 2

Background theory

2.1 Cloud computing

Cloud computing[1][2][3] has revolutionized many things along the way. Previously, many companies used to have a big server room full of machines that used to become obsolete quite soon. Also the amount of servers needed was never the optimal, whether it was because there were too few or too many servers. It also needed maintenance and support, which means having people working just on maintaining the servers.

Cloud computing solves this by enabling these companies subcontract these servers under usage request, lowering the costs of having their own server infrastructure. Some of the most famous cloud computing examples are Amazon EC2 (IaaS)[4], Heroku (PaaS)[5] and Google AppEngine (PaaS)[6], used by large companies to run their cloud services.

Moreover, cloud computing has enabled lots of new business to emerge. All these technological companies previously needed a huge investment in order to have the necessary infrastructure but now the amount of resources needed to start a technological company have dwindled. The initial investment in servers of a new company is far from the prices needed for buying all the new machines, power and conditioning. This has unlocked hundreds of start-ups and has made lean methodologies cheap and easy to apply.

These cloud services are offered according to different models: Infrastructure as a Service, Platform as a Service and Software as a Service. IaaS is the most basic service providing only remote access to a virtual machine, PaaS typically adds also software like web servers and programming languages execution environments and databases. Some of the PaaS may include also auto-scaling and external data storage service. SaaS directly provides software running on the cloud to the end users. SaaS aims to provide a specific software for a concrete task, and it is bounded to the API of the service. This is why SaaS is not needed in this project. Since we want to provide SaaS we need to use either
PaaS or IaaS with the needed software on top, to build our applications on them and provide an API that can be used by the clients.

2.1.1 Comparing IaaS and PaaS

There are many differences between IaaS and PaaS[7]. The main difference is that IaaS only provides access to a virtual machine and the software built on top is also managed by the user of the service. So the user has to take care of maintaining the middleware used to deploy the applications, the management of the database and other tasks. This is good in case there is a need to implement a certain software stack very particular, or to optimize costs since IaaS can be cheaper than a PaaS because the provided service is more basic. Another of the advantages of IaaS is that it can be the easy and fast way to deploy legacy software into some premises while the PaaS is not available or the software is being prepared to take more advantage of PaaS features. This might be useful as a transitory step towards the implementation in a PaaS and to preserve old features of legacy software.

On the other hand, PaaS offers a comfortable way to deploy applications and manage them. PaaS might also add some extra layers providing general authentication and authorization systems, messaging and other services across all the applications deployed. This is very convenient for general purpose applications or users that have to deploy many cloud services and want them to be integrated into the same architecture. It is also easier to take new servers online compared to IaaS because there is no need to setup all the software stack. Also salary expenses of system administrators alone usually already exceed the cost of using PaaS, so it is worth to use it in most of the cases.

In Figure 2.1 we can see the differences between the different models offered. The less things that are under your control, the more dependent on the platform you are but also the more easy it gets to deploy applications.
Service-oriented architecture (SOA) is a software architecture pattern that takes different pieces of software to provide functionality as services to other applications. Different services can be combined to provide one software application, and some of these services can be reused with another one to provide a different application. SOA is based on services loosely coupled and the aim is to have a business-oriented platform that allows users to combine pieces of functionality to create ad-hoc applications reusing existing software. There are some principles documented to understand better what SOA is:

1. **Standardized service contract**: Services are in compliance with the collective service-description documents.

2. **Service loose coupling**: Dependencies within services are minimized and decoupled from their surrounding environment.

3. **Service abstraction**: Service contracts only contain essential information about the services, but services’ logic goes beyond the contract.

4. **Service reusability**: Services are reusable by other applications across the enterprise.

5. **Service autonomy**: Services exercise control over the logic underneath them.
6. **Service stateless**: Services minimize resource consumption by deferring the management of state information when necessary.

7. **Service discoverability**: Services are supplemented with meta data by which they can be interpreted.

8. **Service composability**: Services are parts of compounded participants, regardless of their size and complexity.

There are lots of benefits in using service-orientation. It increases interoperability, because all services are designed to be compatible so that they can be assembled and reconfigured. It also increases federation between the services, so all they have a common contract but can be managed independently. This common contract is tied to a business-centric model that allows the whole platform to be aligned with the business and change together. Also the ROI increases because the services are reused and their repeated value surpasses the cost of development and ownership. It also allows organizations increase their agility. Business requirements can be fulfilled quicker and with less effort by reusing the services, which are interoperable between them.

Now we are going to describe some of the most common components provided by a SOA-based platform. These services are not mandatory for any solution, but they are common basic components that provide useful functionality to the system and give a basic understanding of what kind of components are commonly needed. These components are provided by CPaaS and are the ones that we are going to use.

### 2.2.1 API Manager

Usually a company has many products and services deployed into the same system. These products and services can interact with a client using an API, according to SOA principles. But when a system has many services their APIs need to be organized and structured to guarantee a clear understanding of the whole system and its components. Here is where the API Manager takes action, routing and managing the different APIs deployed into the server in order to make them more understandable for the end user. The API Manager also provides versioning of the different APIs exposed, which is very valuable for the administrator to organize and monitor the different versions of the APIs.

The API Manager has an added feature, and it is to grant access to the different users into the applications. So it also monitors authorization and policies to verify that the user has access to an application. The API Manager doesn’t itself implement any authentication methods, but it can call another decoupled services taking care of authentication to identify the users.
The API Manager can also monitor traffic and identify problems that the users might have. This is very important to measure with quantifiable metrics the behaviour of the users and understand their needs. These metrics can be also analyzed by other components specifically dedicated to that purpose, such as Business Monitors.

### 2.2.2 Identity Server

Applications are managed by different types of users under their identity. Each of the users can have a different role in the system, thus having different levels of privacy for each of the groups. An Identity Server provides security and identity management of the applications provided by the system.

Identity Servers enables the development of different authentication methods and combine them and offer a better user experience, guaranteeing access to all the users across all the applications. This reduces the single sign-on environment and simplifies the interaction of the user with the system.

The Identity Server can be also integrated with other existing authentication and authorization systems, thus enabling companies to reuse existing systems and expanding the capabilities of the system without loosing the legacy systems.

### 2.2.3 Enterprise Service Bus

Enterprise Service Bus is a software architecture model used for the communication between applications interacting with each other in service-oriented architecture. It promotes agility and flexibility regarding communications and interaction between applications. ESB is in essence a middleware that simplifies the interaction between applications and builds a common communication channel between applications acting at the same time as a translator between different protocols. It supports message queuing with protocols like JMS[11] and AMPQ, but also HTTP(S), FTP(S), POP, IMAP or SMTP among others[12][13].

The ESB controls the communications between the different services and provides a common interface for all of them. It declares the different interfaces of the different services in a document and provides access to them through a common protocol. So all service consumers can communicate these services through a common interface and the ESB translates these requests to the corresponding protocol to interact with the service.
This kind of architecture is not the most efficient, but enables SOA to provide reusable services for different applications by establishing a common communications bus. The fact that this middleware has to translate the requests to the corresponding protocol adds some overhead to the response time. This overhead is permitted by the companies using SOA because the aim of these services is not providing a lightning fast service, but to have all their services aligned and operative across them even if it has to compromise a bit of efficiency.

2.2.3.1 ESB Internals

The ESB acts as a lightweight component that lets you connect different applications. This internally works in the following way. A request arrives to the ESB and it is liked to one of the ESB components. Each of these components is a list of small actions chained between them act as a processor of the message. The message goes through these small actions and usually reaches an endpoint at the end of the chain. These small actions are called mediators and the chain of mediators is called mediator sequence or formally called sequence.

