Choosing a Platform as a Service for Software as a Service Provider
Abstract

Cloud computing is still very sparsely researched, especially concerning platform as a service (PaaS) deployment model. PaaS model is still a very new in cloud computing and often confused with more familiar IaaS and SaaS models. First the basic concepts of cloud computing such as essential characters, deployment models and service models are presented and everything in cloud computing lean on these main concepts. The main objectives in this thesis are, to find out why PaaS is better alternative than building an own platform over a selected infrastructure, and to justify, why the selected PaaS is the most suitable for a cloud software project.

PaaS has many advantages and concerns that must be sort out before a decision of deploying one can be made. With PaaS, developers do not need to know much about underlying cloud infrastructure and the infrastructure services are usually completely hidden, instead they can focus more on the development. PaaS platforms bring also a lot of cost reductions from reduced hardware needs, staff and more predictable expenditure. The biggest challenges in PaaS are the security and privacy concerns, and possible vendor lock-in risks.

The four PaaS alternatives which are measured are Amazon Web Services with Beanstalks, Google App Engine, Heroku and Cloud Foundry. These are selected from number of alternatives together with the contractor. These four were among the most well-known PaaS providers and they were the most suitable with the contractor's project and with the criteria. To measure these, five justified criteria, which are derived from the literature, are defined. Measurable factors are derived from scalability, service agreements and vendor lock-in, security, framework and supported languages and pricing models.

Choosing-by-advantage (CBA) decision making method is used to find out the best alternative. CBA is made for fast and sound decision making and it is based on the importance of advantages. The factors from the criteria are derived in away that they should uncover the advantages of each PaaS alternative. Disadvantages are also found out and listed to strengthen the decision further. With the help of the justified criteria and the CBA analysis, the goal is to support and justify the PaaS decision, which will be made.

Keywords
cloud computing, platform as a service (PaaS)
Foreword

I would like to thank my family, friends and colleagues for support and help during this thesis process. I would especially like to thank Product Owner and Board Member Tiina-Maria Siipola, and the rest of the Kodifin Oy staff for giving me the opportunity to do this thesis for them and work with the company during this time. I would also like to thank my thesis supervisor Assistant Professor Henrik Hedberg for many great advices and tips how to proceed after many challenges I encountered during this thesis writing process.

Antti Kreivi

Oulu, March 6, 2014
Contents

1. Introduction..........................................................................................................5
   1.1 Background and incentives for research........................................................5
   1.2 Research questions and research methods.....................................................6
   1.3 Limitations.....................................................................................................6
   1.4 Contractor's case............................................................................................7
   1.5 Contributions and goals.................................................................................7
   1.6 Structure........................................................................................................8
2. Concepts...............................................................................................................9
   2.1 Cloud computing and it's essential characteristics........................................ 9
   2.2 Deployment models.....................................................................................12
   2.3 Service models.............................................................................................17
3. Prior Research....................................................................................................22
   3.1 PaaS advantages..........................................................................................22
   3.2 Target group of PaaS...................................................................................25
   3.3 Open source in PaaS platforms.....................................................................26
   3.4 General PaaS concerns...............................................................................27
   3.5 Scalability....................................................................................................27
   3.6 Service agreements and vendor lock-in.......................................................30
   3.7 Security........................................................................................................33
   3.8 Framework and supported programming languages................................... 36
   3.9 Pricing..........................................................................................................38
   3.10 Other aspects.............................................................................................40
4. Selected PaaS Alternatives.................................................................................41
   4.1 Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Beanstalk..41
   4.2 Google App Engine.....................................................................................48
   4.3 Heroku.........................................................................................................56
   4.4 Cloud Foundry.............................................................................................62
5. Choosing by Advantage and Findings...............................................................68
   5.1 Choosing by Advantage decision making method......................................68
   5.2 CBA Analysis for the selected PaaS alternatives........................................69
6. Conclusion..........................................................................................................77
1. Introduction

Nowadays in the world of technology, cloud computing is the buzzword and concept that appears almost everywhere in the computing industry: in articles, in web forums/blogs and in large technology expos everywhere in the world. It is said that the future of computing is in the clouds and cloud computing has come to stay. Still many shy away from cloud computing and find out too many concerns to embrace it. Also many do not even know that they use cloud environments everyday to share, save and store data with their laptops and mobile devices. (Huotari, 2012).

1.1 Background and incentives for research

The basic idea of cloud computing is not that unique. After all, it is all about centralizing the computing resources and storages in the Internet and accessing applications and processing power externally. Still the basic idea of cloud computing is so attractive that the hype is justified. Who does not like the idea that applications can be run without installing them locally or more CPU power and harddrive space can be enabled without the need to buy expensive hardware, instead pay only for the resources needed? After all, all of those aspects are still a major expense in many organizations and companies. (Barnatt, 2010; Joseph, 2009).

Even if cloud computing is discussed a lot and there are a large number of research articles, documents and blogs all over the Internet, there is still a lack of research from a certain point-of-views. It is said that cloud computing is a great business idea and a future proof concept for new businesses, especially in high developed countries such as Finland, but there are not many examples yet, because the concept is so new. Certain cloud computing scenarios such as, the usage of Platform as a service(PaaS) service model for cloud computing software project, is still very much uncovered and that is one of the incentives and motivation for this thesis.

The main focus of the thesis is the platform of a service(PaaS) service model. PaaS is one of the three service models and it is situated between the bottom infrastructure level(IaaS), and it functions as a platform for the higher level services(SaaS). The thesis focuses on the possibilities, features and advantages, but also on the challenges, the PaaS may provide for cloud computing providers. For example, what kind of value can PaaS platform bring for software as a service(SaaS) and cloud application provider and is the PaaS platform worth the effort and additional costs? PaaS platforms are still recently new idea compared to IaaS and SaaS and the research material of them are really scattered and the whole basic concept of PaaS is still quite vaguely described and one of the goals of this thesis is also to give understanding what PaaS platforms really are and what they are not. SaaS, PaaS and IaaS layers are many times confused with each other or misunderstood completely. For example, the largest cloud provider Amazon Web Services(AWS) is widely used as a cloud platform even if it is not classified as a PaaS itself. Still many people seem to think so. People think that when they are programming in the cloud, they must be using PaaS. If AWS would be a pure PaaS, the infrastructure services would be completely hidden. People are still quite reserved about PaaS platforms and currently many developers and people are still
testing them but not relying on them. PaaS market is expanding and new providers are popping up constantly but at the same time many big players are yet to be found in the PaaS market. (Mann, 2013).

1.2 Research questions and research methods

The research questions of the thesis are: “Why choose Platform as a Service instead of building an own platform over a selected cloud infrastructure? And as a follow-up question ”What kind of criteria can be used to find out the PaaS alternative?” Possible answers for the first question are proposed first by making a general benefit and risk survey of choosing PaaS, by going through a number of different references including scientific journals, magazine articles and web blogs/forums. When going through these, great source criticism must be practiced because there are very few proper scientific references to use. Much of the references are based on opinions of industry's well known people or some cloud service providers' own publications which must be read with caution, so source criticism is the key when handling the source material. The PaaS selection is done with the help of predefined criteria and advantages derived from the criteria. This criteria is based and derived from the first research problem: the benefits and risks of PaaS platforms. There are also justified reasons behind the selected criteria and the criteria is rationalized. This criteria and advantages are resolved by measuring the benefits and risks of the PaaS platforms. Following criteria are measured: Scalability, Vendor lock-in/service agreements, Security, Framework(supported programming languages, portability) and pricing(cost structure, pricing methods).

The qualitative research methods used in this research are exploratory research and Choosing By Advantage(CBA) for decision making. The research is exploratory as its nature because it should find new kinds of solutions and views of cloud computing and PaaS platform research. The research can also suggest hypotheses such as “PaaS platforms enable faster and cheaper way of developing cloud content” and “PaaS platforms are the best way to go for cloud software startups” (Tuomi, 2007.) The Choosing By Advantage(CBA) method is used when measuring and deciding which PaaS alternative is the most attractive to choose. CBA is a decision making method which should produce sound, reliable and grounded results. For example if CBA is used for selecting a car to purchase, one must measure the advantages not the factors. Factors are things like aesthetics, economics, safety, etc and those can not be measured as they are. Instead, advantages can be derived from factors such as safety. If safety is factor to be measured the advantage for that could be for example the number of airbags. The more the car has airbags (an advantage), the better. The PaaS alternatives are measured and compared in this way and the advantages are derived from the predefined criteria (scalability, security, vendor lock-in, frameworks and pricing models). CBA method is simpler and faster so it suits very well for this research. (Suhr, 1999).

1.3 Limitations

Because the time frame in which the contractor is going to made the PaaS decision is short, the alternatives must be limited only to four. There could also be more criteria but for the same reason as above, the criteria is only limited to a five different but well justified one. In this research, the point of view is mainly from a perspective of a small company and the whole research could be different if the scope would be bigger. Also the PaaS decision could be more definite if all the different platform alternatives could be tested against the criteria separately in real development environment in addition
with the CBA analysis but again, time frame and resources are too short and low respectively. Concerning the literature, there could be more proper scientific articles and maybe by having researched more from other fields, more scientific references could have been found. Also more scientific analysis methods for the alternatives such as ATAM could have been used but then the scope would have probably be too large for a master thesis.

1.4 Contractor's case

The contractor of this thesis is a small Finnish startup Kodifin Oy, which is located in Oulu, Finland. They are developing a documentation environment which functions in a cloud. It is planned to serve a solution for various kinds of documentation needs for small and/or big organizations and projects. With the help of the product, creating, reading and distributing documents should be fast and easy between the stakeholders, project managers and developers. The goal of the company is to make a product family for various documentation types and needs; easy to use, fast and profitable toolset; uniform and modifiable document creation environment which supports co-operation and realtime editing and working. At ideal situation, the toolset would replace common office tools like MS Office altogether.

The first product which is under construction is a requirement management service(RMS) which is implemented as a cloud-based service and a software as a service(SaaS). With the help of the RMS, the projects can make robust and uniform requirement specification documents for various kinds of projects. The idea is that the customers do not need to install or save anything in their own computers or databases. Everything is saved and functioning in the cloud and the only thing the customer needs is a web browser. The tool also offers real time communication plugin which helps the communication with the project managers, developers and stakeholders. In addition from the tool itself, the goal is to offer a good documentation and requirement management habits for the customers as a side product. The tool is constructed in a way that the documents are built from various kinds of document components(headings, table of contents, requirement specifications, etc.) and the when document is ready, the tool forms a printable pdf. The document also offers ready made and configurable templates for the customer’s own desires(logos, fonts, design).

The incentive for this research from the contractor's point of view is to find a cloud-based software development platform for the product family. In an ideal situation, the selected PaaS platform would work as a platform and sort of an ecosystem for whole lifespan of the product family and also for completely new products. The other alternative would be to go only for the infrastructure services(IaaS).

1.5 Contributions and goals

The major goal of this thesis is to propose a new view point for cloud computing research and to the PaaS service model. There are also very little proper scientific research of this particular topic and the research, where combination between PaaS platforms and newly found cloud computing software project are presented, can not be found anywhere. This thesis should be a good read for anyone who is interested in establishing a cloud computing company, starting a new cloud software project or being part of one. Also this thesis is for those who are interested in PaaS platforms and their features. The objective in contractor's point of view is that this research should help the
company's decision of whether to take PaaS as a part of their strategy or not and help to choose the right PaaS for their needs if it is chosen to bring into use. Also because the specific topic is current and only marginally studied, there could be a possibility for a magazine article concerning the thesis topic for some large paper in the industry such as T3.

1.6 Structure

The structure of this document is constructed in a way that, first the reader should know the basic concepts of cloud computing. These include the essential characters of cloud computing, deployment models and service models. Next all the meaningful prior research which were found through the research are gone through. These include cloud computing for newly found companies, the reasons, benefits and challenges of choosing PaaS platform and PaaS open source possibilities. Next the criteria for the PaaS measuring is presented. Each criteria is justified from the knowledge found out in the earlier chapters and the advantages from each criteria are derived from this knowledge. When the criteria is sort out, the PaaS alternatives are gone through. There are four alternatives for review. These alternatives are gone through briefly(main components, what they offer, etc.) and they are measured against the criteria and the factors. After that, the findings are found out by choosing by advantages(CBA) analysis and the best PaaS alternative is proposed based on the results. At the end the results and the whole research is discussed and it shall be seen if the research is successful or not.
2. Concepts

There are lots of key concepts and possibly hard to understand concepts and terms concerning cloud computing and cloud services. These concepts and their definitions described in this chapter are based on the work of National Institute of Standards and Technology (NIST) which functions under the Federal Information Security Management Act (FISMA) of 2002, Public Law 107-347. (Badger et al, 2011; Mell & Grance, 2011).

2.1 Cloud computing and it's essential characteristics

According to National Institute of Standards and Technology (NIST): “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.” (Mell & Grance, 2011).

What does this concept mean? In an ideal situation cloud can be accessed from anywhere, anytime and on-demand. All the needed resources like networks, computing power and services (applications) are apparent in a non-transparent cloud. Depending on the cloud type the concept might considered different (see Deployment models of the cloud).

2.1.1 Key characteristics of Cloud computing

NIST defines the standard key characteristics for cloud computing which should be valid for any cloud environment and service.

On-demand self-service means that user can use the features of the cloud such as network storage any time without interaction with the cloud service providers. User can access the cloud and its network with any device which provides the possibility to access to the Internet (workstations, laptops, mobile devices etc.). This means that the cloud has broad network access. Cloud service provider must secure that their services functioning in the cloud are available for customers at anytime. These services include computing power, storage and network bandwidth. This means that resources must be shared in a way that the unneeded resources are not used but only resources that the customers need that time. This is called resource pooling. Users don’t usually know or have control the location of the resource pool but in some cases the high level abstraction of the location can be chosen (continent, country etc.) (Mell & Grance, 2011).

Rapid elasticity is a concept that is affiliated to resource pooling. Resources can be increased and decreased based on the needs of the customers. Resources can be put into service or reclaim back (and possibly export elsewhere where needed) depending on the load. Usually the user don’t see this. According to Armburst et. Al (2009), the elasticity still has some problems of provisioning the load because the demand varies periodically.
and because it can take some time to acquire new hardware and resources, the spike handling must be done and predicted in advance. This can cause problems in provisioning. (Armburst et al., 2009; Mell & Grance, 2011).

![Diagram](image)

**Figure 1.** Issues in provisioning the resource demands (Armburst et al, 2009).

*Measured service concept* is also affiliated to resources and their usage. Cloud systems monitor, control and report the change in load, both for the provider and for the users. Storage, computing resources (CPU and memory usage), bandwidth and user accounts can be monitored. (Mell & Grance, 2011). In addition to NIST defined characteristics, Järvi et. al. define few additional concepts. Many cloud services has no sign-up or starting costs, instead the user pays only for the usage. When the usage gets larger, the cost increases and when the usage is going down, the costs go back down. This is called *pay as you go*. There is also *as a Service* model, which means that the service provider provides the resources, hardware and maintenance. The service is produced in provider's own premises. (Järvi et. Al, 2011).

### 2.1.2 General cloud environment

In cloud systems, their providers and consumers transfer messages over a network between the servers and clients’ workstations. Servers work according to the messages. This relationship function according as a client-server model. In general cloud view, the system is described as a grid of computer systems where the resources are stored and services accessed by the clients. Cloud provider can remove old hardware and replace them with new. Cloud can be thought as a living ecosystem where clients leave and enter the cloud simultaneously and hardware is changed or replaced from time to time. (Badger et al, 2011).
Badger et al. (2011) also define following general assumptions and risks concerning organizations which decide to take cloud computing as part of their strategy. These apply in any cloud type and in any service model deployment:

- **Network dependency**: Clients need working and secure network connection to access the cloud services. Poor or defective network access decreases the ability to use the cloud services.

- **IT skills demand**: Many times cloud service providers don’t have much IT staff in consumer organizations so clients have to maintain and secure the client system.

- **Workload locations are hidden**: Clients usually don’t know where workload is computed, because usually the workload locations are dynamically assigned to manage cloud’s hardware resources efficiently. Tracking and adapting changes should not needed to be done by the clients.

- **Multi-tenancy**: Different clients may have the same cloud provider resources (system/network) in use at the same time. If cloud provider’s management or network security is not sufficient enough, the security of the client may be compromised.

- **Performance limitations**: Because cloud is accessed through network, there are always risk of poor connection due to network load or network speed. (Badger et al, 2011).

Compared to traditional on-site computing, in cloud computing the organization must usually at least in some content, drop both control over the underlying cloud infrastructure and visibility. However many times the customers of cloud provider may need to have control and visibility. For these cases, the cloud has a boundary controller which handles the access between the outside and inside of the cloud. (Badger et al, 2011).
Depending on the cloud deployment model, the security perimeter configuration and access to the boundary controller can be deployed different. The security perimeter can be either on-site or outsourced. (Badger et al, 2011).

2.2 Deployment models

There are four types of clouds conceptualized: Private cloud, community cloud, public cloud, hybrid cloud. These are called deployment models and they are also defined by NIST. Cloud deployment models described in this chapter are based on NIST’s official definitions and recommendations. (Badger et al, 2011; Heiska, 2012; Mell & Grance, 2011).

2.2.1 Private cloud

Private cloud functions usually within an organization or a community and besides the members of the organization, the cloud cannot be accessed nor its content be seen by other users. Customers in private cloud are usually business units of a particular organization. (Mell & Grance, 2011).
Private cloud is inside the security perimeter and boundary controller offers the “gate” inside the private cloud environment. Through the boundary controller, the customer have control over the entrusted resources inside the security perimeter. When considering implementing the private cloud as a business strategy the organization must consider all the predefined risks (network dependency, needed IT-skills, hidden workload locations, multi-tenancy, performance limitations) and in addition, think about how to secure the private cloud, increased costs from cloud management software and possibly limited resources. (Badger et al, 2011).

If the customer decides to outsource the private cloud, there are two different boundary controllers: One for Cloud provider’s cloud and one for Consumer’s own security perimeter.

![Diagram of Outsourced Private Cloud](image)

**Figure 5.** Outsourced private cloud (Badger et al, 2011).

If private cloud is outsourced, the general predefined risks are better understood and controlled. Customer and private cloud provider can arrange for dedicated leased network connections, visibility for workload locations and high-performance networking to be provided by the cloud provider. The tradeoff in this is that, it can increase the costs for the customer dramatically. (Badger et al, 2011).

### 2.2.2 Community cloud

Community cloud is a kind of private cloud in which multiple organizations have access to it. Community cloud can have storages and shared resources for multiple organizations that need them and are part of the community. It can be owned and managed by multiple organizations. (Mell & Grance, 2011).
In on-site community cloud model, each organization have their own security perimeters and boundary controllers. Organizational perimeters may also contain secured local cloud segment which may contain critical data. Community cloud offers many configuration possibilities between the organizations which access it. The easiest way of defining community cloud would be that it is a collection of private clouds linked to each other. The challenge in managing the community cloud is the decision about how to share the resources between the participants. (Badger et al, 2011).

The same general risks are presented also in on-site community cloud as in on-site private cloud. The amount of managing, maintenance and resource needs are naturally increased when more groups are accessing the cloud and more secure perimeters there are. It is also important to understand the dependencies between the organizations in the cloud infrastructure when concerning IT skill needs and network dependency. Workload locations must be known at least in that content, that it can be documented properly. Outsourcing these tasks is also an option. (Badger et al, 2011).
Outsourced community cloud scenario is very similar to outsourced private cloud scenario. Cloud provider is responsible for server-side and cloud provider provides perimeter and boundary controller for the community. The general risk statements for outsourced private cloud remain largely true also in outsourced community cloud. (Badger et al, 2011).

2.2.3 Public cloud

Public cloud provides an open use by general customers and public. It can be managed by private organizations, a business, an academic facility or combination of them. It can also be operated by an open source community. (Bawa & Girdhar, 2012; Mell & Grance, 2011).
Public cloud scenario is very similar to the general cloud view described in figure 2. Main difference is that the customer of public cloud are illustrated as its own secure perimeter and the customer has also its own boundary controller. The consumer’s storage and computer resource needs are potentially much larger than in public client’s case; also the communication links are usually carried over the public Internet, so the cloud have more diverse client pool and possible security attack risk is larger. (Badger et al, 2011).

