The Role of Service-Oriented Infrastructure

CSE 445/598 Distributed Software Development

Christian Korscheck
Department of Computer Science and Engineering
Ira A. Fulton School of Engineering
Arizona State University
christian.korscheck@asu.edu

***Abstract*— This paper analyses the requirements for the deployment of web applications from the point of view of an individual developer on the one hand and a company on the other hand and focuses on the service-oriented infrastructure. It compares the individual features and capabilities of two solutions: Amazon Web Services and Google App Engine. The paper uses this comparison to summarize the roles of both solutions in the evolvement of the web and shows up future prospects.**

***Index Terms*— Service-Oriented Infrastructure, Web Application, Amazon Web Services, Google App Engine**

I. Introduction

In the course of the introduction of web services a trend can be observed, away from conventional desktop applications towards web applications. Basically, there are two groups which help to advance this trend.

On the one hand, there are individuals who develop applications in their spare time as a hobby. They produce small web applications as an act of creativity and they benefit severely from the platform independency and the easy distribution of web applications. They can reach a widespread group of users, way easier than they could with traditional desktop applications. Users of their application only need a browser, but no specific operating system. Furthermore, if someone develops a useful web application, he could start his own company with this product.

On the other hand, there are companies who derive a great advantage from web applications. They save a lot of money, because no single-user licenses of desktop applications have to be bought. The applications are kept centralized on a service oriented infrastructure and therefore updates of the software can easily be applied during runtime. No distribution of software or patches is required.

Since every web service needs an infrastructural base, the developers of web applications have to think about the following questions: How much computing power is enough to handle simultaneous requests of the customers and keep the web application in service even in traffic hours? How can costs be kept short so that no money will be spend on unused computing power? What happens, if the demand of the web application rises and additional infrastructure must be provided?

These questions can't be answered easily. Cuil[1], a search engine developed by former employees of Google, was launched on July 28th, 2008. The servers crashed a few minutes after launch, because they could not handle all the visitors and search queries [1] [2]. 50 million queries on the first day were far more than expected. To address this problem, developers of web applications need flexible infrastructures, scalable with an attractive pricing model. Furthermore, if they don't have to care about the infrastructure by themselves, time can be spend on developing the actual service and therefore, even small companies can become a business rival for large corporations if they provide a good product on a stable infrastructure.

[1]http://www.cuil.com

This paper compares two solutions for developers of web applications: Amazon Web Services and Google App Engine. Both of them can be used as an infrastructural base for web applications. The advantages and disadvantages of both solutions are examined in the following.

II. Service-Oriented Infrastructure

A web service or web application consists of multiple layers.

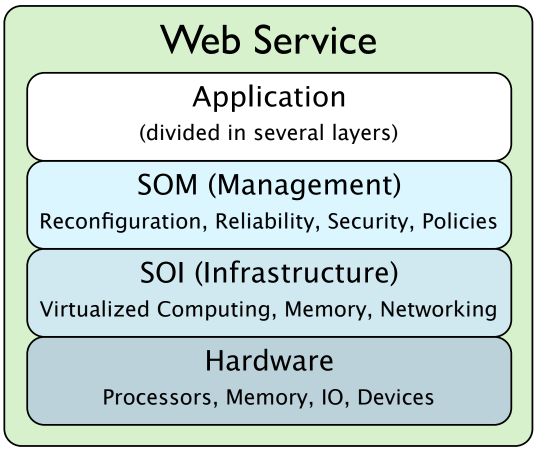

Web Service
Application
(divided in several layers)
SOM (Management)
Reconfiguration, Reliability, Security, Policies
SOI (Infrastructure)
Virtualized Computing, Memory, Networking
Hardware
Processors, Memory, IO, Devices

Fig. 1. "Components of a Web Service"

Figure 1 shows the structure of a web service according to a *Service-Oriented Enterprise* (SOE) framework [3]. The application layer is shown simplified, it usually can be divided into several layers. For the consideration of the infrastructural base of a web application, the main focus lies on the three underyling layers.

Those three layers have to work hand in hand to create a strong infrastructural base for web applications. The most important attributes of a *Service-Oriented Infrastructure* (SOI) are scalability, reliability and security [4].

The bottom layer is composed by the hardware itself. This can be a single server or a cluster of servers. The way how the hardware is connected to each other, how resources are shared or distributed and how computing is done efficiently, is managed by the overlying layer. The third layer is called *Service-Oriented Management* (SOM). Security policies and configuration issues are applied by this layer. It helps to manage the underlying hardware in a human-friendly way and is usually an application by itself.

These three layers are almost independent of the application layer. Often, it is important for the developer of an application to know about the hardware he uses and this is also true for web applications even if they are platform independent. But knowing about the hardware and managing it are two completely different tasks. Usually, developers of applications are not responsible for the management of the hardware. This task is done by other employees of the company or even by a third party. The management of hardware consumes time and money and is a real problem for startup companies which try to focus on the development of their application. As mentioned in the introduction, it is quite difficult to estimate the required computing power for a new application.