• Mediators
A mediator is a processing unit in the ESB. A mediator has access to all the parts of the ESB and the message. It is able to transform the message. Mediators can be customized and contain other mediators. There are two kinds of mediator:

- **Node mediators**: contains child mediators.
- **Leaf mediators**: does not contain any child mediator.

Some examples of mediators are: sending a message to an endpoint, drop a message, get a property from the message’s URL or log the output of the message. The power of these mediators comes when you set a chain of them, converting it into a mediator sequence.

### Sequences

A mediator sequence or just sequence is a list of mediators executed in order. Messages are sent along all the mediators.

![Sequence of mediators](image.png)

**Figure 2.3:** Sequence of mediators.

Sequences can send messages to endpoints as a last step in order to forward the message to a specific URI. This is what makes a ESB a proxy too, being a middleware between the exposed service and the actual application’s endpoint. Sequences can call other sequences also, as we can see in the following example.

```xml
<sequence name="foo">
  <log/>
  <property name="test" value="test value"/>
  <sequence key="other_sequence"/>
  <send/>
</sequence>
```
Endpoints
Endpoints are external destinations for messages leaving the ESB. There are different kinds of endpoints depending on the type of the destination: address endpoints, WSDL endpoints, load balancing endpoints, fail-over endpoints and HTTP endpoints.

- **Address endpoints**: defines an endpoint with a destination URI.
- **WSDL endpoints**: endpoint definition based on a WSDL document.
- **Load balancing endpoints**: defines a set of endpoints where to send the messages in a round robin fashion.
- **Fail-over endpoint**: defines a set of endpoints where to send messages in the following way. The messages is sent to the first endpoint, and if it fails it will be sent to the second one, and if it fails to the third of the list and so on. The ESB switches to the primary endpoint when it becomes available again.
- **HTTP endpoints**: allows to define REST endpoints using URI templates. These templates use variables to form the destination URI.

### 2.2.4 Application Server
An application server[14] is a software framework that provides an environment where to run applications, independently of what the applications do. The aim of the application server is to execute efficiently procedures for supporting its applications. Many application servers also support other features to allow developers to focus on implementing the business logic. Some of these features are clustering, fail-over or load balancing.

There are many advantages of applications servers. Here there are some:

- **Data and code integrity**: the business logic is centralized, so it is easy to upgrade and update without risks.
- **Centralized configuration**: changes to the configuration can be done in a centralized place.
- **Security**: application servers provide security layers. This prevents security issues by developing data access mechanisms away from the client side and without exposing the database layer.
- **Performance**: the usage of the client-server model enhances the performance of large applications with a big workload.
• Total Cost of Ownership: in combination, the previous benefits result into savings for the companies developing enterprise applications.

• Transaction support: the business logic behind the applications can let them perform atomic operations.

Application servers are widely used in all kinds of web architectures. There are lots of examples of application servers in different languages, like GlassFish (Java EE), Node.js (Javascript), Tomcat (Java EE), Zend Server (PHP), Microsoft Application Server (mainly C#) and many others.

2.3 Message-oriented Middleware

Message-oriented middleware[15] (MOM) is a key part of the SOA. MOM enables different pieces of software to be communicated by sending and receiving messages asynchronously along a distributed system[16]. MOM allows applications to use the same protocol to communicate across different heterogeneous platforms, reducing the complexity of building different applications that should communicate between them.

MOM have different advantages or reasons why it should be used[17]:

• Asynchronicity: unlike request-response communications, MOM uses an asynchronous system using message queues to store temporally the data. This message broker acting between the message consumers is in charge of all the logistics. It also offers persistent storage and it is fault-tolerant. Using the message broker as an intermediate controller means that the consumers don’t need to establish a connection between them.

• Routing: The message broker is in charge of gathering all the messages and distributing them among the different consumers. So MOM is good for routing messages and taking care of who has received what. Depending on the implementation, routing information can be on the client information, on the messaging layer or both.

• Transformation: MOM systems can have a built-in message transformation system interfering in the communication, intercepting the message and casting it to the accorded format in order to provide a common format. Many modern MOM systems provide mapping tools to determine the different rules that should be applied.

However MOM systems provide lots of advantages, they also have some disadvantages that should be taken in account when designing a message-oriented system:
• Extra component: having an extra component in the system means making the software less efficient because it adds extra computation. This might act as a bottleneck in highly scalable applications.

• Synchronicity: there might be applications that are near real-time and need a synchronous communication because they are waiting for the response of another application to continue. MOM systems provide asynchronous communication, and this slows down all this process because the message is not delivered right away. However, most MOM systems implement facilities to solve this problem, like grouping request and response in a similar way as a synchronous communication works.

• Standards: the lack of a standard API is one of the main problems of MOM systems. There are some implementations, like AMQP, JMS, DDS, XMPP or STOMP. The most common standards are AMQP, JMS and XMPP because they are the ones implemented by the most used MOM systems. However, not all the MOM systems implement all of them, and some of them have different implementations of the same protocol and that makes them not interoperable. Each of the major vendors have its own API, implementation and management tools.

Messaging has many roles in different areas such as Internet of Things, SOA systems or messaging platforms. However, some of the MOM implementations are more focused on SOA systems than other ones rather used for chat messaging applications or IoT, for example.

The role of MOM is specially remarkable in SOA systems. MOM provides a bus to exchange asynchronous information between services, thus making them loosely coupled to provide business processes. In fact, the most fundamental part of a SOA system is the communication mechanism that lets the services communicate between them. This could not be achieved without a messaging bus for asynchronous information like MOM provides. Lots of companies have solved the problem of intercommunication of their services by using MOM in their SOA systems.

There are different APIs implemented for MOM, such as JMS, AMQP, MSMQ, DDS or XMPP. We are going to take a look to the most relevant ones for our solutions.

2.3.1 Java Message Service

The Java Message Service API is a MOM API for sending messages between different clients. JMS is part of the Java EE platform and it allows applications to create, send, receive and read messages. JMS has been a robust and mature specification for many years. It also allows interoperability between languages running on top of JVM like Groovy or Scala. With JMS you can replace any JMS message broker with any other without major changes in the configuration and with no changes in the source code.
JMS defines a standard that can be used between Java Platform applications but it can’t be used with other applications using other languages. For example, a Java EE application can send a message to another Java EE application through a message broker implementing JMS. However, the compatibility problem is solved by most of the vendors by using JMS integrators that translate JMS messages into other standards like AMQP. However this translation is not very optimal nor efficient, it can be used for legacy Java EE applications using JMS if they need to be integrated with other messaging applications using other languages. In Figure 2.4 we can see how some message brokers like ActiveMQ support different APIs like JMS and STOMP and allows them to communicate.

2.3.1.1 JMS Properties

In JMS, the way messages are sent from the producer to the consumer is very simple. The messages are exchanged between producers and consumers connected to the same queue or topic. The consumer will only receive messages from the queue matching the name of the queue where it is connected. Later we will explain what queues and topics are.

JMS provides five type models for different kinds of data: Object, Map, Text, Bytes and Stream. Depending on the kind of data that is sent, one of this type models has to
be selected. \textit{TextMessage} and \textit{ByteMessage} are preferred for better compatibility with other standards.

Additionally, JMS provides a filter called message selector that allows the receiver to filter only the messages matching the parameters selected. This means that, for example, in a topic there might be few subscribers, a provider can send a message with a particular header property (for example some country code) and if some of the subscribers have a message selector with the same property (the same country code), it will be sent to the subscriber. However if another subscriber has a message selector with a different property (another country code), it won’t be delivered to this subscriber.