Because the public cloud is accessed through public Internet, the network dependency depends on Domain Name System(DNS) servers. Workloads can be located anywhere but provider may offer location restriction policies and clients can configure their accounts to a specific location. The locations are usually very scarce and there are not much options for more specific locationing. Because elasticity, scaling and resource optimization is important in public clouds, there might be a trade-off concerning multi-tenancy and increased risk of possible attacks. (Badger et al, 2011).

2.2.4 Hybrid cloud

Hybrid cloud can be a combination any or all of the previous cloud types. Hybrid cloud can be thought as a combination of other clouds and those clouds function as entities inside the hybrid cloud. (Mell & Grance, 2011).

![Hybrid cloud scenario](image)

**Figure 9.** Hybrid cloud scenario (Badger et al, 2011).

Hybrid cloud varies between very complex configurations to more simple one. There are many situations where hybrid cloud configuration can be useful. For example consumer can concentrate the routine workloads to private cloud and if the load increases, share the workload to one or more external clouds. One cloud can also be
function as a backup for critical data, etc. Many times, a newly developed cloud service software run on multi-cloud configurations (see service models). For example the contractor of this thesis will be using hybrid cloud environment including private cloud (company’s critical data and resources) and public cloud (the Software as a Service for the clients of the product) (Badger et al, 2011).

2.3 Service models

There are three main service models defined by NIST that can be applied depending on the business model. (Badger et al. 2011). The main three service models are Software as a Service (SaaS), Platform as a Service and Infrastructure as a Service (IaaS). As stated by Lenk et. Al (2009.), the service models can be compiled in a stack where all the models are linked and related to each other can be seen.

Figure 10. Cloud stack (Lenk et al, 2009).
There are also unofficial service model types such as Humans as a Services (HaaS) and an emerging idea of Everything as a Service (XaaS). The reference stack of service models defined by Lenk et. al. (2009) are based on the idea of XaaS: A full combination of all the possible service models combined together. (Lenk et. al, 2009; Mell & Grance, 2011).

2.3.1 Infrastructure as a Service (IaaS)

IaaS is located closest to the physical hardware. IaaS provider offers computing power, storage, networks and other resources for customer to deploy and run its software which can include operating systems and applications. The customer doesn’t have direct control over underlying cloud infrastructure but there are possibility to control over operating systems, storage, applications and sometimes limited control of network components and security. (Mell & Grance, 2011). IaaS can be thought as a conventional data center in which the resources are virtualized. Usually the customers have direct control over servers and resources(the ability to control instances all the way to OS-level) and they can usually be deployed with a push of a button. IaaS offers also downward scalability and scaling is fast. (Järvi. et. al, 2011).

2.3.2 Software as a Service (SaaS)

In SaaS, the service provider offers an application which works in a cloud. For example Google docs (Google Drive) and MS Office 365 are SaaS services. In SaaS, customer does not need to install anything locally and anything that is needed is in the cloud. Because the application is hosted by the provider, users do not have to worry about software updates neither. The application is usually accessible with a web - or a mobile browser. The users usually don’t have any control beneath the application i.e. the underlying infrastructure. (Lenk et. al, 2009; Mell & Grance, 2011). Lenk et al.(2009) also distinguish SaaS layer in Applications and Application Services. Application Services can further be composed and divided in Composite Application Services(for example Opensocial) and Basic Application Services such as GoogleMaps and Myspace. (Lenk et. Al, 2009).

2.3.3 Platform as a Service (PaaS)

The main focus of this thesis is the PaaS platforms and they should be understood properly. First it is important to know where PaaS stands in the service model architecture. PaaS is located between the lower IaaS and upper SaaS models (figure 11.). IaaS services are the basis for PaaS services and many times the PaaS provider hides the infrastructure from the users. SaaS services can be built over a PaaS, using the tools and frameworks it offers. Services such as Google App Engine, MS Azure and Heroku are PaaS platforms. (Joseph, 2009).
In PaaS, the service provider offers a set of programming languages, libraries, services and tools which the customers can use to create and deploy their own cloud application or a service. Developers can build SaaS services over a PaaS platform. Ideally PaaS customer does not manage or control the cloud infrastructure (network, servers or storage), instead they are done by the PaaS provider. Customers have control over their deployed application and configuration options for the hosting environment. According to Orlando (2011), PaaS layer is generally the most confusing of the service models because it is hard to distinguish from IaaS or SaaS layers. (Mell & Grance, 2011; Orlando, 2011).

Lenk et. al.(2009) categorize PaaS layer also in Programing Environments and Execution Environments. Examples of PaaS services include Django(Programming Environments), Google App Engine, Heroku and Microsoft Azure(Execution Environments) PaaS can also be created by the cloud customers themselves(or modify open source solutions to their needs(Carlsson, 2012) but the existing options are usually better particularly for a smaller projects(such as an entry level cloud service) because it
takes a lot of time and expertise to create, maintain and configure PaaS.(Lenk et. al., 2009).

When choosing PaaS, it is important to know what kinds of features are needed for the particular application that is deployed. If PaaS selection is done wrong, the customer might pay for unnecessary features which can be a key to a failure especially for smaller application projects. Resource pooling and rapid elasticity are usually the key aspects when selecting a PaaS service. Correct PaaS selection means that the time spent on developing the system architecture is saved which means that business agility is increased. When developing an application over PaaS, the development costs can be only marginal but the application can still serve a large group of customers. (Järvi et. al., 2011).

Järvi et. al.(2011) categorizes PaaS providers as “Infrastructure-PaaS” platforms and “Application developer-PaaS” platforms. Infrastructure-PaaS hides the infrastructure below completely and customer can focus their efforts completely on the application development. Examples for Infrastructure-PaaS includes Google App Engine(GAE) and Heroku. Application developer-PaaS provides a simple graphical way of developing the application(WYSIWYG, What You See Is What You Get). This means that the PaaS generates the code without programming needed by the application developer(for example LongJump). The difference between Infrastructure-PaaS and Application developer-PaaS can be dramatic and this is one of the things which may be confusing. (Järvi. et. al, 2011; Dictionary.com, 2012).
The customer using external PaaS solution can solely focus on application development instead of wasting resources on maintaining servers, storages and networking. This is particularly important for small development teams and companies, since server -, operating system -, storage -, etc. costs should not be worried. (Heiska, 2012).
3. Prior Research

Despite that there are lot of various type of literature and theory concerning cloud computing and its various aspects generally, there are not much literature or research concerning whether to choose PaaS as a strategy in SaaS business or not, or even about PaaS itself. Some of the literature are PaaS provider’s own documents, so source criticism must be taken account when researching those. Much of the knowledge about PaaS platforms are written in web blogs or forum discussions. It is clear that moving to cloud has undisputed benefits, but what about Platform as a Service? For cloud application developer the questions to be asked are: Is PaaS a valid alternative for a cloud computing application project(s) or should the application be built using only infrastructure services? Before deciding whether to deploy PaaS as a strategy or not, it is important to understand the advantages and disadvantages of choosing one.

Also in this chapter, the criteria from which the PaaS options will be measured, are derived and defined. These criteria are selected in a way that it supports the goals of the client and the criteria should be valid also for other cases. The criteria must be selected in a way that the PaaS provider options can be compared rationally with. Still the criteria could be different in different kind of cloud application development. For example, database aspects may not be as important in some other cases. The justification for the criteria selected is strictly based on the previous literature and known reasons.

The criteria has been selected in a general point of view of a small company but also the specific required aspects from the contractor must be taken account of. These aspects mentioned in this chapter are the most important but it is also important to understand why these were selected and some other important aspects ignored. One reason is that there is a limited time frame in which the selection must be made, so there is no time to go through every aspect which might affect the selection further. Also some of these aspects have been a requirement from the client itself but they should also be generally valid. Before the criteria can be defined, the functionality and understanding of the PaaS is critical. (Rglian(a), 2012).

3.1 PaaS advantages

Before jumping straight to some of the PaaS providers' offer, it is obviously good to find out reasons why PaaS is better option than building own platform. It is also good to know the core competence of the development team, if cloud operating or cloud technical knowledge is not the main skill-set of the team, external platform is naturally a good option. (Riglian(b), 2012). The idea for PaaS was enabled a lot earlier already by datacenters (1990s), server virtualization(2000s) and virtualized resources(mid 2000s). (Carlsson, 2012).

According to Information Week’s article(Babcock et al, 2009)., PaaS platforms have brought a new paradigm shift to the world of cloud computing. Without much capital resources, PaaS enables developers and independent software vendors(ISV) to create content and cloud services(SaaS). For that very reason PaaS platforms are particularly
good for small, newly found companies and projects, because in small companies, the revenue must be made positive fast without too much spending. According to cloud computing blogger David Cummings (2012), PaaS is great because it offers easy up and down scaling and it lessens all the tedious system administration tasks from the staff. PaaS platforms offer extensive toolsets for the price (Babcock et al. 2009). PaaS may have different forms and platform sets for social-, enterprise-, database-, and many more frameworks. Particularly web application developers and system integrators benefit from PaaS because of the rich development environments. (Babcock et al., 2009; Cummings, 2012).

Cummings (2012) also states that PaaS offerings like Heroku offer really fast deployment, even if one has limited technical understanding. Deploying content over PaaS is much more simple than deploying dedicated server or traditional cloud instance. Also one good reason is that a developer will not necessarily know the network demand in advance. If the service goes down because the network traffic, potential customers will turn away. It should work also another way, when the demand decreases, startup should not have to pay for unnecessary usage. This is where usage-paid pricing of PaaS comes in. (Armbrust et. al, 2010; Cummings, 2012).

It should be understood, who are the consumers and customers of PaaS. Anyone who are part of the application development are the consumers of the platform such as application developers, who design and build the software, and application testers, who need to run the applications in various testing setups and environments(preferably real cloud environment). Application developers and administrators can also be PaaS users. They may need the ability to configure, monitor and publish the different versions of the application. In PaaS, the development is done using the same standards and technologies as the application will run in production. (Babcock et al., 2009). Application end users can also be thought as PaaS consumers, because they access the SaaS which is deployed over a PaaS cloud. For example their customer and subscription data is located in the database built in the PaaS. (Badger et al., 2011).

The PaaS benefits can also be approached by listing the requirements and additional tasks when not using the PaaS. There are lots of tasks for the developers and technical implementation of the application when not using pre-built platform. First of all, there is the need for acquiring and deploying server for the application. In PaaS the infrastructure management(including servers) is handled by the platform provider. (LongJump, a Service of Relationals, Inc., 2008). Operating systems, run time environments, code repositories and other required middleware are needed which are usually provided by PaaS. Also configuring these takes a lot more time without a provided platform. Moving, copying and testing the code is also more laborious without a platform. (Orlando, 2011.) PaaS also eases the up and down scaling, while more time consuming tasks such as system administration are decreased. (Cummings, 2012). Some PaaS providers such as Cloud Foundry and OpenShaft have merged and offer platform tools to cover the whole application life cycle including aspects outside of development. (Carlsson, 2012).

LongJump (2008) has released a documentation about lowering costs and improving business value with the use of PaaS and aspects mentioned in this document apply for most of the PaaS providers. By introducing PaaS, the company not only add economic benefits but they can also increase value in process- and IT-management. With PaaS, fully utilization of business process modeling, work-flow tools, object/data modeling is made possible easier. (Babcock et. al, 2009). For small companies, the economic reasons can be more important than others. PaaS provides a common infrastructure on a
single platform which is key for creating and running multiple applications and reducing complexity which is an economic reason itself. This also helps the customization and adapting for changing business requirements and needs. LongJump estimates costs in an interesting manner and according to them, application development with PaaS is already cheaper with the first application developed and it gets cheaper each time new application is made (Figure). Deployment of PaaS means that faster development is faster income. For example developers using Force.com built web applications 4.9 times faster than with conventional ways. (Babcock et al., 2009). According to estimation of Forrest Research, 80% of the IT budgets are spent on maintenance and only 20% towards new application projects. (LongJump, a Service of Relationals, Inc., 2008). PaaS provider may also offer a lot of other platform tools which might otherwise cost a lot more. PaaS may have inbuilt platforms for social, enterprise, database, office tool integration and reporting applications. Also, some PaaS providers offer free add-on applications for business process modeling (BPM), customer relationship management and enterprise resource planning. (Babcock et al., 2009).

As stated earlier, testers benefit from PaaS significantly. PaaS enables to deploy as tested, and test as deployed. This means that it is possible to run the application in multiple machines, different configurations and different locations using various kinds of tests for compatibility and performance, which are impossible in a local development environment. Also publishing new releases and betas to real environment is really easy and can be done basically with a push of a button. Also the nature of the development in PaaS setup encourages innovation, because experimenting various aspects is fast and simple. (Babcock et al., 2009).

SearchSIO.com (2011), provides the following business benefits of PaaS deployment:
- **Ubiquitous access**: Resources can be accessed from anywhere when Internet connection is apparent. This is especially important for companies that have sites in different locations.

- **Eliminating upgrades**: Staff does not have to invest upgrading or maintaining operating systems or other platform specific entities.

- **Low capital expenditure**: In PaaS platforms, the revenues are largely project driven so capital needs are easier to manage.

- **Predictable expenditure**: Clients can control the resource utilization so the expenses are more predictable for example in outsourcing cases.

- **Facilitates collaboration**: PaaS platforms allow easier collaboration between organizations, teams and third-party entities. Many different applications can be made over the same platform so interoperability is easier to achieve.

- **Fast implementation**: Without the need of local installation the platform adoption is much faster than in local platforms.

- **Rapid scalability**: Resources can be utilized as needed without delay. (SearchSIO.com, 2011).

Many of these benefits are familiar and affiliated to the general cloud characteristics defined earlier.

### 3.2 Target group of PaaS

To help understand the target group of PaaS, Badger et at(2009) defines PaaS consumers as application developers, application testers, application deployers, application administrators and application end users. Developers, testers, deployers and administrators are the ones who are responsible for designing, implementing, testing and publishing the application for the end user. End users uses the application which is deployed over the PaaS. For them the PaaS is usually completely hidden. (Badger et al, 2009).

According to *Forrester Researcher’s James Staten*(2009), even though PaaS is simpler to use, the consumers are still more interested in IaaS services than PaaS. PaaS is still an attractive option to *true and skillful* coders because it offers possibilities to build more complex applications without the need to worry about the middleware, operating system configurations and the infrastructure. On the other hand, simple application developers such as web page developers don’t necessarily need all the options that PaaS can offer, so they tend to leave it out of the scope of choosing platform options for their applications. In addition, very skillful developers might want to jump straight to IaaS service providers. (Staten, 2011).
The deeper the cloud abstraction goes, the appeal for them decreases, as more and more technical skills are needed. Depending on the PaaS, the users need simple (PaaS such as LongJump) to moderate technical skills (Heroku, GAE). (Staten, 2011).

3.3 Open source in PaaS platforms

Open source enables the developers to modify and share the source code without any charge. In the last few years more and more open source communities are being established all over the Internet. Open source is an adversary for the world of patents and proprietary software and it is based on an idea that knowledge and know-how is distributed freely to anyone who is interested. Open source is already a significant factor in cloud computing and also in PaaS business as platforms such as Eucalyptus and Appscale are widely used. Open source has many advantages. Cost savings are the most obvious one as proprietary software can cost a lot of money. (Bawa & Girdhar, 2012). Also Babcock et al (2009) states that open source PaaS is usually safer option concerning vendor lock-in risks. (Babcock et al, 2009). These are some of the reasons why open source PaaS can be a great alternative for any companies and projects.

According to Brian Sullivan, a president of Sullivan Software Systems, which provides Open Platform as a Service, open source PaaS environment allows the developers to add and move code, functions, applications or systems between different servers and make them work together in one environment. OS PaaS makes it possible for different systems and parts work together. For developers this possibility opens a lot of opportunities because they are not limited or locked in one particular environment of languages. Developers can combine many different programming languages and modules together without the fear of encountering any standardization or legal problems. Still some open source offerings such as Google App Engine are locked for only few particular languages. (Swigart, 2009).
Many PaaS offerings are based on open source software or languages. In addition of Java, Heroku is based on Ruby. AppScale is also based on Python. From the major players only Force.net and Azure are completely based on proprietary software. Many cloud computing providers are increasing their participation in open source and the relevance of open source is increasing all the time. Yet in PaaS business the open source is still scattered and mixed with proprietary technologies and it is still quite hard to stay on tracks which is open and which is not. According to clouds360.com from the 20 top PaaS vendors, only four of them are open source(AppScale, Visual WebGUI, GirdGain, OpenStack). (Clouds360, 2011; Krintz et al, 2012; Leung, 2011).

### 3.4 General PaaS concerns

According to various guidelines (including the NIST recommendations), blogs and journals, there are lots of things to consider before PaaS should be decided to take part of the business strategy. There are many possible obstacles and problems in which the developers and businesses may stumble. Many of these issues are valid for cloud computing in general but there are also lot of PaaS specific ones. These things must be thought seriously before jumping straight to PaaS. Orlando (2011) has also classified desirable features and things to consider for the PaaS. Many of these are also mention in NIST official recommendations and in other references.

According to Orlando (2011), Platform as a Service is often the hardest to understand service model of cloud computing and often mistaken as IaaS or SaaS. The PaaS providers themselves usually just advertise their products in a way that they don’t concern about the possible limitations and risks, so caution must be practised when researching their offerings. Many PaaS providers are set apart from each other by the solution stack, which contains a set of different applications and tools to assist developers and application development (Figure 12). According to Babcock et al (2009), PaaS provider might go out of business, which means that services can drop as fast as overnight. There is also a risk of service price increase in a short period of time or quality of service drop-down. Also, the frameworks of different platform services usually have their own interfaces, methods and costs. This must be taken account if thinking about moving to a different platform. Babcock also states that, PaaS does not necessarily offer increased flexibility, instead it gives power and ready-made services. There are also limitations in the form of time intervals of the requested messages. Even if tasks are executed quickly in local environment, the case might be different on-line and if the tasks are not executed within the specified time interval, there might be problems. (Babcock et al, 2009). Orlando (2011) also has a set of questions in which the developers or companies might want to answer, other references also take some of these aspects account. These are described along with the criteria. (Orlando, 2011).

### 3.5 Scalability

Scalability is the first criteria from which the PaaS options will be measured. PaaS should be intelligent enough to provide proper load handling to the underlying infrastructure and to handle the loads the application causes. (Mell & Grance, 2011; Orlando, 2011). The platform should have fast scaling: It should always measure the usage needs and provide resources as needed. It should work also in another way: It should free resources when not needed. It should also be noted how the scaling is handled and how flexible it is. This aspect may also have immediate impact to costs, because some platforms are charged by usage. (Järvi, et. al, 2011).
Auto-scaling is one of the most important feature of PaaS. To achieve real-time demand scaling, platform provider is required to balance traffic across multiple servers, monitor the load on each server and start new servers when needed. Measuring the quality of auto-scaling is hard because all PaaS providers offers some kind of auto-scaling system. Database systems, programming paradigms and tools for different PaaS providers may also vary. Also database usage might be different when PaaS offers centralized database, compared to a situation where database is located on the same server as the application. (Yuan(a), 2011).

The provided platform must be scalable because the service must be ready if customers start to pour in rapidly, but also to scale down in low usage situations if the platform is charged by usage. PaaS should be intelligent enough to provide proper load handling to the underlying infrastructure and to handle the loads the application causes. (Orlando, 2011). In addition a sensible solution for database scalability would be a desired(although uncommon feature), since the contractor’s application is based heavily on databases.

There are many advantages to be considered when measuring scalability. Yuan(2011) presents many factors that can be compared concerning scalability handling in PaaS. To the final analysis, the following important factors are measured to find out the possible advantages. These factors are summarized in table 1.