Messages in JMS are divided into three different sections. The first one is the header section that contains JMS header properties like the message ID, the time stamp or the length. The second one is the properties section that contains a set of key-value pairs with application specific properties. The third and last one is the message body section, that contains the actual data of the message.

\section*{2.3.1.2 Distribution models}

JMS can distribute messages following two kinds of models: queues and topics. Queues are FIFO message queues in the message broker where the producer sends a message and the first consumer that asks for it gets the message. Once the message is received in the consumer, it is erased from the message broker. However, persistence of the messages depend on the implementation of the message broker and the configuration of the queue. This is good for point-to-point communications between two applications for example, or also for load balancing data between the consumers.

The second model are topics. Topics have two kinds of roles, the publishers and the subscribers. The publishers are the ones that are sending the messages to the topic, and the subscribers are the ones that retrieve the data on the other end of the pipe. All the data sent to the topic can be seen by all the subscribers of that topic. If a publisher sends a message to the topic, the message broker is in charge of forwarding a copy of the message to all the subscribers of that topic. Once all the subscribers have received the message, the message is erased from the message broker. This again can be disabled depending on the message broker and its configuration. This functionality is similar to a chat room service.
One or many producers can send messages to the queues or topics, being distributed by
the message broker depending on the distribution model. Consumers can use message
selectors to filter these messages and receive only the ones addressed to them.

2.3.2 Advanced Message Queuing Protocol

Advanced Message Queuing Protocol[18], from now on AMQP, is an open standard
application layer protocol for MOM. AMQP defines message orientation, queuing, rout-
ing, reliability and security. AMQP was made to solve the problem of interoperability
between platforms by creating a standard for the structure and transmission of mes-
sages. Using AMQP servers, we should not worry about availability and reliability from
applications anymore. Now they are simpler, more functional and cheaper.

AMQP does not provide any specification for a standard API. It provides a specification
standard wire binary protocol to determine how the message structure should be and how
it should be sent. The good part of this is that AMQP implementations are interoperable
and each vendor provides a concrete API that fits in a concrete system or programming
language[19][20][21]. Also AMQP message brokers are interoperable and completely
agnostic to the client used.

Lots of SOA systems use AMQP as their messaging bus to communicate their services.
This is due to many reasons. First of all, AMQP provides compatibility between different
programming languages that might be used along their applications. Most of the AMQP
message brokers also implement JMS compatibility, so also Java applications using JMS
can be integrated in the SOA system. Furthermore, AMPQ is an open standard and
its not tied to any provider or vendor. This makes it very attractive for companies that
don’t want to be tied to any company or to a certain implementation. Companies can even implement their own customized software from open source projects and strengthen the most important features like security or persistence.

2.3.2.1 AMQP Properties

The version of the protocol used for most of the vendors is the 0.9.1 [22], which is the one we are going to talk about. The structure of AMQP messages is quite similar to the JMS messages[23], but there are few differences on the usage of the different parts. These are the parts:

- **Header**: contains immutable application-defined properties.
- **Properties**: contains routing properties and metadata.
- **Body**: contains the message content.

However, the differences are so little that it is not a big issue to make compatible AMQP messages and JMS messages. A footer can be included along the message as a optional part. There is also a difference with JMS regarding the body content. While JMS has five kinds of body types, AMQP only has one kind of body type: a byte message. This byte body type can be converted to one of the five JMS body types using the specification provided by AMQP. Even though, it is better to use text or bytes body types when converting to JMS for better portability.

2.3.2.2 Distribution models

AMQP handles message routing in a different way than JMS does. AMPQ sends a message to an exchange along with a routing key. Exchanges are the delivery service of the messages, routing them depending on the ways of delivering the message[17]. There are some types of exchanges:

- **Direct exchange**: sends the message directly to a single queue matching the routing key. This is the equivalent to the point-to-point messaging model in JMS. The messages are bound to the queue that exactly matches the routing key. Another difference with JMS is that you can bind multiple direct queues to an exchange, meaning that different consumers can receive the same message.
• **Fanout exchange:** this exchange broadcasts the messages to all the queues bounded to this exchange. It is the equivalent of a publish and subscribe model in JMS without message selectors.

• **Topic exchange:** it copies and queues the message to all consumers interested in the message based on pattern match of the routing key. In JMS, the equivalent would be the publish and subscribe model with message selectors. The difference with direct exchange is that the routing key does not have to match completely, the routing key has to match a pattern.

For example, if the consumer is bound to "services.eu", if a message comes with a routing key "services.eu.finland" it would match the pattern and thus, it would
be sent to the consumer. On the other hand, if a message arrives with a routing key "services.usa.ca", it would not match the pattern and it would not be sent to the consumer.

- **Headers exchange:** it examines the headers and queries them against the predicates provided by the interested consumers. The matching results are sent to the corresponding queue. A headers exchange is used for routing multiple attributes that might be difficult to route using a routing key. This exchange ignores the routing key and evaluates the header attributes instead. Because information in headers can be more complex than the single string used as a routing key, it allows much more accurate queries.

- **System exchange:** this is a special kind of exchange that routes the message to an application service or a system service. In this exchange type, no queues are involved. JMS does not have any equivalent and its implementation is completely optional for AMQP message brokers.

Exchanges never store messages, but they retain the parameters that bind a client with its routing information. Bindings are arguments supplied to exchanges to enable routing of messages. Depending on the type of the exchange, more or less bindings are needed. For example, in direct exchange the sender is providing the binding that sends a message to a queue. However, in topic exchange the receiver provides the binding information that makes all messages to be routed to the correct consumers. The latter is called consumer-driven messaging because the producer does not know anything about the receiver, it is the receiver that decides if he wants to be bound to a queue.
Chapter 3

Scenario description

The company which I’m working for has been using cloud services for some years. However, the architecture of the cloud services is not very optimal, because different projects are deployed and managed independently. Complex applications with different utilities have been spread through different cloud providers, platforms and they all use different systems to implement a similar functionality. The aim is to use a way to optimize enterprise processes by grouping these cloud services under the same umbrella, maintaining them loosely coupled but also making them interoperable to reuse the resources.

In this chapter we are going to gather the requirements of this platform in order to preserve the main functionalities of the applications. This will be needed in order to know what we need from the CPaaS, what are the things that it provides that can be used and what are the components that CPaaS must have in order to make it fully functional with the current applications. After that there is going to be an explanation of the current architecture, what are the main key points of the different software pieces and how they affect to the architecture.

3.1 The requirements

The list of requirements can be very extensive and detailed. However, the aim of this chapter is not to dig deep into the details but to have an overview of the key requirements and the components needed.

• The platform MUST be able to deploy existing applications

Currently there are lots of applications deployed in different places and in a future more of them will be created. The new platform must be able to deploy these applications in order to preserve them. The applications are developed in few different programming languages, so the new platform must support all of them.
as well as new programming languages that might be required in the future. The deployment of these applications should be easy and preferably provide some kind of user interface.

- **The platform MUST be able to manage current authorization and permissions**

  All the application users, administrators and other roles must be controlled and the platform must have some way to give permissions or authorization to them. The access to the applications must be controlled and secure.

  Some authorization systems are already implemented and shared across different services within the company. This system must be supported in order to preserve it.

- **The platform MUST provide a relational database**

  Most of the cloud applications manage a lot of data every day. This data should be persistent in most of the cases. In order to store this data in a structured way and supporting legacy web applications the system should provide a relational database. Many different relational databases might be needed in order to support full functionality of the system because each one of them has different specific properties, limitations or just a different structure.

  In these systems, relational databases are mainly used for storing user information, metadata information, forums, wiki pages, help content and usage statistics. This kind of information is usually pretty well structured and not very flexible in terms of data structure. This is the type of information that should be stored in relational databases.

  The current paradigm regarding relational databases is quite complex since different applications use different products. But anyway, changing the database controllers and minor changes depending on the databases used should do the trick and applications should eventually be able to change from a database to another without painful changes.

- **The platform MUST provide a non-relational database**

  The same way that some applications use a relational database, there are also other ones that use non-relational databases. These databases must be supported by the platform. Usually this databases are needed to store more flexible data with not so well structured form. Different implementations of different non-relational databases are needed because they have different features that are needed for different reasons. Different databases might be needed in different situations in order to solve some technical problem, or just for supporting some implementation.

  Depending on the demands of the current applications there might be some requirements for these non-relational databases, like transaction support, that cannot be
achieved with any non-relational databases. In that case, some specific implementation must be supported in order to maintain the integrity of the application.

- **The platform MUST provide a messaging bus**

  Messaging is very important for the current architecture. It provides a fault-tolerant, persistent and reliable way to interchange messages between the client and the server. The current cloud solutions need a Message Broker in order to interact with the clients to provide an asynchronous communication.

  This component is mandatory because it provides the necessary features needed in order to make the clients and server be connected and communicated in an asynchronous way and providing reliability in the reception of the message.

- **The platform MUST provide a BLOB/file storage system**

  Collaboration tools among other applications need to share a lot of data. This data needs to be transferred into a scalable platform that allows to store and transfer big chunks of data without getting stuck. CPaaS must provide some kind of storage system that allows to do this.

  SOA is not very well prepared for this kinds of needs, but the IaaS provider should provide some vendor-specific solution to this that might be the best option.

### 3.2 Current architecture

The current architecture is quite complex because the different cloud services are spread all over different platforms and they have a different architecture in each of them. In some of these platforms there are some components that are key components for the applications deployed. These key components are not needed in many of the other applications, but they would need to be implemented in a common platform to preserve the functionality.

Because of privacy policies of the company there is not a lot of information available for external sources, so we will try to make an overall picture of the architecture without entering in the details. The cloud solutions are currently divided into two different platforms in different vendors for different purposes. The first one is for a very specific purpose, so it is built into a customized architecture that easily takes advantage of messaging in order to communicate with the clients. The second one is for more generic purposes. It is much bigger in terms of complexity and amount of data processed. It contains all kinds of web clients that are served to the different stakeholders as well as services to store some specific data about software features.

Even if the current cloud solutions are spread into different platforms it is better to take an overall picture of the components required by the applications to run in these
platforms. In this section we are going to analyze the different cloud components needed to deploy the applications and make them work and how are they structured currently.

### 3.2.1 Architectural components

The current cloud architecture used is more complex than the one shown here because of many different approaches currently being tested. However, the one shown here reflects the main components and provides a clear overall idea.

![Schema of the current architecture.](image)

The requests sent to the server are processed through an Elastic Load Balancer (ELB) that derives the request in a round-robin fashion to the deployment instances and adds new ones when needed. If the usage is lower than required, the ELB also can shut down an instance.

Each instance runs a web application that has access to the database. The application can access also the Message Broker, located in some other server and provided as a service by the vendor. Even that the Message Broker is not located in the same cluster, because it’s a generic Message Broker provided and maintained by the vendor, the access is fast and it does not affect negatively the performance of the applications.

There should be a special mention to the databases, which are replicated in the different instances used. These databases follow a master-slave pattern and they are non-relational databases. There is, however, a relational database. In the same way that the Message Broker is provided by the vendor, the database it is also provided by the vendor and it can be accessed from any application. About the replication, scalability and other matters the vendor takes care itself, providing a comfortable service with low
costs of maintainability. The non-relational databases had to be deployed in the server cluster because it is needed a certain version that provides features not available in the vendor’s solution.

This schema is quite common in simple applications or companies that have one cloud solution because it makes it highly scalable and efficient. However, for large companies that have multiple cloud solutions and provide different services to different stakeholders the complexity of the platform increases to provide more flexibility, maintainability and reusability. We will see an example of this in the CPaaS architecture.

### 3.3 CPaaS architecture

CPaaS is based in an open source middleware platform and deployed into a major IaaS provider. CPaaS takes benefit of the open source middleware to build a reliable enterprise platform that allows to implement a SOA. CPaaS has been built to support major features that can be applied to multiple solutions across the platform, such as security features, an identity provider for authentication, support for multiple databases, governance and monitoring and a communication bus between applications.

In this chapter we are not going to explain the theory of how the components should work because it has been already explained in the theory background, but we are going to focus more in how they interact in order to provide the desired functionality. We are also going to enter in the details of the specific software that CPaaS is using to provide a deep understanding of the platform and its peculiarities.

CPaaS uses a middleware composed by different components that interact between them. These components are different and completely independent. They are similar to Lego bricks that compound a whole thing, and you can decouple and remove them if you are not using them, or add more if you need some extra functionality.

#### 3.3.1 Components

The components used in CPaaS are basic functionalities extended and customized to be used along all kinds of services. They are implemented independently of each other and they provide different functionalities that can be combined and complemented.

In Figure 3.2 we have an overall view of the CPaaS architecture, with its different components and their linkages. In the drawing we can see two kinds of linkages, one that links components with points and the other one with arrows. The links that use points mean that from one of these components you can access to the other side, meaning that there is a communication between these components and there is an access point
from one side to the other. On the other hand, arrows mean that one component is using the other one to work.

So in this case, a Web Server and Application Server apps can use the API Manager since the API Manager exposes the APIs to the clients to use it. From the API Manager you can access to some services exposed in the ESB, you can access to the Message Broker or directly to the Application Server directly. It is a good practice that if you have to expose an API you do it through the ESB and not directly from the Application Server to the API Manager. This adds a bit of overhead created by the ESB but allows more maintainability and flexibility. From the API Manager you can access also to the Deployment Platform that shows a visual interface to deploy the web applications and services.

From the ESB point of view, you can use the Message Broker as an external JMS Message Broker instead of using the default one. This would provide a better separation of the ESB and the Message Broker itself and a boost in the performance since the Message Broker can be scaled if necessary independently of the ESB. The ESB can access to Web Services exposed locally by the applications or can interact with the Messages in the Message Broker, but can also access to remote endpoints.

Finally, from the Application Server you can access to the Message Broker and to the database, but you can also access to the services exposed in the ESB and the APIs exposed to the API Manager.
3.3.1.1 API Manager

The chosen API Manager provides a visual way to manage your APIs and expose them to the clients. Provides a web interface that allows the administrators to route the APIs, manage their versions, monitor their usage and get statistics. The idea is to provide a store like Google Play or Apple App Store where you can manage, upload, rate and see the statistics of an API.
The API Manager also deals with security and authentication. It is tied to the Identity Server, which is in charge of providing authentication. This communication allows the ESB more flexibility as it does not deal directly with authentication. Just selecting the authentication protocol Moreover, it provides to the user an easy to use way to select security settings.

### 3.3.1.2 Identity Server

The Identity Provider manages authentication and authorization of the users. It provides an interface to manage the authentication and authorization protocols used and manage the different consumers. The Identity Server used supports OpenID, SAML2, OAuth and Kerberos KDC among other protocols.

So for instance, you might have an application that needs OpenID to authenticate its users and you want to make it compatible with their Google ID. But you might want to have another application that uses SAML2 to authenticate their users. These users should have different rights, so they have to be authorized to use one or other application using OAuth. All these cases are managed by the Identity Server.