The task of load balancer is to distribute the load to workers or nodes. Dynamic load balancing means that these workers and nodes can be assigned at runtime according to the load balancing policy provided. Usually the worker nodes have identical hardware and software resources and configuration. Most of the PaaS providers have built-in dynamic load balancing system so it’s not one of the most important factors to considered. With load balancers there are usually features such as session affinity and elastic load balancer(ELB). Session affinity means that if session ID is sent, the load balancer will send all the requests from the same session ID to the same workers and nodes. This does not usually work well when large scalability is needed. Elastic load balancing means that the balancer monitors the nodes and workers and start up new ones if more capacity is needed. (Azeez, 2011). Concerning the analysis, the presence of built-in load balancer is a possible advantage.

At default each load balancing instance that is created, a unique canonical Domain Name System (DNS) name is reclaimed. Many PaaS providers offer also a feature that lets the user to create custom DNS names(for example Amazon)to associate with the load balancers. This feature is good for tracking the load balancing. (Amazon Web Services(e), 2012). The possibility for custom domain configurations for load-balancers is a possible advantage for the PaaS.

Auto-scaling means that the resources are scaled up and down automatically when they are needed at any given time. The user don’t have to worry about the scaling at all. If the applications needs more resources, auto-scaling of application server handles the resource distribution. It is equally useful for either low or high traffic(saving costs). Popular ways of doing the auto scaling are Front-end Site Traffic(scale based on the number of incoming requests) and Back-end Batch Processing(scale based on the size of the job queue). (RightScale Inc, 2012). The ability to auto-scale the app server is a possible advantage.

Auto scaling the database is not a simple task. Not many if any PaaS providers offer the feature. Database can be scaled for throughput and size. Proper database scaling means
no database downtime and the database is transparent to the application so now code changes are needed. According to Haber, scaling the database is a major challenge in dynamic cloud environments. The biggest challenge comes from the fact that databases are state based systems and cloud has a natural stateless environment. So far no PaaS providers offer proper auto-scaling of relational database but for example Xeround offers a database scaling add-on for Heroku, provided that the databases are in Xeround’s service. Xeround can also be integrated to other PaaS services.(Haber, 2011; Kapuya, 2011). Google App engine offers database scaling but GAE does not support relational databases at all. (Yuan(a), 2011). The possibility to auto-scale the database is a possible advantage.

HTTP session management is important and challenging, especially in Java-development, because the session state of HTTP is managed in memory by default. Normally the developer must configure the load balancer to monitor the session ID of all incoming requests and direct the requests to the server behind the load balancer. This may cause problems in the load distribution over time, as the load balancing may become unbalanced, since each server will contain some sessions (“sticky sessions”). Developers must also set up a cache for the memory and this may be cumbersome for the bandwidth and the CPU. All HTTP sessions can also be stored into the database. Not many PaaS providers offer a good way to deal with these problems. Google App Engine and Heroku offer their solutions but many others forces the responsibility to the developers and offer solutions like “sticky sessions” which aren’t necessary the ideal solution. (Yuan(a), 2011). The possibility of session management is a possible advantage for the PaaS.

<table>
<thead>
<tr>
<th>Scalability criterion</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>built-in load balancer</td>
<td>The presence of built-in load balancer brings flexibility and eases the development.</td>
</tr>
<tr>
<td>Custom domain possibility for load-balancer</td>
<td>The ability to configure custom domains for load balancers helps the tracking of load balancing.</td>
</tr>
<tr>
<td>Auto-scaling of application server</td>
<td>Auto-scaling allows the users not to worry about scaling.</td>
</tr>
<tr>
<td>Auto-scaling of database</td>
<td>Auto-scaling allows the users not to worry about scaling.</td>
</tr>
<tr>
<td>Clustered and automated HTTP session</td>
<td>Helps load balancing and distribution problems.</td>
</tr>
<tr>
<td>management</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Scalability criterion factors

These are the factors in scalability criterion that the PaaS alternatives will be measured against. Clustered and automated HTTP session management can be thought as part of built-in load balancer so things like “sticky sessions” can be mentioned when reviewing the built-in load balancer. Concerning the contractor's case, the most important factors are everything that are affiliated with reducing of work load of a development team (auto-scaling, load balancing), since there are staff limitations. Also tracking the load-balancers are important because they may have connection to overall costs (risk of using unnecessary resources).
3.6 Service agreements and vendor lock-in

One of the most crucial aspects of the selected PaaS is that it is not locked to any particular infrastructure. There should be possibility to change the infrastructure layer if necessary. Some providers such as Microsoft lock their customers to their own infrastructure. (Babcock et al, 2009). Vendor lock-in means that a product or a service vendor has made it difficult or even impossible to transfer to another offering. This means that the customers can be “trapped” to the service. (Rouse(b), 2012). When customer is locked in certain provider, it also may bring problems like sudden increase of costs (PaaS provider increases prices) and technology changes which may have catastrophic consequences: rewriting of the application. (Babcock et al, 2009; Monroy(b), 2012; Pradeeban, 2011). Generally, concerning vendor-lock in, instead of developing over an infrastructure, choosing PaaS is better. Some PaaS providers force the customer to certain IaaS which can arise many problems. Staten(2011) states that the main concern about choosing PaaS is the vendor lock in. (Staten, 2011). Vendor lock-in can also be a problem when trying to move to a different PaaS because the standardization is lacking. (Heiska, 2012).

Also possible data locations are tightly locked into the possible infrastructure vendor lock-in. When selecting PaaS, it is important to know how data sharing, location and responsibilities impacts the decision. Possible data locations must be thought when choosing the platform provider because some customers may have their own rules concerning the data location. (Mell & Grance, 2011). In the contractor's case, the requirement is that the data must be located at least inside the EU and if possible, preferably in Finland. Each consumer who is thinking about PaaS subscription, should think about (among other things) data location configuration. (Mell & Grance, 2011). For example European Union requires the company to have a responsibility to ensure that any data stored has suitable security measures. European companies can store their data in U.S only if the third party meets the EU privacy protection standards. That is why security and data protection measures must be well known in PaaS environments. If inadequate care is taken, the company using certain PaaS services may lead to law violations. (Cobb, 2008). Also things like amount of application limitations and server unavailability guarantees are advantages or disadvantages to be taken account. (Haddad, 2011).

Possible issues such as: What could happen if the possible provider bankrupts or abandons their platform services all together and what will happen to critical customer data, e-mails etc. when the service is shut down. In the end, service availability is ultimately the service providers and PaaS customer’s responsibility. The company who has the responsibility of someone else’s data storing must guarantee the server up-time. 99,9% up time should the target of any online service. Even then the possible 0,1% downtime is almost half a day in a year. Besides downtime, performance is another issue which may cause data processing unavailability. Cobb states that PaaS providers probably deliver better up-times than other organizations can. On the downside service level agreements must be understood more in PaaS environments. (Cobb, 2008). The service agreements must also be read carefully for example for a situation where PaaS provider bankrupts and to avoid possible vendor-lock in these situations. (Martin, 2010; Monroy(a), 2012). It is recommended to select a well-known and financially stable provider and design the software in a way that it is not too dependent on the platform. (Riglian(a), 2012). One possible issue or “trap” comes from the fact that some providers offer a lot of useful integrated and third party add-ons. Developers may become
dependent on these services, hence introducing another vendor lock-in risk. Add-ons should be selected carefully. (Monroy(a), 2012).

The lack of proper standardization is a big risk concerning lock-in within the PaaS providers and cloud computing in general. Because of this, the comparability and moving between the platforms can be difficult. Because open standards are lacking, **locking** to a particular infrastructure or a platform is not a desirable option. According to a survey presented in Heiska’s thesis, 80% of the respondents thought that the lack of standardization is moderate to a severe reason to cancel the move to another platform. (Heiska, 2012).

Some kind of exit strategy should be maintained if the application is successful enough and “grows” too big for the current platform. If the application grows too big, it may be necessary to exit the PaaS strategy and go straight for infrastructure providers. For these cases it is important that the PaaS applications can be migrated off the platform. These things should also be secured in the code level (dependencies, environment configurations, etc.). (Monroy(a), 2012).

Some points of views suggest that lock-in is caused by the data itself, so the vendor lock-in in platforms becomes obsolete. If there are very large amounts of data, moving them take so much time that it is not worthwhile anymore so locked in a vendor is not the issue anymore. There is also other concerns such as proprietary APIs which have already been addressed companies such as Heroku, which uses open standards and conventions. The proximity of third-party network services may also lock the application in such a way that it can not be moved anywhere. (Kim, 2012).

According to NIST, the portability becomes an issue when platform require proprietary languages and run-time environments. Many PaaS providers have different platform implementations and the interfaces used are not always compatible with each other. For consumers, there is an option for generalization of interfaces. However this can increase the costs and limit the PaaS provider’s enabled value. (Mell & Grance, 2011). According to Järvi et al (2009) even thought the application is built using the techniques that work with other platforms as well, the problem with PaaS providers is that the infrastructure abstraction is many times hidden and moving the application from one platform to another demands rewriting of code. (Järvi et. Al, 2009; Orlando, 2011).

Four advantage factors are chosen to the analysis: the presence of vendor lock-in and application lock, amount of application limitations, ease of portability and server locations. These are summarized in table 2.

Most preferable situation would be that the customer can choose their infrastructure themselves and PaaS can be applied over it. Any possible application locks must also be studied carefully (license agreements, proprietary languages, etc.) to find out, if the PaaS provider locks the application in their platform. This also depends on the application developed so developers have also responsibility concerning the lock-in. Some of the technologies used might be dependent on the PaaS providers platform. It is also mentioned that developers may run in to lock-in unaware and unintended if the cloud platform is different than expected. The written code has an effect how it will run in a particular PaaS. Some platforms are more restrictive than others and these aspects must be taken account of. (Mann & Vaughan, 2012). Martin (2010) states that data mobility and ownership aspects are one of the most important concerns to worry about in PaaS. It is important to know can the users move their data and applications to another service provider if needed. (Martin, 2010). Possible advantages or disadvantages comes from
the presence of infrastructure lock and/or if the applications are locked to the PaaS itself.

There can also be some limitations concerning the number of applications that can be developed. Many providers limit the amount of applications, the consumer can develop on particular PaaS. It is important to know that the selected PaaS meets the needs. (Orlando, 2011). The ideal situation would be that the platform does not have amount of application limitations but this is usually possible only in the most expensive options. The most important aspect is that the PaaS should not have to be discarded too soon because of the application limitation. Usually this is not issue but in the most inexpensive options, this can be a notable issue. Generally speaking the more applications the platform supports, the better. (Haddad, 2011).

Portability can be measured by finding out how easy it is to move applications to another platform or to another environment. Best situation would be that no code changes are needed in the application code when it is moved elsewhere. According to Orlando(2011) the platform should be inclusive and portable. The platform should have the ability to have multiple customer applications functioning on the same platform. The application must be able to be moved to another IaaS. PaaS should handle all the possible infrastructure level problems and SaaS developers should not have to concern with infrastructure level coding themselves. Some PaaS providers rely completely on some particular IaaS provider. For example, Microsoft Azure rely completely on their own infrastructure. (Orlando, 2011).

The underlying infrastructure determines the server locations. Because of the critical data the provided service saves, the data servers should be located in Finland and at least inside the EU. For example Amazon offers locations all over the world(S3 servers) and if the PaaS provider uses Amazon’s infrastructure(Amazon Web Services(e), 2013), the defined server locations should be studied. For example if the cloud service is located in Finland the Finnish data protection laws must be obeyed. If data center is located inside the EU, the laws of the country in which the server is located is followed. (Järvinen, 2011).

<table>
<thead>
<tr>
<th>Service agreements criterion</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>The presence of vendor lock-in</td>
<td>The PaaS may be locked to a certain infrastructure. The users may be locked to the PaaS. These may disable the data and application portability completely.</td>
</tr>
<tr>
<td>Amount of applications</td>
<td>The amount of applications the PaaS allows the users to develop over the platform. Generally, more the better.</td>
</tr>
<tr>
<td>Portability of application</td>
<td>How easy the developed applications are to move to a different platform or infrastructure.</td>
</tr>
<tr>
<td>Data locations</td>
<td>Preferably, the user can decide the geographic locations, where their data is stored.</td>
</tr>
</tbody>
</table>

Table 2. Service agreements and vendor lock-in criterion factors

These are the factors in scalability criterion that the PaaS alternatives will be measured against. Concerning the contractor's case the most important factors to consider in this
criteria is the vendor lock-in since it was discussed that there should be a possibility to change the PaaS vendor if necessary, so portability aspects are important. Also data locations are important since the data stored in the cloud may be forced by a law. They must be at least inside the EU or even inside Finland. The deciding of data locations would be a desired feature.

3.7 Security

There are a lot of aspects regarding security in cloud environments and the PaaS selected should be reliable especially in a case where the customer’s data is critical. Lacking security might be the breaking point of a software service provider. In 2009 security reasons were the most common reason why companies didn’t want to move to cloud services. Privacy and availability, both affiliated to security, were one of the most common concerns. People are worried about how much personal data is collected and how this data is used to possible harm the user. Availability was considered also a security concern. People are worried about the possible service shutdown: What happens to the critical data if the service goes down? Can the data be lost if the provider bankrupts? (Martin, 2010). Also in recent polls, security related problems are still the most common concerns in cloud services (figure 20). (Emison Masters, 2013). In the contractor’s case, security can be considered one of the most critical aspect of this criteria and the security features of the platform must be studied carefully.

![Cloud Service Concerns](image)

**Figure 17.** Cloud service concerns (Emison Masters, 2013).

Generally customers share of responsibility gets smaller as the stack goes up. Responsibility is greatest in the IaaS layer. Any security concerns over virtual machine(VM) layer such as passwords, versions and patching are on the hands of the customer. When moving up to PaaS and SaaS models, the security responsibility moves more and more from the customer to the provider. (Corrington, 2012). Still with PaaS,
there are many security issues to be investigated. Even though many PaaS providers offer robust security aids, in the end the responsibility is on the shoulders of the developers and their solutions. Cloud service provider is responsible for network and application layer firewalls, IDS/IPS blocking, DDoS (Denial-of-service) attack blocking and maintaining and repairing security vulnerabilities. Even if PaaS provider has these responsibilities, SaaS provider has them also. (Huotari, 2012). That said, security can be a “game breaker” for a web service. For example if customer’s credit card information are stolen, it can be assumed that the customer will never use the service again. Especially today (with social networking), poor security and news about security breaches spread fast over the community. In the software development point-of-view the main concern should be that what security responsibilities are handled by the platform itself and what should be handled by the developers them selves.

The NIST documentation (2011) has extensive recommendations concerning security when choosing PaaS and also general concerns in cloud computing which must be taken account when choosing the PaaS. Unlike the isolated local resource using applications, PaaS applications access network itself, so the security must be taken more seriously. The PaaS should have tools to configure the application to run securely. Client-server communications should be encrypted, for example by having an own dedicated service for the encryption. There should also be integrated frameworks for identification, authorization, access control and a protection of customer’s data from another customer (multi-tenancy). (Jansen & Grance, 2011). There are several points in the interfaces that must also be addressed. Even if the PaaS offers basic solutions for these concerns, the developers must address issues such as identify management, user authorization methods and user access rights. There should also be guarantee that data can be reliably deleted by customer’s request. Logging, alert systems and reporting for activity and data transactions should be monitored and it is important that the selected PaaS has tools for these operations, otherwise they might be overlooked. Also, even if the PaaS has encryption and other basic security methods, the security requirements should be checked that they are met and if necessary enhance the security some other way. (Corrington, 2012).

The increased security is one of the benefits of choosing PaaS as a strategy. Still PaaS brings some additional security concerns and responsibilities, especially for the developers. Many of the same general cloud computing security concerns that apply in IaaS and SaaS services, also apply in PaaS solutions. (Cobb, 2008). The security concerns must be taken account in every phase of the development. PaaS (and SaaS) customers move their data outside of their own firewall safety and chances of severe security breach increases as hackers today focus more on tempering the data. Sometimes the SaaS customers themselves may want to test the provider’s ability to withstand possible attacks by having their own testers to try and penetrate the security barriers. The provider must also fix, document and test possible security flaws in their system. Some customers may be scared to put their data into public cloud. This may bring business risks and lost customers to the service provider. (Stafford, 2012).

Some analysts believe that public cloud is safer option, because it is more unlikely to get hacked or robbed of particular data, because there are more data in the public cloud and it is harder for hackers to find particular customer’s data in the public cloud (analogy: It is harder to get robbed in a baseball match than inside one’s own home). (Stafford, 2012). Still, according to ActiveState (2011), a good way to increase security is to choose private PaaS. It enables all benefits from public PaaS but provides elevated security. Private PaaS is particularly good for organizations which demand privacy such as healthcare and financial records. Some customers even require extended
privacy for their data. Private PaaS keeps the data within the consumer. A private PaaS is exclusive for the consumer’s organization and it is more secure from vulnerable attacks. (ActiveState(a), 2011). It is also mentioned that some organizations have chosen private PaaS as a mean to standardize the application infrastructure. Private PaaS also helps to gain flexibility and agility without compromising the security and performance characteristics. (Watson, 2012).

The following factors are measured for the PaaS comparison analysis (summarized in table 3): the possible presence of SSL encryption, possibility for private or hybrid cloud configurations and the general extensiveness of authentication systems.

When choosing PaaS it’s important to know the platforms ability to handle security concerns such as encryption, data availability, authentication systems and other usual developer’s security responsibilities such as framework, code repositories, bug database, etc. (Cobb, 2008). Providing SSL encryption is a good way to handle these concerns. (Orlando, 2011). SSL (Secure Sockets Layer) is one of the most important security attributes that web service must have and it is quick, easy and cost-effective to enable. SSL is a small piece of code that can be installed over a web site or a service. SSL encrypts the information between trusted participants (server-domain) and it is effective for example “eavesdroppers” of sensitive data and communications. SSL can contain encryption levels up to 256-bit which is virtually impossible to decrypt with conventional means. SSL certificates also function as an authentication of web sites and services. The SSL provider requires that the web site is trusted and secure before offering the SSL service. SSL certificates can also be self-signed but trusted third party are more secure. Every web site or a service which contain any critical data and data transfer should be SSL secured. To recognize SSL secured service, SSL web addresses starts with https://. SSL secured web addresses also give a message that the provider cares about security and their service is secured. (GeoTrust, Inc., 2011) Concerning security, SSL (Secure Socket Layer) or lack of it is really important factor. If SSL is not supported the security is severely compromised. (Orlando, 2011).

Usually private cloud scenarios are valid options in infrastructure (IaaS) level development. Some PaaS providers offer private PaaS (PaaS in private cloud) which elevates the security and adds privacy for delicate data. In private cloud scenarios the customer has the control over it. (Badger et al, 2011). Public PaaS may have security issues because of multi-tenancy. There may be many customers in the same public cloud where the platform is also located. If public cloud gets successfully attacked, everything inside the cloud may be compromised. (SearchSIO, 2011). Possibility to run PaaS in private cloud is particularly good if organization uses already private cloud in their business. If the user wants more flexibility and availability, hybrid cloud is a good compromise without losing much of the security aspects. (ActiveState(a), 2011).

The objective of authentication frameworks is to make sure that users can be identified correctly without the system to be vulnerable to security attacks and identity theft. The nature of the possible attacks are usually: impersonating, social engineering, brute force or password resetting. When measuring authentication framework in PaaS platforms, operational standards such as password complexity requirements, password change forcing timescale, password compromise prevention and the presence of security standards can be reviewed. (Shinder, 2012). Authentication, access and authorization controls are important for the PaaS since usual security attack schemes include impersonation, social engineering, brute force and password reset attacks. Most PaaS providers rely on simple user name and password authentication which can be strengthen for example by “secret question”. It would be preferable that the platform
offers some kind of known method for authentication. For example Microsoft uses *roles-based framework*. (McPherson, 2004; Shinder, 2012).