The Identity Provider has lots of options and lets the administrators have full control of the authentication and authorization process. From administration of Identity Providers of OpenID, management of applications that use OAuth, system for cross-domain identity management or SAML2 SSO.
3.3.1.3 Enterprise Serial Bus

The ESB is a key component in the future of the cloud services since it is the backbone of SOA. The ESB used allows many protocols, including the ones currently used by the current architecture. So in that sense it fits the purpose. However, as the current architecture is not using any ESB to communicate different services, it is a future feature to take in count.

The ESB selected also provides an interface to interact with it. However, there are lots of things that can be customized by changing the code directly in the source. The user interface also provides this possibility by prompting the code into the user interface using an embedded text editor and validating the XML code when submitted. This makes the customization of the ESB very comfortable to work with, providing an interface but also allowing the advanced users to work directly on the code. Event that on the first instance a user interface might seem the best option to configure the ESB, it is usually more handy to just type the corresponding XML code because the user interface usually provides lots of options that are useless for the basic usage and that makes its usage a bit confusing and difficult. And even if a user wants to use an advanced feature, it is more likely to be done directly modifying the XML code.

The ESB used allows to use all of the major protocols used for web services and transporting data. For example JSON, SOAP, HTTP, HTTPS, POP, IMAP, JMS, AMQP, TCP among many others. The important point is that all of the used protocols are supported by the ESB so that they could be integrated into it if necessary, modularizing the application and integrating it better into the service-oriented architecture.

3.3.1.4 Message Broker

The Message Broker is another key component in the architecture. It allows asynchronous communication between server-side applications and clients. It is a key component because it ensures reliability and persistence of the messages exchanged between the producers and the consumers. In our case, the Message Broker selected supports the protocols currently used by the applications.

Apart from supporting the current protocols used it is also very important that the Message Broker selected is highly scalable, fault-tolerant and allows persistence. The messages received by the Message Broker should be stored persistently (not only in volatile memory) until they are delivered to the receiver or receivers. This way we make sure no messages are lost on the way and there is no information loss. This might be critical for some SOA systems using a Message Broker such as banks’, but it is also important in construction field since losing partial information would mean that some building would lack some part of the building which would lead to catastrophic
consequences. This is why the Message Broker is so important and has a critical role in the architecture.

### 3.3.1.5 Application Deployment Service

There will be also an Application Deployment Service component in charge of the deployment of the applications and services. This component will be able to deploy different applications in different programming languages and using different frameworks. As this component will be customized in order to support different requested programming languages and frameworks, the company in charge of the development of this deployment platform should be aware of the different technologies used.

The administrator will be able to manage and deploy different applications through a user interface. This Application Deployment Service will provide an interface so that the user can upload a compressed file with the web application inside, mentioning the framework/language used and the platform will deploy it in a virtual machine.

### 3.3.2 Scalability

CPaaS architecture is also very scalable. All the components are controlled by an Elastic Load Balancer that allows to redirect the requests to the corresponding instance. The balance is done in a round-robin fashion, routing the request to the instance with less workload.

This ELB also takes care of automatically increase the number of instances used of every component when the workload of the instances overpasses a given CPU workload limit. It happens the same the other way around, then there are instances that are not necessary, the ELB will automatically shut down them.

In Figure 3.2 we could allocate the ELB along the different components because it takes care of all of them and acts across all the platform.
Chapter 4

Analysis of the integration

4.1 Analysis of the integration with the customized PaaS

Once we know what is the current architecture, what we need from a new PaaS and the architecture of the CPaaS let’s compare them and analyze the viability of moving the cloud services into CPaaS. This information will serve to solve Q1 in the next chapter.

First we are going to find out the common points between the two platforms, so that we can compare them not taking in count the needs of the software. These platforms are obviously different and provide different features. We are going to see also which architecture is likely to fit better in the future plans of the company and its strategy.

Then we are going to find out potential difficulties that can emerge when moving the cloud services into CPaaS. This is an important part because the company should be aware of the main threats and possible difficulties and problems that they might have to face if they move their cloud services into CPaaS.

Once all these options have been analyzed, there will be a comparison table to provide clear and structured information about both platforms and their pros and cons. This visual approach makes decision-taking an easier task and will provide very valuable information in a readable way.

Finally, there is going to be a proof of concept. That means that there will be some prototype for proving that the platform works and satisfies the needs of the software. This prototype will be used internally inside the company, so it won’t be public nor available because it is out of the scope of this thesis. There will be just a brief explanation of what has been done and how, without showing the actual result of the prototype. However, this prototype will be presented to the company and will provide a very valuable feedback about CPaaS.
4.1.1 Common points between the platforms

The different platforms analyzed have some differences because they are designed for slightly different purposes. However, they have also lots of common features and in most of the cases they are used in most of the applications. Now let’s go a bit deeper into which are the common points between them.

• **Deployment support**

  The first of these common points is that in all the platforms it is possible to deploy the current applications. This is a mandatory point because rewriting the whole application would imply a big and unnecessary investment. It would be needed new people with skills to develop those applications, experts in the technology used that the company currently doesn’t have. It would be a big waste of resources plus it would delay the whole process.

• **Messaging support**

  The second common point is that in all the platforms there is a Message Broker that acts as a intermediate point for messages and allows asynchronous communication. This is also very important since it allows to maintain the same system used by applications to communicate with the clients asynchronously.

• **Database support**

  They all allow to store data into databases too. Relational and non-relational databases are supported by the platforms. As it was said previously, there might be some changes to adapt from one database to another. There might be some drivers that need to be changed or some parameters that will need to be tuned but in the end they all work similar.

• **Monitoring support**

  They all provide monitoring support for supervising the APIs exposed, the resource consumption and other important measures like bandwidth consumed, CPU usage or memory usage. This is very important since it lets the administrators know measurable metrics about the cloud usage and performance, as well as statistics about the visits.

4.1.2 Potential difficulties and differences

Although we have seen many common points between the platforms there are also some differences. It is very important also to detect possible difficulties that can arise. In the case some unpredicted problems appear it could be determinant for the company and
there would be a lot of effort wasted. Thus, it is very important to detect the difficulties and risks that porting the cloud services to CPaaS implies.

In the next sections we can see different problems that can appear. Some of them will appear for sure as a part of the transition to CPaaS, some of them might appear or not but we should be prepared for it, and some can be avoided using a different platform for the purpose.

4.1.2.1 Databases slightly different

Applications using relational and non-relational databases might need to modify the drivers used in order to support some other database. There might be some applications that are already using the correct ones, but some of them will need to be changed for sure. Also, the database structure in relational databases will need to be adapted if there are some differences, as in different implementations the size of some fields differ and need to be adapted to some other naming that fits the needed size of the field.

Also in non-relational databases, as they don’t use a common specific language like SQL in relational databases, there might be some changes in the drivers and libraries used to communicate with the database. In this case there are two options. The first one is to change the drivers or libraries to adapt it and take benefit of the whole platform. This would improve the response time because the database will be allocated in the same server cluster and the communication between cluster nodes is very fast. Moreover, it’s also a security principle to hide the database behind the business logic and not expose it to external sources. The second option is connect to the old database hosted somewhere else and continue using the same libraries. This second option can be used as a transitional step, but should be avoided in the long term.

4.1.2.2 Different BLOB storage

The platform proposed has its own BLOB storage system, and it should be used to store files and large amount of data. This BLOB storage system is more convenient than the current one because of two reasons. The first one is that it is located within the same server cluster, so it provides better availability and speed. The second one is that prices for this platform are cheaper than for the currently used. So this would mean that the clients sending or receiving BLOBs should change their libraries in order to send them into this platform.

This step can be done in different ways. One of them would be providing a service within the ESB and also exposed to external sources that would take care of managing these BLOBs. This would take benefit of having a SOA implemented on the server side.
4.1.2.3 Usage of AMQP instead of the current protocol

This is another drawback of changing into CPaaS. Currently the system is using a closed protocol property of a famous company. This makes it available only for certain platforms, excluding CPaaS. AMQP instead, is an open protocol and open source implementations can be found all along the web. If a new Message Broker is used, the applications using it should use AMQP or JMS mainly to interact with it. This would require few changes and new implementations using AMQP libraries. However, this should not be a big problem because most of the AMQP clients’ libraries are very well documented and there is plenty of information about them.

This difficulty can also be paled by keeping the old Message Broker and communicating with it even if it is located in another server. Again this solution might be good as a quick fix for the problem, but ideally it should use the Message Broker inside the same platform.

4.1.3 Comparison table between the options

In this section we are going to compare the current platforms used for the cloud solutions and the studied platform, called CPaaS, in form of table. This way makes easier to spot out the differences and provide a clearer comparison.
Table 4.1: Comparison table between the current platforms and CPaaS.

<table>
<thead>
<tr>
<th></th>
<th>Current platforms</th>
<th>CPaaS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application deployment</strong></td>
<td>Allows the deployment of the current applications.</td>
<td>Allows the deployment of the current applications.</td>
</tr>
<tr>
<td><strong>Messaging</strong></td>
<td>Uses a closed protocol of a proprietary.</td>
<td>Uses AMQP and JMS, allowing open source implementations and flexibility.</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>Good performance. Platforms oriented to a unique purpose generally.</td>
<td>Slightly slower performance because of the general usage and the middleware used.</td>
</tr>
<tr>
<td><strong>BLOB storage</strong></td>
<td>BLOB storage provided by the platform. Vendor specific service.</td>
<td>BLOB storage provided by the platform. Vendor specific service.</td>
</tr>
<tr>
<td><strong>Enterprise Service Bus</strong></td>
<td>No ESB in any of the platforms.</td>
<td>ESB provided by the platform allowing interoperability of the services.</td>
</tr>
<tr>
<td><strong>Modularity</strong></td>
<td>No modularity. Cloud services spread through different platforms using different resources.</td>
<td>Allows modularity using the ESB.</td>
</tr>
<tr>
<td><strong>Databases</strong></td>
<td>Different databases allowed. Also using vendor-specific databases.</td>
<td>Different databases allowed. No vendor-specific databases planned.</td>
</tr>
</tbody>
</table>

It is visible that CPaaS provides some long-term benefits, regarding modularity and reuse of the resources. However, some performance might dwindle on the transition theoretically, due to the overhead introduced by the middleware.

### 4.1.4 Proof of concept

To demonstrate that CPaaS works and it’s a suitable option as a platform for our cloud solutions, a proof of concept in form of prototype will be implemented for internal
purposes. Unfortunately, this prototype won’t be seen by external sources outside the company because of strict privacy policies. However, a brief explanation of the proof and some schemes will be presented in order to understand how the proof of concept was made and how SOA has an important role in CPaaS.

The aim of the prototype is to send and receive files synchronized on the server side. So one client can share a file and other client should see that the file has been uploaded and then fetch it. All this, planned to use different services in charge of providing different things and all managed by a web application. With this approach we prove how SOA works and how it can be used. It’s a good example that uses the capabilities of this architecture and at the same time proves the good behaviour of the different components.

4.1.4.1 Use cases

There are mainly three use cases in this prototype, as we can see in Figure 4.1. In the three use cases we have one actor that interacts with the server. The first use case is when a client wants to upload a file. This process has to be synchronous because if two clients send requests to upload the same file the latter should be denied. The second use case is when this happens and the clients get a notification saying that there is a new version available. The third one is when a client asks for the latest version of a file.

![Figure 4.1: Use cases of the prototype.](image-url)
4.1.4.2 Architecture overview

The architecture of the prototype is based on a service-oriented architecture provided by CPaaS. The only drawback encountered during the design of the prototype is that the deployment platform is still not available for use. It is still being developed and there are no news about any release, so it had to be dropped out of the prototype. However, applications can be deployed independently and work the same way they would with the deployment platform.

In Figure 4.2 we can see the major components used in the prototype. First of all, the clients consuming the APIs access to it through the API Manager, which manages the APIs and its versions. The API Manager sends the request to the application, and the application requests from the different services the information needed. This is done through the ESB, which provides a unified interface for all the services. From the API Manager we can access also to the Message Broker and applications can read the queues of the Message Broker and send messages to it. Applications can also access the database.

As said, the aim of this prototype is to be able to upload files and receive a notification on other clients whenever it is uploaded and be able to fetch it. So there are two things that should be done, one is sending the file and the other one is storing in the database the information about the file. This architecture provides all the significant pieces needed to achieve that.

![Figure 4.2: Prototype architecture overview.](image)
4.1.4.3 How it works

Here we have mainly three functionalities that need to be supported. The first one is that we should be able to upload a file, handling the errors that might happen on the way and taking care that there can’t be concurrent uploads. The second thing is that we should be able to fetch a file, also controlling possible errors caused mainly by network connections. The third one is that whenever someone uploads a file, all the users working on the same project should receive a notification.

The process is the following one. A user wants to upload a file, so first of all it sends a notification to the server saying that it is going to do it. This is done using Messaging, so using AMQP the server receives the notification of someone trying to start an upload. The server-side application handles it, looks if there is already someone doing the same and if not, it creates a temporal record for that file in a separate table in the database. The temporal record is stored into the database through the ESB. The service that handles the database will provide an interface to interact with it.

Once we have the temporal record that prevents other users to try to upload the same version of the file, we upload the actual file. This is done by accessing to the application that sends the file to the file storage service through the ESB. Using this method we provide modularity and we can reuse the services without exposing them directly outside the server.

When the file is uploaded successfully, we can make official that the file has been uploaded by moving the temporary record to the final table with all the other records. It is then when we send a message to the Message Broker saying that there is a new version of the file. After that, all the users working on the same project should get a notification about it.

If some client wants to retrieve a file, it will call the application, and the application will check through the ESB what’s the latest version available in the database, in the final versions’ table. Then it will pick the file corresponding to that version from the file storage service, also through the ESB and send it back to the client along with the response.

4.2 Viability of the project

After analyzing and comparing the current architecture and CPaaS, we can agree that CPaaS is a suitable and viable option to consider. This solution satisfies all the requirements described in The requirements. Moreover, it allows modularity, enabling loosely coupled services that interact between them. This makes the services reusable and easy to maintain.
This is not only a recommended practice, but it is also aligned with the IT strategy and will enable future implementations provide a lot more value to the company. It would be an important step for future cloud solutions to use this kind of architecture.

There is a mention also about the performance. This architecture makes the performance dwindle but it is not really a big issue, since the aim it’s not to provide a lightning fast response, but instead provide a reliable system that can integrate and manage well all the complexity of the different cloud solutions. This is much more important and valuable for the company than the performance.

Also, all the current cloud solutions can be deployed into this platform without affecting to much to the current implementation and without having to modify the architecture of the software. So current applications can still use the same technologies, and CPaaS satisfies the requirements of the applications. This is also very important because otherwise it would be a lot of effort to adapt the applications to the platform.

So we have answered now Q1, and once this is done we will answer to Q2 in further chapters.
Chapter 5

Redesign of the architecture

5.1 Architecture redesign

The current cloud solutions can be ported to CPaaS to take benefit of its architecture, but what are the changes that need to be done in order to adapt it? What are the things that need to be redesigned in order to support the current solutions? This is basically what the research question Q2 asks, and it will be resolved in this chapter.

The current architecture, as explained previously, is very simple and does not allow modularity. The way that we could redesign the software in order to modularize the services used in the cloud solutions is by applying a service-oriented architecture. This will be a very important step in the long term since the number of cloud services offered is increasing and their complexity is also becoming bigger.