<table>
<thead>
<tr>
<th>Security criterion</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>The presence of SSL encryption</td>
<td>SSL is crucial for any web service which handles crucial data.</td>
</tr>
<tr>
<td>The possibility for private or hybrid cloud configurations</td>
<td>Private cloud increases security and privacy for data. Hybrid cloud is a good compromise if there are a mix of different cloud configurations within the organization.</td>
</tr>
<tr>
<td>The extensiveness of authentication systems</td>
<td>Extensiveness can be measured by investigating the known security methods and certifications, the PaaS has.</td>
</tr>
</tbody>
</table>

**Table 3. Security criterion factors**

All of these factors should cover and are affiliated to aspects like privacy, usual security attack schemes (brute force, DDoS, etc.) and social engineering. In the contractor's case, each security aspect is important since the data stored in the service may be crucial. In-built SSL encryption is important, since SSL would be used even if the PaaS didn't have built-in SSL encryption. The staff also took a security training concerning these factors (especially risks concerning authentication schemes) discussed in this chapter. There were discussion about the cloud configurations such as would it be necessary to store the data in private environment but nothing concrete were decided but the possibility to private cloud configurations would be nice.

### 3.8 Framework and supported programming languages

The first thing when surveying PaaS alternatives is to decide for what kind of framework and for what programming languages the application is going to be implemented. It should be known that what languages and technologies the development team already know and match the PaaS selection based on that. According to Yuan(2011), the most important aspect are the framework stacks, the platform supports. At least for Java PaaS vendors, he believes that supporting as many stacks as possible is very important. (Yuan(a), 2011). Orlando(2011) states that, ideally a PaaS should support any frameworks that are based on the language of choice for that platform, so it is important to find out what frameworks and languages does the PaaS support. Also the PaaS provider's database solutions must be compatible with the consumer's database. (Orlando, 2011). NIST(Mell & Grance, 2011) recommends that PaaS should contain only standardized languages and tools (and data access protocols like SQL). Proprietary languages and tools should only be used if they are the only practical options. (Mell & Grance, 2011).

Different PaaS offerings may have different kind of approach to the application development and application life cycle. Some offerings are focused on deployment and others may be leaned more on the run-time and application management side of development. It should be clarified that should the PaaS have more support on developing completely new applications from scratch or running existing applications better on the cloud. (Riglian(b), 2012). From the business point of view it is important to ensure that flexible, elastic and portable cloud environment is built. This means that it
is important to study, should the development team adopt programming language-specific platform, multilingual platform or a hybrid that supports development and deployment over multiple cloud environments. Multilingual platform offers more flexibility and enables to work with multiple different frameworks which can be changed depending on the case. Downside is that these offerings do not usually offer the same level of support, libraries, APIs or configurability than language and platform specific offerings. (Sullivan, 2012).

Framework should be studied and it’s features should be clarified. General cloud characteristics: On-demand self-service, Resource pooling, Rapid elasticity and Pricing strategies must be sufficient enough. Development concepts such as continuous integration, delivery, release management and testing, should also be studied by inspecting the middleware components and services. (Haddad, 2011). The platform should have a robust, well known application development framework. According to PCWorld(2010), tool robustness and tool efficiency is one of the most common concerns in PaaS. (Martin, 2010). Ease of use is also affiliated to this concern. It is desirable that PaaS is easy enough to use. WYSIWYG interface(Dictionary.com, 2012) with pre-built widgets and UI elements is ideal to help the application development to be rapid and iterative. Web-based monitoring services are also good for monitoring the scaling and other platform and cloud specific features. (Mell & Grance, 2011).

The platform should contain all the tools which are enough for the business and application development. For example PaaS should have tools for migrating the data in and out the cloud. (Orlando, 2011). Framework also affiliates with vendor lock-in. The interfaces, methods and costs should also be known. NIST recommends that well documented and flexible API for data such as user authentication, assets and database calls should be offered right out of the box. Well constructed API allows more flexibility and software customization to meet the needs of the application. NIST recommends that the interfaces should be generic enough to support application portability and interoperability. (Mell & Grance, 2011).

The factors to be measured for the PaaS comparison analysis(table 4) are the programming language offerings, programming framework stack extent, database framework extent, the presence of graphical user interfaces and web-based monitoring, and the presence of testing and/or build environments.

More supported languages and frameworks bring more flexibility and configurability. Based on the contractor’s case one or more of the following languages should be supported: Java(SE/EE), Python, MySQL, SQLite and PostgreSQL. The requirements here are affiliated to the programming languages. In the contractor’s case, the PaaS should offer frameworks for Java(EE, Play, REST, etc.) and/or Python(Django). (Yuan(a), 2011). The PaaS should support those database solutions which are selected and sensible to a particular application. More supported database solutions bring more flexibility and configurability. This factor is also affiliated to the frameworks. (Orlando, 2011; Yuan(a), 2011)

At ideal situation, the PaaS has both, inbuilt graphical interface for the various features and options and command line tool set for more advanced configurations and tweaking. Also a web-based GUI monitoring is desirable feature for the PaaS. Many PaaS solutions such as ActiveState’s Stackato offers extensive web tools and GUI for monitoring scaling and other purposes. (ActiveState(b), 2011; Yuan(a), 2011). A robust inbuilt test framework would be desirable feature. Ideally the platform has an integrated build and test environment. This would enable the developers to continuously develop,
build and test code in real environment. Continuous integration with developing teams is one of the keys in robust software development process. If the platform has proper testing and building environment separate build management is not needed and the software processes are much more visible to the developers. (Yuan(a), 2011).

<table>
<thead>
<tr>
<th>Framework criterion</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>The extensiveness of supported programming languages</td>
<td>More supported programming languages bring more flexibility and options for the developers.</td>
</tr>
<tr>
<td>The extensiveness of supported framework stacks</td>
<td>More software frameworks the PaaS support, the more flexibility it gives for the developers.</td>
</tr>
<tr>
<td>The extensiveness of supported database frameworks</td>
<td>More database frameworks the PaaS support, the more flexibility it gives for the developers.</td>
</tr>
<tr>
<td>User interface tool set</td>
<td>GUI and command-line interfaces are desirable for the PaaS.</td>
</tr>
<tr>
<td>In-built testing and build environment</td>
<td>Helps integration, testing and software development processing.</td>
</tr>
<tr>
<td>Web-based monitoring</td>
<td>Web-based GUI helps the monitoring of application development.</td>
</tr>
</tbody>
</table>

Table 4. Framework criterion factors

These factors should cover the framework extensiveness of the PaaS alternatives enough. In the contractor's case these factors were not discussed as much as the others. The framework was selected with the skills of the development team in mind. The database framework should preferably support any relational approach. Testing aspects were discussed and it was clear that testing should be fast, constant and in the real environment as much as possible.

3.9 Pricing

While the pricing is naturally one of the most obvious factors for a small company and small projects, there are still more factors and advantages to inspect than just the price. It should also be known what kind of pricing methods the platform provider has such as pay-as-you-go or fixed costs. The goal is not to find some fixed cost limits but instead, find the methods of pricing and more long term estimation about the costs.

In NIST official recommendations, measured service and pay per use are one of the standard cloud characteristics. (Badger et al, 2009). In pay-as-you-go pricing model, the usage is monitored and the system generates the bills based on the pricing model. In the end, the bills should cover the business value first, not just the IT resources themselves. Business users of the particular organization can not calculate the platform value easily from the CPU time, consumed network resources or data storage, if they do not have enough technical expertise. With co-operation with the IT staff, the value can be derived for example from the number of users, processed forms, received market pieces and sales transactions. Ideally the platform should also have a support for monitoring, metering and billing, based on business oriented entities. (Haddad, 2011).
When measuring pricing as a criteria, it is not about which one is cheaper but which one offers best and most efficient cost structure. It is important that the data management service on top of the platform has efficient cost structure as well. Many PaaS providers don’t offer some database solutions at all. Google App Engine for example does not offer relational database frameworks because their cost structure is not suitable. (Google Developers, 2012; Google Inc(c), 2013; Kapuya, 2011). According to Armburst et al, pay-per-usage pricing model is best for new and small companies because the demand is unknown in advance. When the service becomes popular it may have a large spike on demand and after the novelty ends, the demand drops. There is also an option for usage-based pricing which means that the customer can buy resources beforehand for example for 1000 hours. This means that these hours can be divided for any period of time, but this pricing model is only good for cases where the demand is well-known. Usually the usage fees are calculated from the consumer types (developers vs. application end users), storage, processing, network resources consumed, requests and the usage time of the platform. (Armburst et al, 2009; Badger et al, 2009).

Many PaaS providers offer free entries to their services. For example Heroku gives 750 free dyno hours when registered. (Henry et al, 2012) Free-tiers and how much can the customer do with them should also be studied. (Yuan(a), 2011) After the free-tier is consumed, there may be dramatic cost increase when the applications scale the PaaS. At some point there may be the need to calculate whether it is cheaper to host in-house cloud with additional staff to admin the cloud. It should be estimated what are the PaaS costs in a year, if the application uses more and more PaaS resources. At some point, there may come a situation that paying for PaaS services is not sensible anymore. (Monroy(a), 2012)

The factors for the PaaS comparison analysis (table 5) in pricing criteria are, the free-entry offerings, pricing model extensiveness and the estimated entry level web application costs. Free entries can be measured by researching the PaaS provider’s entry offerings. How much does the PaaS offer for free (data storages, instances, load balancing, etc.) and how long? Concerning the pricing models, preferably the customers can choose what kind of pricing model they want to use. Pay-per-usage and usage-based models should be possible. Costs for web applications are measured by studying the free entries and basic costs of entry level needs for a relatively simple web application.

<table>
<thead>
<tr>
<th>Pricing criterion</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extensiveness of free-entries</td>
<td>This factor is studied by investigating PaaS provider's free entries.</td>
</tr>
<tr>
<td>Extensiveness of pricing models</td>
<td>Pay-per-usage and usage-based models should preferably be offered and decided by the customer.</td>
</tr>
<tr>
<td>Costs for entry level applications</td>
<td>This factor is studied by researching free entries and costs for entry level services for entry level application development.</td>
</tr>
</tbody>
</table>

Table 5. Pricing criterion factors

These are the most important factors for pricing criterion. Pricing is naturally a big concern in the contractor's case since it is a small company with limited financial resources. Extensive free-entry would be naturally a very important feature. Also costs for entry level applications should be studied very carefully. The pricing model type
should be able to be selected by the customer. Also the predictability of the costs must be high. It is also important to estimate the possible future costs when the resource need is higher by studying the instance prices of the PaaS providers.

3.10 Other aspects

In addition to the aspects above, there are aspects such as ease of use, provided support and portability. In this time frame the usability aspect is not as important as some of the others. Of course the PaaS should be easy enough to use and preferably contain a robust UI. This has been thought when selecting the PaaS alternatives and these aspects are measured when measuring the framework advantages. Support is also an important aspect but again, it is not considered as important as some of the others and support is usually tied to the pricing. The more expensive alternatives have more extensive support systems in various price categories. Portability can be a critical aspect but it is not that until later, when multiple applications and larger user group are present and when platform transfer may be needed. Portability aspects such as vendor lock-in and platform migration can also be considered as part of the service agreements.
4. Selected PaaS Alternatives

In this chapter, the selected Platform as a Service (PaaS) options are reviewed based on the criteria which were defined earlier. After researching and studying various potential PaaS alternatives the following were selected for a review: Amazon Elastic Compute Cloud (Amazon EC2), Google App Engine (GAE), Heroku and Cloud Foundry. It is also described why the particular alternatives were selected from very large array of providers. Each alternative is reviewed against the criteria (scaling, service agreements and vendor lock-in, security, supported programming language frameworks and pricing).

4.1 Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Beanstalk

Amazon Elastic Compute Cloud more commonly known as Amazon EC2 is a web service that provides scalable infrastructure and platform for building and hosting software and applications. Customer of EC2 pays only for the capacity that is needed. EC2 provides a web-based GUI and command line tools. All the needed hardware and resources are located in the cloud provided by Amazon. Amazon also offers the customers to choose the location of where their application is deployed geographically. Amazon offers deployment in the United States (North Virginia, Oregon, North California), Europe (Ireland), South America (Sao Paulo) or Asia (Singapore, Tokyo). Amazon EC2 beta was announced on August 2006 and production started on October 2008. (Amazon Web Services(e), 2012; Barr, 2006; LaMonica, 2008;). Amazon has a wide variety of cloud computing services and choosing the right ones is a challenge without a proper study of the offerings. For example Amazon CloudFormation can be a great addition if EC2 services are used. Amazon Beanstalk is the PaaS add-on for EC2 and together with other EC2 services a complex and feature rich platform can be built. (Amazon Web Services(a),(b) 2012).

4.1.1 Framework and main components

Amazon EC2 is extensive service and it consist of many different components and sub-services. The main component is AMI (Amazon Machine Image). AMI can contain an operating system, servers and applications. Instances can be made of AMI. Instances are copies of AMI and function as virtual servers in the cloud. Instances can be accessed from client’s host computers. There are many different AMIs available with different configurations for various scenarios. AMIs can also be configured and customized by the customer itself. (Amazon Web Services(e), 2012).

Amazon has servers and data centers in North America, Europe, South America and Asia. Instances can be located closer to customers location which can be crucial for example in legal and other requirements. Amazon offers three storage options: Amazon Elastic Block Store (Amazon EBS), Amazon EC2 Instance Store and Amazon Simple Storage Service (Amazon S3). (Amazon Web Services(e), 2012).
Amazon EBS is usually the most sensible alternative because it consist of hard disks in which the instances can be attached. It also functions as a backup storage in a way, that a snapshot of EBS storage can be taken and saved in Amazon S3. Instances can be stored in the Instance Store. Instance Store can be used to store non-important and non-persistence data. The data can be deleted if the instance is terminated. Amazon S3 is an Internet storage which provides a simple web interface. It allows the consumer to store and retrieve any amount of data from anywhere. The root device storage stores the images which are used to boot the instances. (Amazon Web Services(e), 2012).

Amazon EC2 offers two ways of database solutions: Amazon Relational Database Server(Amazon RDS), which is a relational database in the cloud. RDS offers options such as software patching, backup handling and backup storage. There is also an option to use a database AMI in which the EC2 instance is used as a database. (Amazon Web Services(e), 2012).

Networking and security aspects are configured for each instance. A public IP address is assigned to each instance in the EC2 network. If instance fails a replacement instance can be launched with different IP address. There is also an option to use elastic static IP addresses. Security groups can be used to control the access rights to the instances. For example only a specific IP addresses can be assigned to have control to the specific instance. Security groups enable the control of protocols, ports and IP ranges to be assigned. Different rule sets can be assigned to different security groups and each instance can be assigned to a single or more security groups. (Amazon Web Services(e), 2012).
Amazon Web Service also offers monitoring the statistics for instances and EBS volumes, automatic user predefined capacity scaling and automatic distribution of incoming traffic across multiple instances. In addition, Amazon EC2 has AWS Identify and Access Management (IAM) service integrated in it. This allows control over user and group creation, sharing the resources between the users in the account, assign security properties and user access control for each user and AWS billing for all users under the account. (Amazon Web Services(e), 2012).

To manage, configure and create all the components and AWS resources, CloudFormation service can be used. It functions in a way, that the user creates his/her own templates to describe the AWS resources, runtime parameters and dependencies he/she wants and needs to use for the application. CloudFormation takes care of provisioning of services and dependencies even if the user do not figure them out him/herself. CloudFormation allows modifying and updating the Amazon resources in a controlled and predictable way that supports version control way of thinking, the same way the developers think about it in software development. In CloudFormation, templates controls the resource stacks via command line tools and APIs. CloudFormation is a free add-on and it supports many of the Amazon Web Service resources. (Amazon Web Services(a) 2012).

4.1.2 Amazon Beanstalk

Beanstalk is a PaaS service which works in conjugation with other AWS services such as EC2, S3 and Amazon SNS. Beanstalk enables a quick deployment and management of applications in Amazon cloud. Beanstalk automatically handles capacity, load balancing and application monitoring when the customer uploads his/her application, without losing the control of the infrastructure and other services. Compared to conventional PaaS solutions, Beanstalk does not reduce developers’ control over the infrastructure. Customer retains full control over the Amazon resources. Beanstalk supports software stacks such as PHP, Python and Ruby with Apache HTTP, .NET with IIS 7.5, Java with Apache Tomcat and NodeJs. In addition Beanstalk supports Git version control and Java Web Application Archive as deployment methods. In future Beanstalk should support even more languages and APIs for the developers. (Amazon Web Services(b),(c), 2013).

Beanstalk has automated capacity provisioning, load balancing, auto-scaling and application deployment. It works in conjugation with git repositories. Amazon provides toolkits for Visual Studio and Eclipse and with these toolkits, deploying and managing the application is handled in the same IDE environment. Beanstalk automates many management tasks such as monitoring, version deployment, health checks and logging. The advantage in Beanstalk compared to other PaaS offerings is that it does not restrict the infrastructure control in any way and users can take control over many infrastructure parts. Other AWS resources are in use any time. Still the infrastructure control can be left to the PaaS itself if the user want to focus just on the applications. (Amazon Web Services(b), 2013).

In addition Beanstalk allows the user to select operating system(Linux or Windows), choose from several database options(EC2 databases), to troubleshoot from logging straight to EC2, to run the application in various availability zones, to use SSL encryption for increased security, to monitor events, health, etc. of application with CloudWatch, to change application server settings and environment variables, to run
other components side-by-side in EC2 and to access log files. (Amazon Web Services(b), 2013).

4.1.3 Scaling

Concerning the scalability, Beanstalk with EC2 services performs well. EC2 has virtually infinite scalability for any normal applications. Auto scaling conditions can be configured as the user desires. Amazon claims that their auto scaling is particularly good for applications that have weekly, daily or even hourly variability in usages. Auto scaling in EC2 is enabled by Amazon CloudWatch and it is a free feature for anyone who is registered to Amazon’s services. Auto scaling works in a way that it increases and decreases the amount of instances according to traffic. User can define additional rules with the help of CloudWatch. (Amazon Web Services(c), 2013). Factors and possible advantages are derived from the previous chapter.

Beanstalk offers (with EC2) built-in load balancer. Also, with the help of Amazon Route 53(Amazon’s scalable DNS web service), Beanstalks allows custom domains for load balancers. User can use domain names configured by Amazon Route 53 for Beanstalk applications. Beanstalk with EC2 offers auto-scaling of application server, but does not support any kind of auto-scaling of databases. (Yuan(a), 2011; Amazon Web Services(c), 2012).

4.1.4 Service agreements and possible vendor lock-in

Beanstalk works naturally only in AWS infrastructure and it cannot be moved to another providers infrastructure. According to Yuan, migrating applications from and into Beanstalk is easy and portability of applications are high. Vanderveken in his blog also confirms that migrating a small web application from AWS Beanstalk to another vendor is fast, easy and it takes only little effort. (Amazon Web Services(b),(c),(e), 2013; Vandervaken, 2012; Yuan(a), 2011). Beanstalk lets the user to create up to 25 applications and 500 application versions. AWS allows 10 environments to be run across the applications. The default AWS account is limited to 20 EC2 instances and 10 Elastic Load Balancers. More resources can be requested for evaluation. (Amazon Web Services(b),(c), 2013). EC2 and Beanstalk is available in following zones: US East, US West, US West 2, EU(Ireland), Asia Pacific(Singapore), Asia Pacific(Sydney), Asia Pacific(Tokyo) and South America(Sao Paolo). Amazon has the most extended infrastructure offering concerning the location. (Amazon Web Services(b),(c),(e), 2013).

Possible vendor lock-in risks come from the fact that Beanstalk PaaS-service works only with AWS IaaS-services. Possible advantages may be freely configurable data locations and extensive migration and portability possibilities.

4.1.5 Security

Amazon does not offer in-built SSL encryption nor the possibility to run private or hybrid cloud configurations. Concerning the SSL encryption, the developer has the responsibility to configure their SSL encryption themselves. The SSL certification can be ordered from external authority such as VeriSign or Entrust. Amazon offers detailed guide how to configure SSL to the users domain. (Amazon Web Services(e), 2013).
Beanstalk cannot be run in a private cloud by default without additional services. By default all applications are public. Developers can use Amazon VPC (Amazon Virtual Private Cloud) to private and isolate applications to a virtual network defined by the user. Users can also combine VPC with other Amazon services (AWS Direct Connect) to make a hybrid cloud environment. (Amazon Web Services (d), (e), 2013; Yuan (a), 2011).

Possible advantage may become from the extensive authentication system. Amazon offers AWS Identify and Access Management (IAM) framework for authentication configuration for their services. This can be used in conjunction with Beanstalk. With IAM, the user can define minimum password length, additional password complexity rules (lower/uppercase, numbers, symbols, etc. rules) for other users. With IAM, user groups and group permissions can also be defined. With the help of IAM, authentication security can be increased greatly. AWS account allows the user to have full access to all AWS resources. With IAM user can make different usernames and passwords for services like web pages and forums. AWS has also Multi-Factor Authentication (MFA) service which can elevate the security even further. With MFA to log in, in addition to username and password, an authentication code is needed to log in to an MFA device. MFA can be enabled for any IAM account. (Amazon Web Services (d), (e), 2013).

4.1.6 Framework and supported languages

Amazon has extensive programming language and development framework support. It also does not restrict the use of any database service. Amazon has many possible advantages in this criteria. Beanstalk supports Java, PHP, Python, Node.js, Ruby and .NET. Development stacks include Apache Tomcat (for Java), Apache HTTP server (for PHP and Python), Nginx (for Node.js), Passenger (for Ruby) and Microsoft IIS 7.5 (for .NET) (Amazon Web Services (b), (c), 2013). Beanstalk does not restrict the user to any specific database technology. Users can choose Amazon Relational Database Service (RDS), DynamoDB, SimpleDB, or even other service provider's offers such as MS SQL server or Oracle to be run on EC2. (Amazon Web Services (b), (c), (e), 2013). Beanstalk has both, graphical web interface and command line tools. In addition Beanstalk has also web console and web based monitoring tools. (Amazon Web Services (b), (c), 2013; Yuan (a), 2011). Beanstalk does not have separate testing environment, instead the user can make own instances for production, development and testing purposes. (Amazon Web Services (b), 2013; Yuan (a), 2011).

4.1.7 Pricing

The largest possible advantage comes from the fact that Beanstalk's PaaS-service is free. In related services Amazon offers 750 hours of EC2 Linux/Unix/Windows usage, 750 hours of Elastic Load Balancer processing and 30 GB of Amazon Elastic Block Storage. In S3 service 5 GB of standard S3 storage is offered. From RDS 750 hours of Micro DB instances are offered, which includes 20 GB of database storage, 10 million I/Os and 20 GB of backup storage. Most of these are valid only for 12 months and are only for new customers. (Amazon Web Services (b), (e), 2013).

In EC2, the costs come from the services which are used in conjugation with Beanstalk. AWS services can be counted for pay-per-usage or usage-based. For example user can buy 15 GB of Load Balancer data processing which cost 0.12$ or buy it by usage-based $0,008/GB. (Amazon Web Services (b), (c), (e), 2013). Yuan (2011) states that developing
an entry level web application for Beanstalk is expensive compared to many other competitors and customers have hard time of keeping track how much they have to pay for the AWS usage. After the free tier period, the price can be even in a single node setup ~40$ a month. (Yuan(a)(b), 2011).

4.1.8 Summary

The main reason why Amazon EC2 and Beanstalk are selected to the survey is that in conjugation they offer a lot of configurability, flexibility and a lot of additional Amazon services work in conjugation with Beanstalk. This can be useful later when more content and additional applications are created. It also provides a lot of control for the infrastructure, for example customer can choose the location of the servers from various alternatives. Without Beanstalk, EC2 alone should not be called strictly a PaaS, instead it is a set of various cloud configuration and service components from which the customer can build his/her own platform environment. For example when enabling Beanstalk and CloudFormations with EC2 instances, a very flexible, robust and scalable long-lasting cloud platform can be constructed. From there the user can add new AWS components and services when desired or when new features are needed. Possible advantages in each criterion are summarized in table 6.
<table>
<thead>
<tr>
<th>Scalability</th>
<th>Possible advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>built-in load balancer</td>
<td>Beanstalk offers built-in load balancer.</td>
</tr>
<tr>
<td>Custom domain possibility for load-balancer</td>
<td>Amazon's scalable DNS web service allows custom domains for load balancers.</td>
</tr>
<tr>
<td>Auto-scaling of application server</td>
<td>Yes</td>
</tr>
<tr>
<td>Auto-scaling of database</td>
<td>No</td>
</tr>
<tr>
<td><strong>Vendor-lock in and service agreements</strong></td>
<td></td>
</tr>
<tr>
<td>The presence of vendor lock-in</td>
<td>Locked to AWS infrastructure</td>
</tr>
<tr>
<td>Amount of applications</td>
<td>25 applications, 500 application versions</td>
</tr>
<tr>
<td>Portability of applications</td>
<td>High</td>
</tr>
<tr>
<td>Ease of migration</td>
<td>High, easy to migrate from and to</td>
</tr>
<tr>
<td>Data locations</td>
<td>US, EU, Asia, South America with user control</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td></td>
</tr>
<tr>
<td>The presence if in-built SSL</td>
<td>No, developer's responsibility</td>
</tr>
<tr>
<td>Private cloud possibility</td>
<td>No, but virtual private cloud possibility(VPC)</td>
</tr>
<tr>
<td>Hybrid cloud possibility</td>
<td>No, but can be configured with VPC for virtual hybrid environment</td>
</tr>
<tr>
<td>Extensiveness of authentication system</td>
<td>Extensive(IAM framework, MFA)</td>
</tr>
<tr>
<td><strong>Framework</strong></td>
<td></td>
</tr>
<tr>
<td>Supported languages and framework stacks</td>
<td>Java(Apache Tomcat), PHP(Apache HTTP server), Python(Apache HTTP server), Node.js(Nginx), Ruby(Passenger), .NET(MS IIS)</td>
</tr>
<tr>
<td>Supported database frameworks</td>
<td>Any, does not restrict users</td>
</tr>
<tr>
<td>GUI</td>
<td>Yes</td>
</tr>
<tr>
<td>Command line interface</td>
<td>Yes</td>
</tr>
<tr>
<td>Web based monitoring</td>
<td>Yes</td>
</tr>
<tr>
<td>Testing environment</td>
<td>No</td>
</tr>
<tr>
<td><strong>Pricing</strong></td>
<td></td>
</tr>
<tr>
<td>Free entries</td>
<td>Beanstalks free, 750 hours of EC2 usage, and load balancing, 12 month durations only</td>
</tr>
<tr>
<td>Pricing model options</td>
<td>Pay-per-usage, usage-based</td>
</tr>
<tr>
<td>Costs for entry level applications</td>
<td>Possibly high, hard to track</td>
</tr>
</tbody>
</table>

Table 6. Criterion and possible advantages for Amazon EC2 with Beanstalks
Overall, Amazon EC2 and Beanstalks perform very well against the criteria. The biggest concern in this context is the possibility of high entry level costs for entry level applications.

4.2 Google App Engine

Google App Engine (GAE) is Google's PaaS option for hosting web applications in Google's data centers. GAE is one of the first true PaaS platforms that developers really adopted and it is still one of the purest platform services out there. It was first built only for Python applications but later extended to support Java, Java Virtual Machine languages, PHP and Go. The different between Beanstalks and GAE is that GAE hides the infrastructure completely from the user. (Google Developers, 2012; Google Inc(c), 2013; Yuan(b), 2011).

4.2.1 Framework and main components

GAE consists of application and runtime environments, The sandbox, datastores and databases, account services and various APIs for different services (for example Google Search).

Application environment

GAE has dynamic web serving and it supports Java, PHP, Python and Go environments. GAE has also automatic scaling and load balancing and persistent storage. Users have access to APIs for authenticating and email services for Google Accounts. GAE offers also installable local development environment to simulate online Google App Engine. Tasks can be queued and scheduled for specific time intervals and triggered when desired. Each GAE provided environment (Java, PHP, Python, Go) include standard protocols and necessary technologies for web application development for the particular platform. (Google Developers(a), 2012; Google Inc(c), 2013).

The Sandbox

GAE has a sandbox environment which isolates the application in its own environment which has independent hardware, operating system and physical server location. The sandbox can limit many crucial things that may introduce security vulnerabilities. An application can only be accessed by provided URL fetch and email services. The application can be connected only by HTTP and HTTPS(SSL) requests and on the standard ports. Applications are write-protected in runtime and application can read files only uploaded with the application code. Applications use App Engine data store, memcache and other service for all data interaction. Application code is executed through web requests, queued or scheduled tasks. There is a 60 second response time limit in which the response must be return. (Google Developers(a), 2012; Google Inc(c), 2013).

Runtime environments

GAE has four runtime environments: PHP, Java, Python and Go. PHP runtime environment is still only an experimental feature. Early versions of SDK and local development server are available. PHP applications are limited by the white list and only white listed applications can be deployed on GAE. According to Google, the
restriction removal and further developing on PHP environment is under construction. (Google Developers(a), 2012; Google Inc(c), 2013).

Java applications for Java runtime are developed with common Java web developments tools and API standards (Java Servlet, Java Server Pages (JSP)). The most recent Java runtime environment uses Java 7 but the SDK works also with Java 6. GAE Java runtime environment includes also the Java SE Runtime Environment (JRE) 6 libraries. The sandbox restricts the applications for example to open a socket or write files through the runtime. GAE services are accessed by standard Java APIs. For the datastore, Java Data Objects and Java Persistence API interfaces are used. The JavaMail API can be used to send emails through App Engine Mail service. URL fetching is done with Java.net HTTP API. In conjugation with Java, the developers can use also additional languages and web technologies such as JavaScript, Ruby or Scala. (Google Developers(a), 2012, Google Inc(c), 2013).

Python applications can be run on an optimized Python interpreter. GAE has extensive APIs for python web application development (Django), framework and tools for managing and accessing application data. Python environment includes only standard Python library and not all of them can be run in the sandbox. Third-party libraries work only if they are written in pure Python. Attempts to socket open and file write causes an exception. To make things more easy, much of the Python modules which are not functioning are disabled by the runtime environment. Extension code written in another language will not work with GAE Python environment. There are APIs for the datastore, Google Accounts, URL fetch and email services. In addition there is web application framework (webapp2) to help to start building application. (Google Developers(a), 2012; Google Inc(c), 2013).

GAE's Go environment allows to applications developed in the Go programming language. The GAE Go SDK has Go compiler and standard library and it does not have any additional dependencies. It has limitations similar to Java and Python environments in sandbox environment (limited third-party support, socket opening and file writing permitted). The SDK has automated build service and developers do not have to deal with the compiler. Environment has short edit-compile-run flow and it builds the application automatically when source code is modified. As with other runtime environment Go environment has APIs for most of the App Engines services. (Google Developers(a), 2012; Google Inc(c), 2013).

**Datastores and databases**

GAE offers many alternatives for data storage. App Engine Datastore (BigTable) is a NoSQL object datastore with query engine and transactions. App Engine Datastore provides auto-scaling service in which the datastore grows as the amount of data traffic grows. Datastore does not function like a traditional relational database, instead data and objects are filtered by a kind and a set of properties. The application code controls the structure of the data. Java and Python interfaces provides the means to control the data. Google Cloud SQL is a relational SQL database service based on MySQL RDBMS. Google Cloud Storage is a storage service for objects and files with large storage (terabytes). (Google Developers(a), 2012; Google Inc(c), 2013).

**Google accounts**

Google accounts can be integrated to the app with GAE. User can sign to the app with a Google accounts, read emails and have his/her name displayed with the associated
account. User does not need to register any new accounts if Google account is present. Also the developers do not need to develop user account system which saves a lot of developing time. All the same account features can be used within the application than for example in gmail.com. The admin side of the site is also easy to implement with Google accounts. (Google Developers(a), 2012; Google Inc(b), 2013).

**Figure 19.** Google App Engine architecture used in Streak web service. (Mawani, 2012).

**Other App Engine services**

GAE has also a variety of additional services which the application can use. URL Fetch lets the application to access resources like web services and other data through Google infrastructure. GAE's mail services sends email messages through Google infrastructure. Memcache service contains a high performance memory cache that can be used by multiple instances of the application. Memcache can be used for temporary data that does not need the datastore. This way of saving data increases the service speed immensely. There is also API for image manipulation which lets the user to re-size, crop, rotate and flip images in JPEG and PNG formats. (Google Developers(a), 2012; Google Inc(c), 2013).

**The SDK**

GAE has software development kits(SDKs) and emulator application for Java, Python, PHP and Go environments. With these all App Engine services can be run locally. SDKs include all the APIs and libraries. Secure sandbox environment is also emulated from where the developer can check system resources accesses. SDK has also a tool for uploading the application to the App Engine. User can also update old versions and also to test new version in the App Engine without losing the old version. The SDK includes The Administration Console, which is a web-based interface for managing applications running on App Engine. With the console, developer can create applications, configure
domain names, version control, examine logs and browse datastore. (Google Developers(a), 2012; Google Inc(c), 2013).

4.2.2 AppScale

AppScale is an open source GAE based PaaS which can function in a hybrid cloud environment. Its goal is to increase application portability in GAE applications. It has TyphoonAE for Linux and Web2py environments and it offers SQL database migration in Google App Engine. AppScale does not support all of the GAE features such as transactions and namespaces. AppScale is API-compatible with GAE. Developers can upload their code and data to Google cluster and use resources (free-tier or resource rental). Because AppScale supports a hybrid cloud environment, it means that the application can access services from different clouds at different times or concurrently. (Bunch et al, 2011; Krintz et al, 2012).

In addition to GAE APIs, AppScale can be conjugated with Amazon's EC2 services with an independent API which is not possible in GAE alone. With this API, AppScale allows the developers to create EC2 virtual machines and control those with simple command line tools. There is also MapReduce Streaming API which is not found in GAE. With this API developers can specify a Mapper and Reducer program that can run under MapReduce programming paradigm (large data-sets with parallel and distributed algorithm on a cluster). MapReduce allows scalable and fault-tolerant development. In AppScale, Mappers and Reducers are supported by Python, Ruby and Perl languages/environments. (Bunch et al, 2011, Krintz et al, 2012).

AppScale has still some limitations that the developers should be aware of before adapting the platform. Because AppScale is on virtual platform, terminating instances may cause a loss of data. AppScale handles this issue by uploader and downloader implemented in the environment. There are some data limitations both in file sizes and datastore capacity. When datastore increases the performance is decreasing. Also AppScale does not index data, instead it has a filtering feature. There is also some limitations with queuing tasks (not fault tolerant), mail services (only admin is allowed to send mail and reception is not implemented) and AppServers do not scale up and down, instead it deploys on all nodes. Also many GAE features are limited such as OS support (only Ubuntu) and many APIs (especially Java) are lacking such as Blobstore, Channel API and data stores. (Bunch et al, 2011, Krintz et al, 2012).

4.2.3 Scaling

GAE has built-in load balancer and custom domains for load-balancer with the limitation of naked domains (HTTP:\\mydomain.com) which are not supported. GAE has also auto-scaling of application server with web applications. Resources are allocated automatically when requests increase. (Google Developers(a), 2012; Google Developers(b), 2012; Google Inc(e), 2013; Fogg, 2008; Yuan(a), 2011; Yuan(b), 2011).

The difference in GAE, compared to other PaaS providers is the fact that it offers auto scalable database offering. GAE uses its own scalable data store BigTable. It is NoSQL database and it can be construct like relational database into tables, rows and columns which are indexed with unique IDs. This solution has some limitations such as weak support for data queries (too harsh table limitations for an NoSQL database) and difficulty of importing and exporting data because proper APIs are lacking. Also common web applications are usually based on relational SQL design so shifting to
NoSQL paradigm might be challenging. (Google Developers(a), 2012; Google Inc(c), 2013; Yuan(b), 2011). GAE has very good linear scaling for web applications. Especially in HTTP sessions GAE performs really good. On the down side GAE's raw performance(response times) is worse compared to some of the other providers. (Yuan(a), 2011).

4.2.4 Service agreements and possible vendor-lock in

Google has stated that portability and openness of the platform is very important. They have worked to ensure that the users can move applications and data off GAE whenever wished. GAE's SDKs are open sourced and Google have been working with third party groups to enhance standardization and portability between different environments. (Yuan(a),(b), 2012). However portability is still a concern with GAE and many are concerned that they are locked into Google's technologies. There are many open-source communities which have tackle this issue by making an open-source back-ends for Google's proprietary APIs. The biggest challenge is the BigTable database framework, which makes aspects like portability and migration a challenge.(Chun, 2012).

Google itself encourages healthy and competitive ecosystems. Moving GAE application from Google's infrastructure can be time-consuming as the developer has to integrate all GAE's components(which there are many). After all GAE components are made in a way that they are meant to work with Google's infrastructure. In Yuan's PaaS shootout, GAE's portability for applications has been classified as low and the existing applications are not friendly to migrate(due to BigTable). For example GAE does not support full Java SE platform and it is lagging behind many of its competitors. (Chun, 2011; Yuan(a), 2011).

Concerning application limits, with an administrator account, users can create 10 applications. With admin console the developers can delete and create applications. Same ID can not be re-registered. (Google Developers(b), 2012; Google Inc(d), 2013). Google has not revealed the specific data store locations but there are stores in North America and Europe. (Darrow, 2012). As of 2012, Google extended the data stores in Europe; There are more than three European data stores available now. Google stated that European customers need their own datacenters and their data saved inside the EU. (Google Inc(d), 2013).

4.2.5 Security

GAE's sandbox environment functions kind of a barrier for other applications and entities from accessing the code and data of the application. Sandbox isolates the applications from affecting other users. GAE has also some important security certifications such as SAS 70, SSAE 16 and ISAE 3402. (Google Developers(b), 2012; Google Inc(c), 2013).

GAE allows SSL traffic in appspot.com or user specified domain(application's domain). SSL service is configured through GAE Control Panel and it is billed separately through App Engine applications. (Google Developers(b), 2012; Google Inc(d), 2013). According to Yuan(2011), GAE does not have native support for private cloud environments. (Yuan(a), 2011) However lately, with additional third party services, the developers can integrate their cloud environments to GAE. For example RedHat has collaborated with Google to increase portability and openness of GAE, to allow GAE
applications to be run in a private cloud environment. This can be also achieved with AppScale. (Krintz et al, 2012; Ricknäs, 2013).

GAE offers three authentication systems for the applications. One system can be used for one application. Google Accounts allows the users to use their Google Account (gmail address) to access the application. Google Apps Accounts lets the developers to have own log-in on their application and domain. These accounts are made by the application admins. The third authentication system is OpenID. Any OpenID provider may sign in with an OpenID account. Administration is handled using Google Account or a Google App account. Google encourages the administrators to use same authentication as the application's users. GAE developers and admins can access the administration console in its own domain via an Internet browser. By default GAE is configured to use Google Accounts for authentication but the developers can change them and restrict accesses to certain domains only. (Google Developers(a), 2012; Google Inc(c), 2013).

4.2.6 Framework and supported languages

Programming language and framework support are quite limited and GAE does not support relational database services at all. As of 2012, GAE supports Java, Python, PHP and Go. Website view templates can include JavaScript, HTML and web applications can be made AJAX-enabled. (Google Developers(b), 2012; Google Inc(d), 2013). GAE has BigTable non-relational database structure. Applications which are based on a relational database will not run on GAE without heavy modification. GAEs datastore has GQL syntax which is non-relational and it only supports one-to-one and many-to-many relations. GAE has GUI based IDE tools, command line tools and also web-based monitoring. This extensive user interface tool-set provides a possible advantage. (Google Developers(a), 2012; Google Inc(c), 2013; Yuan(a), 2011).

According to Yuan(2011) testing GAE applications on development machines is hard. GAE does not have integrated build or testing environments. There is Python testbed module for unit testing but when using this, the developer must write(code) the tests him/herself in conjugation for example with Datastore and memcache. There are some third party providers who provide solutions for these lacking features. (Google Developers(a), 2011; Google Inc(c), 2013; Yuan(a), 2011).