This new design will change a bit the current implementation of the different cloud solutions because from now on all of them are going to use reusable services that allow the same functionalities without having to maintain different services separately. The connexion between these components and the applications is what needs to be changed. Also, the services should allow different applications to use them. For example if previously one application was accessing a database that had a unique purpose, this database now will be converted into a service and it should allow different purposes for different applications. So in this case the database should provide different tables to different applications and different usernames with different rights for accessing the database through different applications.

5.1.1 Service-oriented Architecture

As explained in Background theory, SOA enables decoupling of the services and reusability. This is done through an ESB that enables the exchange of information between them
binding each service to a specific protocol or interface independently of the technology used behind. In the redesign of the architecture, the ESB will have a major role providing these service interfaces to the other services.

The applications will run as services also, so if in the future some new services need to reuse these services they will be able to do it. This is intentionally done so that the current solutions will be aligned with future business strategies. If there are new products and the strategy is to combine or integrate them to provide more value for the customer, these new products can be easily combined with the current ones.

In this service-oriented architecture, authentication and authorization will be also integrated in the same platform. Unlike in the current implementation, where authentication and authorization reside in a completely different place, the authentication and authorization will be hosted in the same platform and accessed via the API Manager that will expose access to it. The applications in charge of the authentication and authorization will be deployed as applications but won’t be accessible through the ESB directly. Instead, they will be exposed to the API Manager because external users need also to access to these authentication and authorization APIs.

Messaging is another component redesigned in the SOA. The Message Broker will be hosted inside the platform instead of being just an external resource. This means that now we should take care also of the Message Broker, provide scalability and make sure that it is reliable. However, it will also provide more flexibility to the system because it can be customized if there are some technical parameters that need to be changed in order to provide a more reliable persistence or if the performance needs to be tweaked. To get an idea, some of these parameters can be parameters about the database behind the Message Broker or if it implements replication, the heartbeat configuration or if the amount of memory given to the Message Broker. On the other side, for general purposes and if no special tuning is needed, a general vendor-specific Message Broker could be better since it is easier to maintain.

5.1.2 Services

To take advantage of SOA there are some basic services that should be implemented. This is not really mandatory but it would improve the modularity of the cloud services and it also would serve as a first contact with services to see how they can be used and reused in the different applications.

In a future, current cloud solutions should be modularized and served as services between them in order to take full advantage of the service-oriented architecture. This would be the ideal situation but it would need some effort in refactoring the applications and it really depends on many factors.
5.1.3 Application changes

The current software structure should be changed in order to take advantage of the SOA. For this reason the applications should be reorganized and redesigned to meet the requirements of a service and provide a good API to other services while taking care of reusing existing services also.

These modifications on the implementation don’t require a major refactoring to make them work with CPaaS components. However, if a true SOA wants to be implemented, some parts of the software such as database access or messaging should be decomposed and provided as a service. This allows current applications to be easily deployed in CPaaS without any major modification, but it also allows to use these pieces of software as services if required in the future.

5.1.4 BLOB storage

For the BLOB/file storage there are mainly two options and both are valid depending on the kind of service needed. The first option is to build a service in CPaaS that provides an interface to interact with some file storage or BLOB storage. This option would be aligned with the SOA and follow the desired architecture’s principles. However this can have some drawbacks. In practice, BLOBs can collapse the ESB and the ELB with large requests, slowing down the performance of CPaaS in the best of the cases and crashing the server for a lack of memory in the worst case.

The second option is to use the BLOB storage service provided by the vendor and use it directly from the client. This option, even if it is not aligned with the SOA strategy of service interoperability within the same cluster because it does not offer the service through the ESB, provides much better performance and reliability resulting into a better solution. In Figure 5.1 we can see how the decision of using BLOB storage from an external source without providing it as a service affects the architecture of the solution.
This option is forced by the fact that some of the middleware components can collapse if the client sends a large request, as it is expected to happen with large BLOBs. This approach is also dependent on external file storage providers, but it is the best option anyway until CPaaS supports large requests without collapsing.

5.2 Results of the redesign

This redesign of the cloud solutions would make a huge impact on the software architecture of the current and future solutions. The modularity achieved with this redesign will allow the alignment of the different services by reusing the services and providing interoperability. Making these services reusable will allow future services to reuse the current software, not having redundant code and easily extend and integrate different features, products and applications between them. So we ensure that future projects can be easily aligned to the current ones.

With this redesign we can solve Q2 and assert that an implementation of the cloud services in CPaaS would work fine thanks to the prototype that acts as a proof of concept. It is proved also that with this design there will be better interoperability between services and applications, allowing further implementations to be easy to integrate.

In Figure 3.2 we can observe the result of what CPaaS is and how web applications and services are going to be placed within the architecture. In the prototype implemented there are mainly three services provided: a database service, a file storage service and a
web service that handles concurrency and versioning of the files. However, we can host as many services as we need and enable intercommunication between them.

Another of the results of the redesign that should be mentioned is the difference between the theoretical scheme and the one that should be used because of realistic performance problems. This is the case of the BLOB storage service, which should be one of the services provided through the ESB but for performance, security and reliability reasons it will be externalized to an external provider’s solution. This does not change the overall picture a lot, and provides better performance and reliability of the whole platform. These are the few changes that might defer from the theoretical scheme, and it is something that should be in mind also.

In Figure 5.2 we can see how different services will communicate with CPaaS and how the architecture is going to help services to interact between them.

![Figure 5.2: Final scheme of CPaaS and the interaction between its components.](image-url)
Chapter 6

Planning

6.1 Final planning

The final planning has been just slightly different to the one first planned. There was a lot of time invested in making a good guess of the time needed, what would be the main points, the checkpoints and possible threats; so the guess was quite good and the middle checkpoints enabled a good metric of the progress of the work adding a bit of rush when needed. Some things were slightly different and delayed the project a bit. However, because there was time spent in planning, the guess was pretty good.

The fact that there were useful milestones that were used as a checkpoint of whether the project was on time or needed some more attention and extra hours of work. Also there was a lot of support within the company and from the supervisors, which eased the learning curve and helped to make clear the scope of the project in moments of uncertainty.

The total amount of hours of this project goes up to 920. During normal workdays the average amount of hours has been of 10 hours a day. So for the 87 working days this is 870 hours of work. Plus, some extra time has been spent on weekends. Even that this extra time spent outside the normal working hours is difficult to count, it could easily add 50 hours at least.
<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
<th>Pred/Resource Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>Redesign</td>
<td>0 days</td>
<td>Sun 6/4/14</td>
<td>Sun 6/4/14</td>
<td>Magrinya Xbei</td>
</tr>
<tr>
<td>31</td>
<td>Conclusion</td>
<td>0 days</td>
<td>Sun 11/5/14</td>
<td>Sun 11/5/14</td>
<td>Magrinya Xbei</td>
</tr>
<tr>
<td>32</td>
<td>Finish up</td>
<td>0 days</td>
<td>Fri 30/5/14</td>
<td>Fri 30/5/14</td>
<td>Magrinya Xbei</td>
</tr>
</tbody>
</table>
6.2 Problems encountered

In general, the project went quite smooth but there were some problems encountered on the way. At first, there was no knowledge about CPaaS and I had to contact team leads of the product working in other countries to get some information to start with. The lack of information was constant along the beginning of the project, so in that sense there was a lot of research done that was really useful and provided valuable information to the company. Even inside the company, there was no such knowledge because the development of this new platform was still on process. However, some online meetings and a lot of personal research solved the problem.