4.2.7 Pricing

GAE offers free quotas for each one of its service. Blobstore has 5 GB free limit. Channel API has various free limits (657,000 calls, 100 channels, 200 hours). Code and Static Data Storage has first 1 GB for free limit. Datastore has 1GB usage a day free tier which includes 200 indexes(total limit), 50,000 write/read/small operations. Free quotas for applications are reset daily and pricing starts when the quotas exceed the free limits. All the free tier specifications can be checked from Google Developers web site. (Google Developers(a), 2012; Google Inc(a), 2013). Yuan(2011) states that support for normal accounts(free tier) is poor and some of the smaller providers have much better support willingness for free tier customers. Because of GAE's generous free tier quotas, most entry level web applications can be made free. Yuan also states that this is true. (Yuan(a), (b), 2011).
GAE has three pricing categories which vary features: Free, Paid and Premier. Free tier does not include usage based pricing at all. Paid and Premier accounts offer the same features (including usage based pricing) but Premier accounts (150$/month) include extensive operational support. All tools are available in every category. GAE offers safety quotas which include daily quotas and per-minute quotas. Daily quotas are refreshed daily at midnight Pacific time. On the downside, GAE does not have the possibility to usage-based pricing for normal users (only premier accounts have the possibility). (Google Developers(a), 2012; Google Inc(a), 2013).

4.2.8 Summary

Google App Engine offers robust environment for web applications. It's sandbox environment offer fast and easy development for various kinds of web applications. Also it has extensive free tier and entry level applications can be made virtually for free in certain extent. One of the key features of GAE is the BigTable scalable linear database system which also bring some limitations which may turn away customers. First of all it's the only PaaS provider in this comparison to offer auto-scalable database solution but on the downside it's only linear. The linearity means that portability and migration of existing applications may become hard or even impossible because importing and exporting data to BigTable is difficult because the lack of APIs. (Yuan(b), 2011). Possible advantages in each criterion are summarized in table 7.
<table>
<thead>
<tr>
<th><strong>Scalability</strong></th>
<th><strong>Possible advantage</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>built-in load balancer</td>
<td>Yes</td>
</tr>
<tr>
<td>Custom domain possibility for load-balancer</td>
<td>Yes, but no naked domains</td>
</tr>
<tr>
<td>Auto-scaling of application server</td>
<td>Yes</td>
</tr>
<tr>
<td>Auto-scaling of database</td>
<td>Yes, but only for non-relational database</td>
</tr>
</tbody>
</table>

**Vendor-lock in and service agreements**

| The presence of vendor lock-in         | Possible, high risk of locking to Google infrastructure |
| Amount of applications                 | 10 with admin account                              |
| Portability of applications            | Low                                              |
| Ease of migration                      | Low, hard to migrate applications                 |
| Data locations                         | US, Europe (three datastores)                     |

**Security**

| The presence if in-built SSL           | Yes                                            |
| Private cloud possibility              | Not officially but possible with add-ons        |
| Hybrid cloud possibility               | Not officially but possible with add-ons        |
| Extensiveness of authentication system | Limited to Google Accounts(App Accounts, OpenID) |

**Framework**

| Supported languages and framework stacks | Java, Python, Php, Go                           |
| Supported database frameworks           | BigTable(non-relational)                        |
| GUI                                    | Yes                                            |
| Command line interface                  | Yes                                            |
| Web based monitoring                    | Yes                                            |
| Testing environment                     | No                                             |

**Pricing**

| Free entries                           | Generous                                       |
| Pricing model options                   | Depends on the account level(usage-based only in premium accounts), daily quotas |
| Costs for entry level applications      | Free or very low                               |

Table 7. Criterion and possible advantages for Google App Engine
GAE is a good, well-designed PaaS which has a very extensive free tier and it is very good for entry level application projects, on the downside it is still pretty unfinished as many APIs are still lacking (especially in Java) and constantly in work-in-progress mode.

4.3 Heroku

Heroku is a PaaS owned by Salesforce.com. It is one of the first cloud platforms released and it was first released June 2007. Heroku platform officially supports Java, Ruby, Node.js, Scala, Clojure and Python languages and environments. It also offers build-packs for example for Perl and Php environments and application development among others. Currently Heroku supports Linux operation systems and it has stacks for Ubuntu and Debian. (Henry et al(a),(d), 2013). Heroku encourages version control (git) based group development on their platform. Heroku functions over Amazon's (AWS) infrastructure so it has many same advantages and disadvantages of Amazons Beanstalk platform. Heroku has an extensive support of add-ons for data stores, search, logging, analytics and other features. (Emison Masters, 2013).

4.3.1 Framework and main components

Heroku's framework is built around Unix-based process model and it consist of dynos, environment stacks and other services such as “slugs”. These are described below.

**Process model**

Heroku has Unix based process model and abstraction for server- and web applications. The process model has a way of dividing workloads and scaling over time. A process means a running program. To understand the process model, Heroku divides the problem in relationship between the processes and scaling(dynos). (Henry et al(c), 2013).

![Figure 20. Heroku process model. (Henry et al(c), 2013).](image)
Dynos

Dynos are entities (i.e., Virtualized Unix containers) which run a single user-specified command. Commands include web processes, timed jobs and queuing. Admin and maintenance commands can also be run in dynos. The number of dynos are increased or decreased any time when the application needs them or not. Processes are monitored and bad processes are taken down and new dynos are declared over the crashed ones. This means that application is resistant to OS changes and server-side problems. Dynos have also intelligent routing feature which is a tracking system to track the dynos and routing of HTTP traffic. Dynos can be declared in many infrastructure locations which means that if some infrastructure fails, the application can still be run with other dynos which are located in different location. When more dynos are declared, more redundancy is gained. (Henry et al(c), 2013).

Stacks

A stack is a deployment environment which includes operating system, runtime and libraries. Different stacks support different runtime environments and libraries. Heroku has two stacks: Celadon Cedar which is based on Ubuntu (10.04 as of 2013) and Badious Bamboo which is based on Debian Lenny (5.0 as of 2013). Celadon Cedar is the default stack which support most of the environments. Badious Bamboo is for Ruby applications but they can be migrated to Celadon Cedar with instructions which are found in Heroku Dev Center. (Henry et al(c), 2013).

HTTP Routing and the Routing Mesh

Heroku has HTTP routing feature that automatically load-balances and routes HTTP requests (direct routing to web dynos). Load balancer takes care of HTTP and SSL termination and routing mesh makes sure that HTTP requests are passed to web dynos. Heroku has a full HTTP 1.1 support and asynchronous web server handling with multiple responses from a single process (multi-threading). Web sockets are not yet supported in Heroku. (Henry et al(c), 2013).

Logplex

Logplex is Heroku's logging system which keeps logs from the application and other platform components. The main function of Logplex is to clarify the event stream (which can be very convoluted) of the application and platform. (Henry et al(a), 2013).

Slug Compiler

Heroku has so called “slugs” which are compressed and packaged copies of an application. These slugs can be created for example every time when new version is pushed to a source code repository (git push). Slugs are optimized for distribution across the dyno manifold. Slugs are expanded to dynos on execution. Slug compiler makes slugs and it ignores unnecessary files and use only the dependency files (these files can be configured). Maximum slug size is 300 MB. (Henry et al(a), 2013).

Languages

Heroku has official support and build-packs for Ruby, Java, Python, Clojure, Scala, Node.js, Gradle, Grails and Play. (Henry et al(a), (d), 2013).
Other Features

Domains for applications in Heroku are appname.herokuapp.com and appname.heroku.com. Custom domains are aliases of domains above. Naked domains are not recommended. Heroku is controlled through command-line tool set which functions as an interface to the Web API. Through command-line, users can create applications, setup dynos, take backups and configure add-ons. Heroku can be customized and extended by user created add-ons (extensive add-on library in https://addons.heroku.com/). Processes can be managed and commanded with unique configure file. With this file, user can manage what commands are by dynos. User can also select region for the application and where it is run. Currently the location can be selected between the US and EU (Amazon Web Services infrastructure). Heroku has also production check feature. It tests the applications configuration against a set of user defined criteria. For example up-time assurance can be tested with this feature. (Henry et al(a), 2013).

4.3.2 Scaling

Heroku does not offer any kind of auto-scaling per se, instead it uses the process model that lets the user to scale up or down resources instantly through command-line and dashboard. Developers have to do their own server monitoring through the dashboard and add resources when needed. The dyno setup is managed by a dyno manager in each application. As with Amazon's offering, Heroku does not offer auto-scalable database solutions but in-built load balancer is presented. Custom domains for load-balancer are also offered. (Henry et al(a), 2013, Yuan(a), 2011).

4.3.3 Service agreements and possible vendor-lock in

Much of Heroku's vendor lock-in and service concerns are the same as Amazon's as the infrastructure is the same. However there are some additional limitations such as no control over the data locations. Migration of applications is also hard and it takes time and skills to do it. Main benefits of Heroku is the broad selection of third-party add-ons for various features. Concerning application limitations, Heroku allows maximum on 100 applications created on verified and 5 on unverified accounts. Heroku is based on AWS infrastructure so it can not be moved to another IaaS provider without migrating the application to a different platform. (Henry et al(a), (b), 2013). Some web developers think that Heroku is “in the middle” as they abstract infrastructure complexity and at the same time not force the user to code against proprietary APIs. (Zito, 2012). Yuan states that migrating applications in Heroku is not easy and portability is low. (Yuan(a), 2011). On the other hand it is also stated that Heroku environment is open and it is easy to use third-party tools or add-ons to help migration processes. (Godse; 2011).

4.3.4 Security

Because Heroku's infrastructure is hosted by Amazon, it utilizes AWS security technologies. Heroku has also third-party credit card payment encryption system. Heroku has also third party security testers who test the platform regularly. Heroku also states that each of the customer's applications are run in an isolated environment and it cannot interact with other applications or parts of the system. The customer data is treated the same way and databases are SSL encrypted. The security responsibilities of
third-party add-ons are in the hands of their providers. Customers do not have to worry about backups since deployed applications are automatically backed up in the deployment process. Heroku also allows and encourages coordinated penetration tests and network security scans regularly. (Henry et al(a), 2013).

Heroku offers individual user accounts, user groups, various authentication safety measures, stored data encryption and various security certifications. (Emison Masters, 2013). However, there have been reports of security flaws in Heroku platform. For example developer and security consultant Jonathan Rudenberg has criticized Heroku's build system security. (Rudenberg, 2012).

Heroku offers SSL encryption for additional charged add-on. SSL endpoint add-on is used to enable SSL for custom domain. Before user can provision the SSL endpoint, SSL certificate must be purchased from one of the certified SSL providers. Enabling SSL encryption in Heroku is little more complicated than in some of the other PaaS providers. (Henry et al(a), 2013). Heroku does not support running on private or hybrid cloud environments. (Henry et al(a), 2013; Yuan(a), 2011).

Heroku offers SSL encryption for additional charged add-on. SSL endpoint add-on is used to enable SSL for custom domain. Before user can provision the SSL endpoint, SSL certificate must be purchased from one of the certified SSL providers. Enabling SSL encryption in Heroku is little more complicated than in some of the other PaaS providers. (Henry et al(a), 2013). Heroku does not support running on private or hybrid cloud environments. (Henry et al(a), 2013; Yuan(a), 2011).

Heroku has three situational authentication methods: email and password, API token and SSH key. The API token is obtained through email and password. The token is used for authenticating the API requests and it can be generated by the user when needed from the web interface. The SSH key is used in git(version control) authentication. SSH key is created and registered during first Heroku log in. (Henry et al(a), 2013). The applications written for Heroku can use AD/LDAP(Active Directory, Lightweight Directory Access Protocol) authentication. (Emison Masters, 2013).

4.3.5 Framework and supported languages

Heroku works best on command line based Ruby or Python environments but through extensive third party add-on offering, many different kinds of framework environments can be built. Officially Heroku supports Ruby, Java, Python, Clojure, Scala, Node.js and Play. (Henry et al(a), (d), 2013). There are also unofficial third-party build-packs in theory for any language or framework. Popular build-packs include Go, Erlang and PHP. At default, Heroku's database is based on PostgreSQL. (Henry et al(a), 2013). With add-ons additional data stores can be added such as Amazon RDS, Hadoop, Memcache, etc. (Emison Masters, 2013). Since Heroku is first built Ruby development in mind, so it's interface tool set is mainly command line based. (Yuan(a), 2011). Still it offers web-based monitoring and plug-in for Eclipse SDK. Logs can be accessed through web with Logplex feature. (Henry et al(a), 2013). Heroku does not have native integrated build or testing environments but testing on development machine is easy. (Yuan(a), 2011). Heroku recommends the developers to have different environments(repositories) for staging and production. If this is not enough, through add-ons various kinds of testing environments can be configured. (Henry et al(a), 2013).

4.3.6 Pricing

Heroku has a mix of usage-based and monthly based pricing models. Free tier is extensive and is enough for smaller web applications. On the other hand, Heroku may become expensive fast when more resources are needed. Especially increasing add-ons increases the price dramatically. Heroku offers 750 free dyno-hours per month. This tier is valid for any type of dyno. In practice the free tier is enough to run one dyno for
entire month free of charge or two dynos for half a month (375 free dyno-hours per two dynos). In addition, Heroku offers 10k row free starter database. (Henry et al(b), 2013).

Heroku charges .05$/ per dyno-hour after the free tier offer is surpassed. For example if application has four dynos it costs $0.20 per hour. It should be noted that Heroku is charged on clock time not usage. (Henry et al(b), 2013). Even if the dyno is on low usage it will cost as much as in full usage. (Emison Masters, 2013). However, user can put dynos to a sleeping mode which stops the usage. Add-ons have their own month based pricing. Basic 10M row database is $9 per month. (Henry et al(b), 2013). Because the free tier is so generous, in practice entry level applications developed on Heroku are free. (Henry et al(b), 2013; Yuan(a), 2011).

4.3.7 Summary

Heroku is a robust platform, especially for Ruby and Python based application since it has extensive command line functionality embedded in it. The free tier is also extensive enough to support most of the entry level web applications which makes it a good alternative when entering to cloud computing business. Also the extensive third-party add-on options brings a lot of versatility to Heroku. On the downside, Heroku may become very expensive fast, when more expansive applications are made. Also the infrastructure is completely hidden and unconfigurable by the user and when the infrastructure goes down, so goes the applications. Also some of the third-party features may be difficult to use and poorly documented. Possible advantages for criterion are summarized in table 8.
<table>
<thead>
<tr>
<th>Scalability</th>
<th>Possible advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>built-in load balancer</td>
<td>Yes</td>
</tr>
<tr>
<td>Custom domain possibility for load-balancer</td>
<td>Yes</td>
</tr>
<tr>
<td>Auto-scaling of application server</td>
<td>No</td>
</tr>
<tr>
<td>Auto-scaling of database</td>
<td>No</td>
</tr>
</tbody>
</table>

**Vendor-lock in and service agreements**

| The presence of vendor lock-in              | Yes (AWS infrastructure) |
| Amount of applications                      | 100 on verified accounts, 5 on unverified |
| Portability of applications                 | Low                  |
| Ease of migration                           | Low, hard to migrate   |
| Data locations                              | AWS data locations(US, Europe), no user control |

**Security**

| The presence if in-built SSL                | Yes, paid service     |
| Private cloud possibility                   | No                  |
| Hybrid cloud possibility                    | No                  |
| Extensiveness of authentication system      | Limited to email/password, API token and SSH key |

**Framework**

| Supported languages and framework stacks   | Ruby, Java, Python, Clojure, Scala, Node.js, Play and third-party buildpacks for others |
| Supported database frameworks             | Default: PostgresQL, buildpacks for Amazon RDS, Hadoop, Memcache, MongoDB, RabbitMQ |
| GUI                                        | No(Eclipse plugin)   |
| Command line interface                     | Yes                 |
| Web based monitoring                       | Yes                 |
| Testing environment                        | No, but easy testing with add-ons |

**Pricing**

| Free entries                               | 750 hours(1 dyno)/month |
| Pricing model options                      | Time-based, add-ons monthly based |
| Costs for entry level applications         | Low                 |

**Table 8. Criterion and possible advantages for Heroku**
Heroku's extensive add-on and third party support is one of its main strengths. Many default limitations can be bypassed with plugins and add-ons if developers have enough technical knowledge.

4.4 Cloud Foundry

Cloud Foundry is an open source PaaS and it is developed by VMware and it is based on Apache License 2.0. Since it's open source nature, Cloud Foundry is used as a base for many PaaS providers (such as Stackato). Cloud Foundry is still at its beta phase and currently it supports run-times and frameworks for Java (Spring), Ruby (Rails, Sinatra), Node.js and Scala (Play). As a database services, Cloud Foundry supports MySQL, Postgre, MongoDB, Redis and RabbitMQ. What differentiates Cloud Foundry from many other PaaS providers is the fact that infrastructure services and hosts below the platform layer can be changed without the need to rewrite the application code (provided that the services support Cloud Foundry). Cloud Foundry can be applied in public, private and “micro” clouds environments. There are also interface for additional services such as data and messaging. What makes Cloud Foundry differ from other PaaS offerings is that it is not hosted by anyone. User can download the source code and host the platform itself. (GoPivotal, 2013; eUKhost® Ltd., 2012).

4.4.1 Framework and Services

Officially Cloud Foundry supports Java, Ruby, Node.js and Scala. Java build-pack has many JVM-based languages supported such as Java, Groovy and Scala. Frameworks for these languages include Spring, Grails, Play and Lift build-packs. Java build-pack contains OpenJDK, Groovy, Tomcat and Play development kits and frameworks. Ruby pack has frameworks for Rails, Sinatra and Rack. Node.js pack supports JavaScript applications over Node.js runtime and Express framework. (GoPivotal, 2013).

![Cloud Foundry APIs](image)

Figure 21. Cloud Foundry APIs (eUKhost® Ltd., 2012).

Like in Heroku, many custom build-packs are also supported in Cloud Foundry. Custom build-packs are available in Cloud Foundry Community (located in github), also some Heroku's third-party build-packs work also in Cloud Foundry. Developers have also
option to build their own custom build-pack (for example Python with Django). All the build-packs have the same data structure to keep them straightforward. (GoPivotal, 2013).

Cloud Foundry offers also many useful additional services such as Redis open-source key-value store which can function like a data structure server (keys contain strings, hashes, lists, etc data). Cloud Foundry has also many database and data store services. From the most common ones, Cloud Foundry has MySQL and PostgreSQL services. In addition CF has also RabbitMQ (enterprise messaging system) and mongoDB, a non-relational NoSQL database service. (redis.io, 2013).

Cloud Foundry has been built open source community in mind. Whole platform (VCAP repository) is git based and it's instances can be run over Ubuntu. Anyone can pull the repository and built local platform including all the major components of VCAP and the additional services. (Zygmuntowicz, 2013); The platform consist of five main components: The Cloud Controller, Health manager, Router, DEAs (Droplet Execution Agent) and the additional services. Each of these components scale horizontally and more instances can be added as fitted. (Cloud Foundry, 2011).

4.4.2 Scaling

The users and developers have a lot of freedom in Cloud Foundry and that applies also concerning scalability. Cloud Foundry scales in two levels, in infrastructure level and in application level. Because of the open nature of Cloud Foundry a lot of scaling is left to the hand of users and developers. (Cloud Foundry, 2011). The good thing about Cloud Foundry is that it is very modular so scaling decisions can be made very precisely. (Von Eicken, 2011) Cloud Foundry offers built-in load balancer. (Cloud Foundry, 2011). Cloud Foundry uses sticky sessions in load balancers which may result scaling problems. Cloud Foundry supports also custom domains for load balancers. (Yuan(a), 2011). Auto-scaling of application server has been planned on Cloud Foundry but as of now it is not supported. (Von Eicken, 2011). Application scaling is the responsibility of the application's owner and developers. Cloud Foundry does not support auto scaling of databases neither. (Yuan(a), 2011).

4.4.3 Service agreements and possible vendor lock-in

Because the open nature of Cloud Foundry, the risk of vendor lock-in issues is low as the developers can host the PaaS themselves. In addition a lot of additional services outside the Cloud Foundry platform can be implemented to work in conjugation easily. The developers have almost endless freedom of building the platform. The parts of the application can be made to run outside the PaaS and for example the PaaS parts can be used as a front-end of some larger overall system. On the other hand, if not planned properly, the platform may become a hassle to manage, since the outside applications need to be managed also. (Von Eicken, 2011).

Cloud Foundry is not locked to any infrastructure. Developers can host the platform themselves and modify the platform almost endlessly to their needs. Natively Cloud Foundry supports vSphere (VMware), vCloud (VWware), OpenStack and Amazon AWS infrastructures. (Von Eicken, 2013). Still the best result would be to work with VMware's offerings, since it is Cloud Foundry's parent company. (Prichard, 2013). Ease of migrating and portability of applications has been classified as hard and moderate,
respectively. The more Cloud Foundry supported services (MySQL, Mongo, etc.) the other infrastructure uses the more easy the migration is. Java applications are the most flexible ones when using Cloud Foundry. For example CloudBees with Jenkins are easy to port to Cloud Foundry. (Yuan(a), 2011). There are also no official application limitations specified. The application limitations depends on how many instances does the developer run. (GoPivotal, 2013). The data locations depend on the infrastructure. Since Cloud Foundry is not locked into any infrastructure and it can be hosted by the developers themselves, the location can be anywhere.

4.4.4 Security

The strength of Cloud Foundry from security point of view is that it can be configured in many different kinds of cloud environments from private to hybrid to public. This brings flexibility to the developers. On the other hand, some aspects like SSL encryption can be a hassle and the user has more responsibility in some security challenges. Cloud Foundry does not include inbuilt SSL encryption and the responsibility for deploying one is on the hands of the developer. There are third-party tools and manuals for SSL deployment. (CloudFoundry-collaboration, GitWeb, 2011). Cloud Foundry's open nature means that it can be run in any kind of cloud configuration for increased security, including private and hybrid environments.

Before Cloud Foundry can do anything with the cloud controllers, the client must be authenticated and authentication request send. During authentication, the security token is created. Token duration can be configured by user (defaults at 2 weeks). Cloud Foundry has User Account and Authentication(UAA) service located between the user's browser and Cloud Foundry's dashboard service. UAA handles all the authorization between the browser and the CF services. (Olds, 2013). Cloud Foundry has also three standards (Oauth2, OpenID Connect, SCIM) dedicated for identification, authentication and group management. (Cloud Foundry, 2011).

4.4.5 Framework and supported languages

Cloud Foundry is particularly good for Java applications since it has extensive JVM support. Openness of Cloud Foundry means that developers can built basically any kind of framework consisting of any languages, run-times and build-packs. Of course the full use of this feature means that developers must have enough skills and understanding of the platforms. Cloud Foundry does not offer any kind of graphical or web interfaces out from the box. A lot of Cloud Foundry's usefulness depend on developer's skills and understanding on frameworks and their interoperability.

Cloud Foundry has a native support for applications written in Java, Scala, Groovy, Ruby and node.js. There are frameworks for Java(Spring and Play), Scala(Lift), Groovy(Grails) and Ruby(Rails and Sinatra). Like Heroku, Cloud Foundry supports also custom build-packs for different customized run-times, frameworks and application environments. Out from the box, Cloud Foundry has support for MySQL, Postgre, RabbitMQ, MongoDB and Redis. Other database solutions can be customized also, through different build-packs and configurations. (GoPivotal, 2013).

Cloud Foundry has in-built command line toolset but it lacks native GUI or web-based monitoring. (Yuan(a), 2011). The open nature means that various kinds of build-packs for application service interfaces and integration with development tools can be
achieved. (GoPivotal, 2013). Cloud Foundry does not have dedicated testing or build services. (Yuan(a), 2011). Instead the Micro Cloud architecture allows the user to run instances locally on development machine for testing purposes. The open nature of Cloud Foundry allows fast testing of applications in any cloud configuration with minimal modification. (GoPivotal, 2013).

4.4.6 Pricing

Pricing and pricing models are still to be defined. For now Cloud Foundry itself is basically “free”. The costs depend on the cloud environment setup. As of 2013, Cloud Foundry is yet to be priced and its pricing models are yet to be decided. The prices should be specified before the end of 2013. For now, Micro Cloud can be downloaded for free and open-source code can be downloaded from gitweb. Because Cloud Foundry can be downloaded for free, entry level costs are virtually free(depending on the cloud setup). (GoPivotal, 2013).

4.4.7 Summary

The philosophy in Cloud Foundry is different from other PaaS providers. Cloud Foundry is built to embrace the openness and freedom. With build-packs and add-ons, it has virtually infinite configurability and it can be tailored for many scenarios. It is the only PaaS that enables proper integration of various private and hybrid cloud environments together. It lets the developer to decide the infrastructure as well but also hides it if not wanted by functioning over VMware's default infrastructure. On the downside, the readiness status of Cloud Foundry is still very much in progress and the pricing models are yet to be decided so adopting it may be risky. At it's current state, Cloud Foundry requires a lot of understanding about cloud configurations, environments and technical knowledge if wanted to make use of it fully. Factors and possible advantages are summarized in table 9.
<table>
<thead>
<tr>
<th>Scalability</th>
<th>Possible advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>built-in load balancer</td>
<td>Yes, with sticky sessions</td>
</tr>
<tr>
<td>Custom domain possibility for load-balancer</td>
<td>No</td>
</tr>
<tr>
<td>Auto-scaling of application server</td>
<td>No</td>
</tr>
<tr>
<td>Auto-scaling of database</td>
<td>No</td>
</tr>
</tbody>
</table>

**Vendor-lock in and service agreements**

| The presence of vendor lock-in                      | Not locked to anything                                  |
| Amount of applications                              | No limitations                                           |
| Portability of applications                         | Moderate                                                 |
| Ease of migration                                   | Low                                                     |
| Data locations                                      | Any, depends on the infrastructure user decides         |

**Security**

| The presence if in-built SSL                         | No                                                      |
| Private cloud possibility                            | Yes                                                     |
| Hybrid cloud possibility                             | Yes                                                     |
| Extensiveness of authentication system               | Security token(2 week duration), UAA service, three known identification standards for authentication and group management |

**Framework**

| Supported languages and framework stacks            | Java(Spring, Play), Scala(Lift), Groovy(Grails), Ruby(Rails, Sinatra), buildpacks for others |
| Supported database frameworks                        | MySQL, Posgres, RabbitMQ, MongoDB, buildpacks for others |
| GUI                                                  | No                                                      |
| Command line interface                              | Yes                                                     |
| Web based monitoring                                | No                                                      |
| Testing environment                                 | No but local testing possibility with micro cloud       |

**Pricing**

| Free entries                                        | Free                                                    |
| Pricing model options                               | Not yet decided                                         |
| Costs for entry level applications                  | Depends on the cloud configurations                     |

**Table 9. Criterion and possible advantages for Cloud Foundry**
One of Cloud Foundry's main strengths come from the fact that it gives the developers almost full control of its configurations and it let's the users build the platform as they please and host it where ever they want. Also with the help of many buildpacks, there are virtually no limitations about the framework or database support.
5. Choosing by Advantage and Findings

After the PaaS alternatives are gone through and their criteria analyzed step by step, Choosing By Advantage (CBA) analysis can be made. CBA analysis should provide the best PaaS alternative (the PaaS with most advantages compared to others).

5.1 Choosing by Advantage decision making method

The origin of Choosing By Advantages (CBA) method was invented by Jim Suhr in 1969. It was created to help make sound decisions and to help to show that the decisions made are sound. The final birth of CBA happened not earlier than 1981 when Suhr understood that decisions are based on importance of advantages. CBA can be useful for architects, engineers, counselors, consultants, officials, organization leaders and others who need to make large decisions regularly. CBA may also be a good alternative for many situations which need sound decision making such as: organization planning, selecting contractors or vendors, selecting materials, equipment and products, and day-to-day management decisions. (Suhr, 1999). CBA has been used by many important decisions for example a highway location decision for the 2002 Winter Olympics was made with CBA. What differentiates CBA from other decision methods is the fact that, in CBA system, measurable terms such as factors, criteria, attributes and advantages are not used as synonyms. In CBA an attribute is defined as a characteristic, a quality or consequence of only one alternative. An advantage is a difference between two attributes of two alternatives. (Decision Innovations, 2013).

In CBA the decisions must be based on the importance of advantages not attributes alone. To achieve this, the decision maker must understand how to use attributes and advantages for comparison of alternatives. For example if cars are compared, factor could be fuel consumption. In this factor it is decided that the lesser consumption is better. The attribute in this case is the amount of fuel consumption. Car A has 7l/100km and car B has 5,5l/100km. In this case the B alternative has the advantage and thus wins the factor (table 1). In CBA the decision can be measured also by a disadvantage instead to an advantage but the end result would be the same. In this case A has a disadvantage in fuel consumption compared to B. Also the amount of advantage/disadvantage is important when looking at the overall decision. Alternative B has an advantage of 2,5l/100km over A or A has a disadvantage of 2,5l/100km compared to B. (Macomber, 2012).

<table>
<thead>
<tr>
<th>Car Alternatives</th>
<th>Factors</th>
<th>Advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fuel Consumption</td>
<td></td>
</tr>
<tr>
<td>Car A</td>
<td>7l/100km</td>
<td>B</td>
</tr>
<tr>
<td>Car B</td>
<td>5,5l/100km</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Airbags</td>
<td></td>
</tr>
<tr>
<td>Car A</td>
<td>5</td>
<td>B</td>
</tr>
<tr>
<td>Car B</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interiors</td>
<td></td>
</tr>
<tr>
<td>Car A</td>
<td>Leather seats</td>
<td>A</td>
</tr>
<tr>
<td>Car B</td>
<td>Vinyl seats</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Performance</td>
<td></td>
</tr>
<tr>
<td>Car A</td>
<td>150 bhp</td>
<td>A</td>
</tr>
<tr>
<td>Car B</td>
<td>90 bhp</td>
<td></td>
</tr>
</tbody>
</table>

Table 10. CBA – Buying a car
The decision is not always made by the number of advantages. Sometimes the importance of advantages can outweigh the amount of advantages. For example in the car example, alternative B wins clearly if the criteria for the car selection would be economical driving and safety even if the amount of advantages are even. (Macomber, 2012). In CBA the decisions must be based on importance of advantages. (Suhr, 1999). In addition to traditional CBA analysis(table 1.), two-list method can be used. In two-list method(table 2), a list of advantages of each attribute is made. For each advantage, an importance is decided. Then the alternative which has the most important set of advantages is chosen. If there are a same number of advantages, the one with more important advantages(depending on the criteria) is chosen. (Decision Innovations, 2013).

<table>
<thead>
<tr>
<th>Advantages of A</th>
<th>Advantages of B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nicer seats</td>
<td>Less Consumption</td>
</tr>
<tr>
<td>More horsepower</td>
<td>More Airbags</td>
</tr>
</tbody>
</table>

Table 11. Two-list method

Concerning the thesis and contractor's case, CBA is used because the decisions must be made quickly and the decision made must also be sound and valid for a longer time. Also the PaaS has clear features and factors which are suitable for CBA.

5.2 CBA Analysis for the selected PaaS alternatives

Each PaaS criteria has factors to be measured. These factors can be derived from the previous chapter and CBA analysis can be made from these factors. After the CBA analysis, advantages can be find out for each PaaS alternative and the best alternative should be clear.

5.2.1 Scalability

First factor that can be found out from scalability is the presence of in-built load balancer. Each alternative that has in-built load balancer has an advantage or disadvantage if it is missing. Each PaaS alternative has in-built load balancer so none of the alternatives has advantage here, although Cloud Foundry has sticky sessions which may produce scaling issues in some situations. This brings slight disadvantage for Cloud Foundry in this factor. (Cloud Foundry, 2011) The second factor from scalability is the presence of custom domain for load-balancer. Custom domains are good way to tracking down scalability. Each alternative that has custom domains for load-balancer has an advantage and a disadvantage if it is missing. Amazon, Google App Engine and Heroku has custom domains for load-balers. Cloud Foundry has a disadvantage in this factor also.

The next factor in scalability is the auto-scaling of application server. This feature saves development time and makes the development team to have one worry less to handle. Amazon and Google App Engine has this feature inbuilt, so they have the advantage. Heroku and Cloud Foundry do not have this feature. Fourth factor is auto-scaling of
database and Google App Engine is the only one to have this feature, though it has a limitation of only supporting a linear database solution (BigTable). Final factor in scalability is HTTP session management. This feature saves CPU and memory resources. Google App Engine and Heroku has the advantage in this factor.

<table>
<thead>
<tr>
<th></th>
<th>AWS Beanstalks</th>
<th>GAE</th>
<th>Heroku</th>
<th>Cloud Foundry</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scaling</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Built-in load balancer</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes(sticky sessions)</td>
</tr>
<tr>
<td>Custom domains</td>
<td>Yes</td>
<td>Yes(no naked)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Auto-scaling of app server</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Auto-scaling of database</td>
<td>No</td>
<td>Yes(BigTable NoSQL)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>HTTP session management</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

**Table 12. CBA analysis for Scalability(Advantages bolded)**

Concerning scalability Google App Engine has the most advantages. It has in-built load balancer, custom domains for scalability, auto-scaling of application server, auto-scaling of database and HTTP session management. Amazon comes second with custom domains, auto-scaling of application server and in-built load balancer. Heroku has custom domains and HTTP session management and Cloud Foundry has only in-built load balancer with sticky sessions. In this criteria, Google App Engine is the clear winner with these factors.

### 5.2.2 Service agreements and vendor lock-in

First factor from vendor lock-in is the presence of infrastructure lock. Lack of infrastructure lock is an advantage. When PaaS is not locked to any infrastructure, it gives the developers more flexibility in migration and increases portability. From the alternatives, Cloud Foundry is the only one which is not locked to any infrastructure. With a lot of tweaking and modifying the application source code, Google App Engine allows their applications to be moved to another infrastructure but officially they only work on Google's infrastructure. The second factor from this criteria is the possible lock to the platform. The way to measure the advantage in this is the rate of portability. The easier the porting to another platform is, the better. Out from the alternatives, Amazon has the highest portability rate. Cloud Foundry has moderate rate. Google App Engine and Heroku have low portability rates. Third factor in this criteria is the ease of migration applications to/from the platform. Easy migration brings more flexibility and options to the developers. The easier the migration is, the better. As with portability rate, Amazon has also an advantage in this factor. Migration is classified as hard in all the other PaaS alternatives.
The next factor is the amount of applications the platform allows to be developed on the platform. The higher the limit is, more longevity it brings to the platform as the developers do not need to worry about the amount of applications they can develop and change the platform because of it. Cloud Foundry has the advantage here with no limit in developed applications. Heroku comes second with the limit of 100 applications, followed by Amazon with 25 and Google App Engine with 10 applications. The final factor in this criteria is the data and database geographical locations. This is important factor for legal circumstances and customer agreements. The more options for location and freedom of deciding the location are the advantages. Amazon and Cloud Foundry get the advantage in this factor because, Amazon has locations in North and South America, EU(Ireland) and Asia, and Cloud Foundry is allowed to be deployed in any infrastructure. In addition, in Amazon the users can decide the location themselves, so AWS gets slight advantage over Cloud Foundry, even though Amazon's infrastructure is possible to use with CF also, but it it takes more work and configuring to do so.

<table>
<thead>
<tr>
<th>Vendor lock-in</th>
<th>AWS Beanstalks</th>
<th>GAE</th>
<th>Heroku</th>
<th>Cloud Foundry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infrastructure lock</td>
<td>Yes(AWS)</td>
<td>Officially Yes</td>
<td>Yes(AWS)</td>
<td>No</td>
</tr>
<tr>
<td>Portability rate</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
<td>Moderate</td>
</tr>
<tr>
<td>Ease of migration</td>
<td>Easy</td>
<td>Hard</td>
<td>Hard</td>
<td>Hard</td>
</tr>
<tr>
<td>Amount of apps</td>
<td>25</td>
<td>10</td>
<td>100</td>
<td>No limit</td>
</tr>
<tr>
<td>Data locations</td>
<td>US, EU, Asia, South America with user control</td>
<td>US, Europe</td>
<td>Amazon's locations with no control</td>
<td>Any infrastructure</td>
</tr>
</tbody>
</table>

**Table 13. CBA analysis for Vendor lock-in concerns(Advantages bolded)**

In this criteria both Amazon and Cloud Foundry has most advantages. From the importance of factors point of view in this thesis case, the advantages on Amazon outweigh the advantages on Cloud Foundry. Google App Engine and Heroku do not have any advantages in this criteria.

**5.2.3 Security**

The presence of in-built SSL encryption is the first factor in security criteria. SSL encryption makes the handling of critical data more safe. Out from the alternatives, Google App Engine and Heroku have in-built SSL in them. Heroku's SSL has to be paid so GAE gets the advantage here. AWS and Cloud Foundry do not have inbuilt SSL offerings. Next factor to be measured is the possibility to run the PaaS environment in private cloud configuration. The possibility of running in private cloud, gives more flexibility in security configurations and increases general security. Cloud Foundry is the only one which allows private cloud configuration out from the box. With add-ons,
private cloud configuration is also possible in Google App Engine. Cloud Foundry gets the advantage in this factor. Next factor is the possibility to run the PaaS in hybrid cloud environment. As in the private cloud factor, Cloud Foundry is the only one which supports hybrid cloud configurations from the box. Again, with add-ons, hybrid cloud environment is possible in GAE. AWS and Heroku do not support private or hybrid cloud environments.

The last factor in security is the authentication systems of the platforms. This factor is harder to measure because the security of authentication is hard to measure. To find out advantages in this category, the more certified security standards the platform supports the better. Also more authentication methods used, the better. From the alternatives, AWS has the most extensive authentication system with in-built authentication frameworks such as IAM and MFA(in-built frameworks for authentication, described in chapter 5.1.5), so it gets the advantage in this factor.

<table>
<thead>
<tr>
<th></th>
<th>AWS Beanstalks</th>
<th>GAE</th>
<th>Heroku</th>
<th>Cloud Foundry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In-built SSL</td>
<td>No</td>
<td>Yes(Paid)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Private cloud</td>
<td>No</td>
<td>No (possible with add-ons)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Hybrid cloud</td>
<td>No</td>
<td>No (possible with add-ons)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Authentication fw</td>
<td><strong>Extensive</strong>(IAM, MFA)</td>
<td>Limited to Google Accounts</td>
<td>Email/password, API token, SSH key</td>
<td>2 week security token, three different authentication and identification standards</td>
</tr>
</tbody>
</table>

Table 14. CBA analysis for Security(Advantages bolded)

In security, Cloud Foundry has most advantages, followed by AWS and Heroku. GAE has no advantages. Cloud Foundry's open nature makes handling the security issues more flexible and gives more control over it thanks to the possibility of private and hybrid cloud configurations. In GAE, the users have to rely on Google Accounts and Google's security measures completely.

5.2.4 Framework

First factor from framework is the supported languages. The alternative with most extensive language and framework support gets the advantage. More languages and frameworks gives more options and flexibility for the developers. Heroku has the most extensive set of languages and frameworks supported from the box. In addition Heroku allows build-packs for other languages. Cloud Foundry has also build-packs for other languages but it does not support as much language frameworks at default as Heroku. Second factor is the database framework. As with previous factor, the more database...
options there are, more it gives flexibility and options for the developers. Amazon has a clear advantage in this factor because it lets the users to decide what kind of database framework they want to use in their applications. Heroku and Cloud Foundry has build-packs and third-party add-ons for many database solutions but officially they support only the most common ones. GAE is limited to non-relational BigTable.