There was also some confusing information about the platform itself. There is a lot of documentation but it required to get my hands on it and make it work to see what was it actually and how it works. This took also some time, because there were lots of guesses of the way it works at first that were actually wrong.

On the other hand, some external delays of the platform’s providers created a bit of inefficiency because all the information that could be provided straight by them had to be guessed and extracted from another similar platforms. Also the sandbox release had been delayed, which made that instead of using some out-of-the-box solution, I had to build my own using similar software that required a much more complex installation.

However, because in the project definition we took into account that there could be some difficulties on the way, the planning went more or less as expected.

6.3 Costs of the project

There are several factors that determine the total. We used approximately around 870 hours of development at work made by one software engineer and 70 additional hours more due to different meetings with other software architects and engineers. Each of these meetings was expensive in terms of resources because there were different people involved so the time spent in the meetings should be multiplied for the amount of people attending.

Aside from that, we should take into account the expenses of the hardware and software used, as well as the office space and other minor expenses like power consumption and servers. The software and the hardware used are a bit expensive because we used powerful desktop machines and some expensive software licenses. The total price of the whole project is of 28923 €.
<table>
<thead>
<tr>
<th>Resource</th>
<th>Price/unit</th>
<th>Amount</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software engineer</td>
<td>23.50</td>
<td>870</td>
<td>20.445,</td>
</tr>
<tr>
<td>Software architect</td>
<td>36.70</td>
<td>70</td>
<td>2.569,</td>
</tr>
<tr>
<td>Office space</td>
<td>550,</td>
<td>2</td>
<td>1.100,</td>
</tr>
<tr>
<td>Power consumption</td>
<td>15,</td>
<td>2</td>
<td>30,</td>
</tr>
<tr>
<td>Servers</td>
<td>50,</td>
<td>2</td>
<td>100,</td>
</tr>
<tr>
<td>Hardware</td>
<td>2.470,</td>
<td></td>
<td>2.470,</td>
</tr>
<tr>
<td>Software</td>
<td>2.209,</td>
<td></td>
<td>2.209,</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>28923.00</strong></td>
</tr>
</tbody>
</table>

Table 6.1: Expenses of the project (amounts in €)
Chapter 7

Conclusions

7.1 Analysis of the results

After having answered the questions Q1 and Q2 and showed the results we are going to analyze them and extract some conclusions.

First of all, the results showed that porting our current cloud solutions into CPaaS is feasible and this means that not only current solutions can be ported but that future solutions can also be implemented using CPaaS as a main platform.

The fact that current solutions can be ported into CPaaS is a precondition and completely necessary for further steps in the development of cloud solutions using this platform. However, this is not the main point of this change. As we have seen along this document, SOA provides many advantages for large corporations’ cloud services. In general, SOA is a good approach to provide well structured and reusable software for the different stakeholders.

In the case that we studied in this thesis, it means having a well-formed base where to build further cloud solutions and also adapt the current ones. It is basically a statement of intent and points out a direction to further software development that would enable interconnecting and centralizing the information in the cloud. These advantages are the main point of changing to a platform with SOA.

After the deep analysis done, we can assert that SOA is a very good approach to the kind of architecture needed for a large corporation’s cloud solutions that enables growth and scalability without painful changes, and this makes CPaaS a good platform to achieve that.
7.2 Future work

This thesis opens up a bunch of possibilities for extending the topic. If we divide some of these possibilities into different topics, we could classify them as it follows.

- **Deployment platform** Some work regarding the deployment platform can be done in order to provide a cross-language deployment platform that would provide a good interface, reliability and flexibility. Nowadays new web technologies emerge constantly and it would be good to extend a deployment platform to support future frameworks and languages also.

- **Statistics and metrics** SOA also allows you to monitor the different services in a separate way, providing a good architecture to get valuable metrics of the usage of the different services. There are already components that aim to provide statistics, but they could be extended and there could be more features and statistic metrics useful to implement.

- **Optimization of BLOB storage** One of the problems encountered in SOA is that large requests are difficult to handle. One of the improvements would be to enable SOA to handle them, providing some special functionality for those requests that would treat them differently without blocking the whole system. This would make SOA more modular and improve its functionality.

These are just some examples, but there is much more work that could be done. For example in the business side, it could be a study of how SOA eases the addition of new processes in a company, or what are the effects of using SOA to include new processes.
Appendix A

Project Definition

Started on:
JANUARY 2014

A.1 Introduction

A.1.1 Project’s frame

Our company has been developing high quality desktop software for over 20 years. The code base is composed by millions of lines of legacy code. The usage of this software has been traditionally offline, but when cloud services started emerging new features were developed to fulfill the customer’s needs and make the software more valuable by enabling some online collaborative features. These features are stored in a closed PaaS that was good when the project was not too big. As the demand of users requesting these online features grows, our company needs a more scalable solution not tied to the prices and availability of any closed platform.

A.1.2 Project proposal

The company that owns our company has recently developed a customized PaaS and our company wants to study the viability of using this platform for their cloud services. This customized PaaS uses various providers to optimize the cost and availability of the services depending on the demand and usage. It would be a great improvement to integrate those services into the customized PaaS because the cost of the cloud services would decrease dramatically.
A.2 Understanding the why

Integrating collaborative work in the software is a key aspect in order to satisfy our customers’ needs. Large multidisciplinary teams use the software regularly to work in the same project and all this work should be synchronized using cloud technologies. These features are now somehow implemented, but they imply a relatively high cost because the implementation is tied to a closed PaaS with higher prices than their competitors.

Lowering the costs will be a big improvement since these features are planned to be scaled to lots of users in many companies around the world. Also, more features are planned to be implemented in the cloud extending the current ones and providing a better product to the customers. Of course, all these things will provide more value to the company, are aligned to the company’s strategy and they are planned for prospective versions of the software.

A.3 Scope of the project

The aim of the project is not to build a completely functional system using the customized PaaS but to investigate more about what it is, how it works and if it fits for our purposes. This will be the main part of the project since the customized PaaS is very new and the way it works is completely unknown. After analyzing the platform and if it fits the needs of the company, there will be a redesign of the current implementation of the cloud services in order to use the new platform and possibly a small prototype of how it will work.

The scope of the project can be very vast, but the focus won’t be on any implementation of the current services. Instead, we will focus more on the analysis of the new platform, the pros and cons and an overall picture of how the software could be redesigned. Only if this stage is reached, the scope can be adjusted and extended to fit also the prototype in the project. It is more important and valuable for the company to have a reliable analysis of the platform and to have more knowledge about it rather than spending time implementing something pointless.

A.4 Deliverables

- **Project Definition**: this document defines the aim of the project, the scope and other factors to consider.

- **Thesis document**: document describing the project, what has been done and the conclusions of the study.
A.5 Possible problems to overcome

As this customized PaaS is something new and still not finished, it doesn’t have much documentation. This will be one of the major problems to overcome. The investigation of this platform can be very time consuming and it is planned to last more than other tasks in the project. It is also a very important task, we should pay a lot of attention to it.

Another possible problem might be the lack of resources, mainly time resources. Since there are lots of things to do and it is difficult to predict how long are they going to take, the timing is a key factor in the planning. Writing the documentation can be also very time consuming, but it is as well an essential part of the project.

A.6 Specific conditions

The commercial name of the brands used and the products’ names are under the company’s privacy policies, so they cannot be mentioned in any document that could be delivered to the University or could compromise the confidentiality of the project.

<table>
<thead>
<tr>
<th>Completed on:</th>
<th>Completed by:</th>
<th>Agreed by:</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.06.2014</td>
<td>Xavi Magrinyà</td>
<td>Teemu Heikkonen</td>
</tr>
</tbody>
</table>
Bibliography