Next two factors, Graphical User Interface(GUI) and Web based monitoring are related to user experience and user friendliness. AWS and GAE has in-built GUI for the platform configurations and settings. Heroku does not have GUI per se, instead it offers a GUI plug-in for Eclipse SDK. Web based monitoring is offered in AWS, GAE and Heroku. As of now, Cloud Foundry does not offer GUI or web based monitoring. The following factor is the presence of command-line tools. Command-line tools give more configurability for the developers and it allows more specific control over the platform. Command-line tools are presented in each alternative. Final factor in this criteria is the presence of testing environment. Ideally the platform has an integrated build and test environment. This would enable the developers to continuously develop, build and test code in real environment. Unfortunately none of the alternatives has this feature. However, Heroku and Cloud Foundry has their solutions for testing. Heroku offers a lot of add-ons to help testing and Cloud Foundry can be tested locally with Micro Cloud(locally saved instance). Heroku gets the advantage with extensive add-on offerings which make various testing environments possible.

<table>
<thead>
<tr>
<th>Framework</th>
<th>AWS Beanstalks</th>
<th>GAE</th>
<th>Heroku</th>
<th>Cloud Foundry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported languages</td>
<td>Java(Apache Tomcat), PHP(Apache HTTP server), Python(Apache HTTP server), Node.js(Nginx), Ruby(Passenger),.NET (MS IIS)</td>
<td>Java, Python, PHP, Go</td>
<td>Ruby, Java, Python, Clojure, Scala, Node.js, Play, buildpacks for others</td>
<td>Java(Spring, Play), Scala(Lift), Groovy(Grails), Ruby(Rails, Sinatra), buildpacks for others</td>
</tr>
<tr>
<td>Database frameworks</td>
<td>Any</td>
<td>BigTable</td>
<td>PostgreSQL, add-ons(Amazon RDS, Hadoop, Memcache, etc.)</td>
<td>MySQL, Postgre, RabbitMQ, MongoDB, Redis, buildpacks for others</td>
</tr>
<tr>
<td>Graphical User Interface</td>
<td>Yes</td>
<td>Yes</td>
<td>No(Plugin for Eclipse)</td>
<td>No</td>
</tr>
<tr>
<td>Web based monitoring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Command line tools</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Testing environment</td>
<td>No</td>
<td>No</td>
<td>No, but easy with add-ons</td>
<td>No(Micro Cloud for local testing)</td>
</tr>
</tbody>
</table>

Table 15. CBA analysis for Framework(Advantages bolded)
In framework criteria, not many advantages can be found. Heroku has an advantage in supported languages and testing environment. Amazon has the most supported language and framework options, it allows most flexible database framework offerings and it has GUI, command-line toolset and web-based monitoring. Depending on the importance of the advantages or severity of disadvantages, Amazon or Heroku are the winners in this criteria. From the thesis case point-of-view Amazon has the most desired features(flexible database framework) in this criteria.

5.2.5 Pricing and pricing models

Pricing criteria has three factors to be measured. Cloud Foundry's prices and pricing models are yet to be decided so it will not be part of this criteria. First factor is free entries and extensiveness of them. The free tier in AWS has basic needs for a simple application but it's strength is that the platform service(Beanstalks) itself is free of charge. Users only pay for the other Amazon resources which are needed. GAE has more generous free tier overall than Amazon. Heroku has the most restricted free tier compared to AWS or GAE. Amazon's free platform service is a great strength but over all Google's free tier is more generous and allows more to be done free, so it gets the advantage. Second factor is pricing models. The one with most extensive pricing model alternatives gets the advantage. In addition of Beanstalks being free, Amazon has pay-per-usage and usage-based models possible right from the get go in any accounts. Google has usage-based pricing only possible in paid and premier accounts and GAE has also daily quotas(per-minute rate). Heroku has only time-based(duration of resources used) pricing and add-ons are charged monthly. Amazon offers most flexible and extensive pricing models and gets the advantage.

Last factor is entry level costs of the platform. It can be measured by researching how expensive would it be to develop first entry level commercial application on the platform. As described in the previous chapter, Amazon has tendency to lift the prices fast when more resources are needed, so it's entry level costs are classified as expensive. Heroku has low entry level costs because the 1 free dyno/month is generous enough to support decent sized applications. Google's generous free tier means that every entry level application is virtually free in GAE.

<table>
<thead>
<tr>
<th></th>
<th>AWS Beanstalks</th>
<th>GAE</th>
<th>Heroku</th>
<th>Cloud Foundry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pricing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Free entries</td>
<td>Free(PaaS), 750 hours, 30 Gb, 12 months duration only</td>
<td>Generous</td>
<td>750 hours(1 dyno/ month),</td>
<td>N/A (free)</td>
</tr>
<tr>
<td>Pricing models</td>
<td>Free(Beanstalks PaaS), pay-per-usage, usage-based</td>
<td>Free(no usage-based), Paid and Premier(usage-based), daily quotas</td>
<td>Time-based, add-ons paid monthly</td>
<td>N/A (free)</td>
</tr>
<tr>
<td>Entry level costs</td>
<td>Expensive</td>
<td>Free or very low</td>
<td>Low</td>
<td>N/A (infrastructure dependent)</td>
</tr>
</tbody>
</table>

Table 16. CBA analysis for pricing
Google App Engine has most advantages in pricing criteria with most generous offering free entries and lowest entry level costs. Amazon has the most extensive pricing models. As of now, Cloud Foundry is free of charge but things may change fast and as soon as it gets out of Beta. Costs of Cloud Foundry depend on the infrastructure and what kind of infrastructure services developers run below Cloud Foundry.

5.2.6 Summary

After the CBA analysis is done for each criteria and for each alternative, the advantages and dis-advantages can be listed for each alternative and find out which one should be the best alternative.

<table>
<thead>
<tr>
<th>Advantages of AWS</th>
<th>Advantages of GAE</th>
<th>Advantages of Heroku</th>
<th>Advantages of Cloud Foundry</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Highest portability</td>
<td>- Autoscaling of database</td>
<td>- Free inbuilt SSL-encryption</td>
<td>- Allows any data location</td>
</tr>
<tr>
<td>- Easiest migration</td>
<td>- Lowest entry level costs (~free)</td>
<td>- Most extensive language and framework support</td>
<td>- Infrastructure can be selected</td>
</tr>
<tr>
<td>- Most extensive authentication framework</td>
<td>- Most generous free entry</td>
<td>- Easiest testing</td>
<td>- Allows private and hybrid cloud configurations</td>
</tr>
<tr>
<td>- Any database support</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Free Platform service (Beanstalks)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 17. Advantages

From advantage point of view Amazon is the best alternative with this criteria and with these factors, since it have the most advantages over the other alternatives. Amazon with Beanstalks allows most flexible and extensive platform. Google's strengths are the low development costs. Cloud Foundry's strengths come from the open nature of the platform. To ensure the best alternative, dis-advantages will be listed also.

<table>
<thead>
<tr>
<th>Dis-advantages of AWS</th>
<th>Dis-advantages of GAE</th>
<th>Dis-advantages of Heroku</th>
<th>Dis-advantages of Cloud Foundry</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Most expensive entry level costs</td>
<td>- Fewest applications possible, largest application limitation</td>
<td>- No control of data locations</td>
<td>- Load-balancers with sticky sessions</td>
</tr>
<tr>
<td></td>
<td>- Most limited authentication and account framework</td>
<td>- Least extensive pricing models</td>
<td>- No custom domains for load-balancers</td>
</tr>
<tr>
<td></td>
<td>- Most limited language framework</td>
<td></td>
<td>- No web-based monitoring</td>
</tr>
<tr>
<td></td>
<td>- Most limited database framework</td>
<td></td>
<td>- No graphical interface(GUI)</td>
</tr>
</tbody>
</table>

Table 18. Dis-advantages
In addition to most advantages, Amazon has also fewest dis-advantages. The only dis-advantage in Amazon is that it is the most expensive to use in the context of the thesis case. In Heroku, users do not have control of the data locations. Heroku has also the fewest pricing alternatives possible. Google App Engine and Cloud Foundry have most dis-advantages. GAE's limitations come from the fact that it is so dependent on Google and its services. Cloud Foundry's dis-advantages clarify the fact that it is still unfinished and on Beta.

When comparing advantages to dis-advantages it comes even more clear that Amazon is the best platform out from the alternatives.
6. Conclusion

There were two research questions presented in this thesis. The first question was:

"Why choose Platform as a Service instead of building an own platform over a selected cloud infrastructure?"

The question can be answered through the literature review and there were many supporting aspects and also opposing aspects. The question is, is the supporting aspects enough to convince the deployment of the PaaS. From the chapter 3 we can list the supporting and opposing aspects. The first aspect that advocates the decision comes from the general benefit which is valid in cloud computing: There is no costs for servers and hardware(power bills, hw maintenance, additional IT staff etc.). This is naturally a great thing for a small company since every bit of saving costs is crucial. But this answer only valid for general decision of moving to the cloud. There are more specific aspects that supports PaaS deployment as a strategy.

If the development team’s core competence is not in cloud operation or cloud technical knowledge, PaaS is naturally a good option, since the cloud infrastructure and server configurations(up/down scaling) etc. are hidden and there is no need for infrastructure management. In the thesis contractor's case this aspect was one of the key reasons why PaaS was in the consideration in the first place. Next advocating aspect was that PaaS provides many important and crucial services such as run time environments, code repositories and other important middleware. This reason came also up in the contractor’s case, since unnecessary time consuming tasks such as system administration wanted to be minimized.

Next advocating aspect that was found from the literature review was affiliated to process and IT-management and to the upper level. It was said that utilization of PaaS makes business process modeling, work-flow planning and data modeling easier. While this may be true, this was only valid on PaaS providers such as LongJump, which is not really a PaaS(a development platform) at all, instead it is a set of management and planning tools implemented on a single cloud environment. From this aspect came one of the challenges of this thesis research and a problem in a definition of PaaS. In the contractor's case, this kind of “PaaS” was not the one which was needed.

Next benefit which the PaaS brings, was also affiliated in the first advocating aspect(economic). A common infrastructure on which the platform lie brings more flexibility and reduces complexity, since many applications can be run on the same platform which is running on a single infrastructure. It was also said that in PaaS, the first application in already cheaper to develop and it would get only cheaper when new applications are made. This was another crucial benefit in the contractor's case which advocated the PaaS strategy.

It was also suggested that PaaS benefits testers and testing. It was said that applications can be run with different configurations, multiple configurations and locations, which are impossible in local development environment. While this is true in many ways, as the PaaS alternative review tells, testing and test made environments are not necessarily
implemented well in PaaS. From the application development point of view, a great benefit in PaaS is that it allows deployment and testing in real environment very easy. This aspect was also discussed and stated important in the contractor's case.

There were also a number of benefits found which many are valid in cloud computing general such as ubiquitous access (available from anywhere), lower and predictable expenditures, but also PaaS specific benefits such as eliminated upgrade need, increased collaboration and co-operation (same platform tools etc. for each developer) and rapid scalability.

There were also found some possible reasons to decline the PaaS strategy found in this research. To begin with, PaaS is usually the most difficult to understand cloud service model. PaaS is often mistaken as IaaS or SaaS and unaware user may order wrong service by mistake. The services must be researched throughout and precisely. This came also up in the thesis research when the PaaS alternatives were selected. At first, LongJump was one of the alternatives but after researching it a bit, it was clear that it was not a platform for a development what was needed, instead it was a collection of services and applications on a single environment. Hopefully, PaaS definition will be more clear in the future.

One challenge in PaaS is also the need to know a lot of before hand what kind of technologies, methods and frameworks the application(s) will be used. Again this was also encountered in the thesis case because it was not completely decided or clear, which technologies or frameworks would be used. Because of this, the PaaS decision would dictate the technologies, which would be used in the end. It was also clear that PaaS has some constant challenges such as security concerns, platform limitations and vendor lock-in. In a small company, these are just the risks that must be taken because the benefits of platform outweigh these issues by far. The concerns above were also measured in the second research question.

There were also aspects that came up by the contractor and by the thesis researcher himself. First of all, the time was the biggest challenge and the PaaS strategy should speed up the application development by months. Costs were naturally one of the concerns which came up also and also affiliated to the reason above: “How fast the PaaS strategy would pay itself back?” Staff limitations and the lack of proper cloud development were also concerns to be taken account. PaaS strategy should support and diminish these limitations well enough.

The second research question was:

“What kind of criteria could be used when selecting a PaaS provider?”

The criteria was selected mainly from a point-of-view of a small company but in the end it should support any PaaS decisions. The reasons behind the criteria were found out researching the literature and some of them came from the aspects found out from the first research question. The criteria were scalability and it's aspects, vendor lock-in and service agreements, security, frameworks and supported languages, and the pricing models. Each criteria were split up in smaller factors, which were measured with CBA analysis. Each PaaS alternative were selected to support these criteria and each of the alternatives would have the factors easily and clearly measurable.

Choosing-by-advantage was selected as an analysis method and it was proposed by the contractor. After researching the method and theory behind, it was clear that it was the
most simple and effective way to choose the correct alternative in this case, because the
time frame in which the decision was made was very short and resources to research the
alternatives with more extensive methods too limited. Robust and sound decision had to
be made. After analyzing the criteria with the various factors, it was found out that out
from the PaaS alternatives, Amazon with Beanstalks was the one with most advantages
and thus was the best alternative and the one to be selected. The selection was
strengthen also by finding out the dis-advantages of each PaaS alternative in each
criteria.

The contractor had also some own suggestions and requirements for the platform and
most of them supported the criteria in the research. Naturally the costs were big concern
for the contractor since the natural limitation of financial resources of a new and small
company. Also the development team and their skills was one big decision influence.
There was also risk that the research would not be completed fast enough and in that
case the most common and well-known alternative would have been chosen. But in the
end the alternative found out in the CBA analysis was the one that was selected as it
supported the contractor's reasons and objectives.

One of the biggest challenges in this research was the lack of proper official scientific
literature found and a lot of time went trying to find of proper scientific references.
There were much literature about cloud computing found in general but literature that
focused on PaaS were harder to find. The reason behind this may be that PaaS as
concept is so new. There may be more literature about general software development in
cloud which may cover cloud software development platforms as well but those
platforms are not classified necessarily as PaaS. A large part of the references came
from web discussions, journals and blogs. When researching these references, a great
reference critic must be practiced. Still when there is no proper scientific literature
available, these are the ones to lean on. To validate the claims in the references, one
must find journals, blogs and discussions which support the same topic or matter and try
to find consistency between them. It also became clear that PaaS platforms and cloud
service models in general are a new topic. When the thesis research started in the latter
part of 2012, there were not many topics and discussion about these things. During the
2013, the articles and discussions particularly about PaaS increased a lot. A large part of
the references presented are from 2012 or 2013.

To sum up the research, it can be said that it was successful when measured with the
baseline of the research. The most important contribution was the criteria and how it can
be used as a framework for measuring PaaS platforms or even general cloud
development as well. The research also clarified what Platform as a Service really
means in the cloud computing stack and how it is distinguished from IaaS and SaaS
models. After the previous literature research, the defined criteria, PaaS comparison and
analysis, this should be clear and the thesis should bring real value in this area. Also it
should be a good future reference for anyone who will be researching and producing the
cloud computing topic of their own. It should also provide good precepts for companies
that are considering cloud computing business opportunities, especially those who want
to provide and develop cloud application services(SaaS). It became clear in the start of
this research that there were not much previous research or documents about PaaS
platforms or even cloud computing in general from a point-of-view of a small company.

From contractor's point of view, this research verified their desire to take PaaS as a
business strategy. The research also clarified and changed some company decisions and
dropped down many PaaS alternatives such as LongJump. Generally it made the PaaS
decision easier for the company. Also many aspects discussed in this research were
faced when the PaaS was initialized and the development started. Things like, pricing model concerns, scaling, security concerns, different cloud configurations and application deployment concerns were encountered early on.

There are many possible further research possibilities and future work concerning the topic. For example, a practical review for each of the PaaS alternative could bring more concrete evidence about the PaaS selection. This could be done for example, by hosting a simple application on each PaaS platform and run the application on simple configurations to measure the platform against the criteria. This could be used to validate the CBA analysis made in this research. Also, IaaS services could be compared to PaaS services, and how much does the development differentiate and how much additional work must be done when the application is hosted over an IaaS, instead of using a PaaS.
References


Amazon Web Services(c). (2013). AWS elastic beanstalk developer guide API version 2010-12-01 (API Version 2010-12-01 ed.) Amazon Web Services, Inc.


Badger, L., Grance, T., Patt-Corner, R., & Voas, J. (2012). Cloud computing synopsis and recommendations, recommendations of the national institute of standards and


# Appendix A. Choosing By Advantage analysis

<table>
<thead>
<tr>
<th></th>
<th>AWS Beanstalks</th>
<th>GAE</th>
<th>Heroku</th>
<th>Cloud Foundry</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scaling</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Built-in load balancer</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes(sticky sessions)</td>
</tr>
<tr>
<td>Custom domains</td>
<td>Yes</td>
<td>Yes(no naked)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Auto-scaling of app server</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Auto-scaling of database</td>
<td>No</td>
<td>Yes(NoSQL)</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Vendor lock-in</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Infrastructure lock</td>
<td>Yes(AWS)</td>
<td>Possible</td>
<td>Yes(AWS)</td>
<td>No</td>
</tr>
<tr>
<td>Portability</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
<td>Moderate</td>
</tr>
<tr>
<td>ease of migration</td>
<td>Easy</td>
<td>Hard</td>
<td>Hard</td>
<td>Hard</td>
</tr>
<tr>
<td>Amount of applications</td>
<td>25</td>
<td>10</td>
<td>100</td>
<td>No limit</td>
</tr>
<tr>
<td>Data locations</td>
<td>US(east/west), EU(Ireland), Asia, South America</td>
<td>US, Europe(three datastores)</td>
<td>AWS(no control)</td>
<td>Any</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSL inbuilt</td>
<td>No</td>
<td>Yes</td>
<td>Yes(paid)</td>
<td>No</td>
</tr>
<tr>
<td>Private cloud</td>
<td>No(VPC)</td>
<td>Yes(with add-ons)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Hybrid cloud</td>
<td>No</td>
<td>Yes(with add-ons)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Authentication</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supported languages</td>
<td>Java(Apache Tomcat), PHP(Apache HTTP server), Node.js(Nginx), Ruby(Passenger), .NET(MS IIS)</td>
<td>Java, Python, Node.js, PHP, Go</td>
<td>Ruby, Java, Python, Scala, Node.js, PHP, Play, buildpacks for others</td>
<td>Java(Spring, Play), Scala(Unt), Groovy(Grails), Ruby(Rails, Sinatra), buildpacks for others</td>
</tr>
<tr>
<td>Database frameworks</td>
<td>Any</td>
<td>BigTable</td>
<td>PostgreSQL, add-ons(Amazon RDS, Hadoop, Memcache, etc.)</td>
<td>MySQL, Postgre, RabbitMQ, MongoDB, Redis, buildpacks for others</td>
</tr>
<tr>
<td>GUI</td>
<td>Yes</td>
<td>Yes</td>
<td>No(plugin for Eclipse)</td>
<td>No</td>
</tr>
<tr>
<td>command line</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>web-based monitoring</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Testing environment</td>
<td>No</td>
<td>No</td>
<td>No, but easy testing and add-ons</td>
<td>No(Micro Cloud local testing)</td>
</tr>
<tr>
<td><strong>Pricing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Free entries</td>
<td>Free(PaaS), 750 hours, 30 Gb, 12 months duration only</td>
<td>Generous</td>
<td>750 hours(1 dyno/month),</td>
<td>Free</td>
</tr>
<tr>
<td>Pricing models</td>
<td>Free(PaaS), pay-per-usage, usage-based</td>
<td>Free(no usage-based), Paid and Premier(usage-based), daily quotas,</td>
<td>time-based, add-on(monthly)</td>
<td>N/A</td>
</tr>
<tr>
<td>Entry level costs</td>
<td>Expensive</td>
<td>Free</td>
<td>Low</td>
<td>Free(cloud setup dependent)</td>
</tr>
</tbody>
</table>