But also individuals who want to release their small applications to the public don't want to waste time on managing the hardware. Signing contracts for webspace, comparing prices and adjusting the hardware to possible rising demand of their application can be an exhausting task. Also, they don't want to spend more money on hardware than needed. If their application won't be used by users, they might want to suspend the service and cease the contract for the server lease which could consume even more time and money, depending on the type of contract.

The purpose of the following sections of this paper is the question, if the problem of managing hardware for web applications can be addressed by using computing power of large companies.

III. Amazon Web Services

Even if *Amazon Web Services* (AWS, [5]) also contain a collection of web service solutions [6], the main purposes of AWS are the infrastructure services. An API provides access to Amazon's infrastructure to allow customers of AWS to build their own web applications based on this infrastructure. Since Amazon provides a world wide online shop system, they spent millions of dollars on building the infrastructure for web-scale computing. The idea behind Amazon Web Services is to sell computing power based on the computing time and resources that are actually used by the customer. To achieve that goal and to satisfy different needs, they provide several services. Four of the most important services are introduced briefly in the following.

A. Amazon Elastic Computing Cloud

Amazon Elastic Computing Cloud (Amazon EC2, [7]) is a service that offers scalability in computing resources. The customer may quickly scale the used capacities, both up and down. This is achieved by providing different sizes of virtual machines and a front-end to manage those machines.

B. Amazon Simple Storage Service

Amazon Simple Storage Service (Amazon S3, [8]) is a service that offers online data storage. The design of this service aims to provide scalability, high availability and low latency at commodity cost. The customer can buy new storage space or release unused storage space and pays only for the storage space he really needs.

The organization of data is achieved by placing the data objects into *buckets*. Buckets can be accessed by an HTTP interface or a SOAP interface.

C. Amazon SimpleDB

Amazon SimpleDB [9] is a distributed database system. Access to the data is provided by web services. It supports general operations on the data with SQL-like query commands.

D. Amazon Simple Queue Service

Amazon Simple Queue Service (Amazon SQS, [10]) is a messaging service that supports communication between computer programs over the web.

These four services can be used as core services to build a web application. Amazon claims that all of these services are reliable, simple to use, scalable, secure und inexpensive. All services work hand in hand and supplement each other. It is not necessary to use all of these services. For instance, a customer could order only Amazon EC2, Amazon S3 and use a MySQL database instead of Amazon SimpleDB. Paying only for resources that are really used and providing easy scalability to adjust the hardware according to the real demand of the web application, makes Amazon Web Services attractive for startup companies. Larger corporations which switched to Amazon's infrastructure services report large savings [11], [12].

But there are also some drawbacks. Even if the management of the hardware becomes easier by administration tools and difficult jobs like load balancing between the servers are handled automatically, there still must be someone who accomplishs this task and manages the (virtualized) infrastructure.

The infrastructure is not completely save from failure. Outages and slowdowns occur like in every other infrastructure, too [13].

IV. Google App Engine

Google's approach [14] is quite different from Amazon Web Services. Where Amazon Web Services can be seen as a more traditional hosting service with a virtual machine setup, Google tries to provide a tightly integrated product. An API provides access to Google's infrastructure, but unfortunately the only supported programming language for web applications based on Google App Engine is Python. Since Google uses its own data storage and file system access, default Python operations can't be used. Instead, Google provides the *Datastore API* for storing data in a database. The data can be accessed by a SQL-like query language called GQL [15].

A. Datastore API

The scalability of Google App Engine benefits from removing JOINs on database tables. GQL statements can be performed on only one table. Simulating one-to-many or many-to-many relationships can be accomplished by calling according API functions. To switch from a relational database like MySQL to Google's Datastore, a paradigm shift is required.

B. Other APIs

At the time of this paper, Google App Engine supports six additional APIs.

- The Python Runtime, supports CGI, sandbox features, application caching and logging.
- Images API, provides image data manipulation as a service.
- Mail API, sends email from the web application.
- Memcache API, enables distributed memory cache.
- URL Fetch API, helps accessing other internet hosts from the web application.
- Users API, the interface to integrate Google Accounts into the application.

The community asks for more APIs (task scheduling is a heavily requested feature) and it is very likely that Google will provide more APIs in the future.

In contrast to Amazon Web Services, Google App Engine has many restrictions for developers and does not provide the same freedom [16]. But therefore, the infrastructure is managed automatically and does not need to be administrated by the developers themselves. Some of the restrictions are:

- Libraries that need direct disk access are not allowed in Python for Google App Engine.
- Sockets are disabled with Google App Engine.
- Subprocesses and threading is not available.
- Most C-based modules are disabled for security reasons.
- System calls are disabled and any third party packages which use any of the above features won't function with Google App Engine. This means third party database software such as MySQL or PostgreSQL will not work.

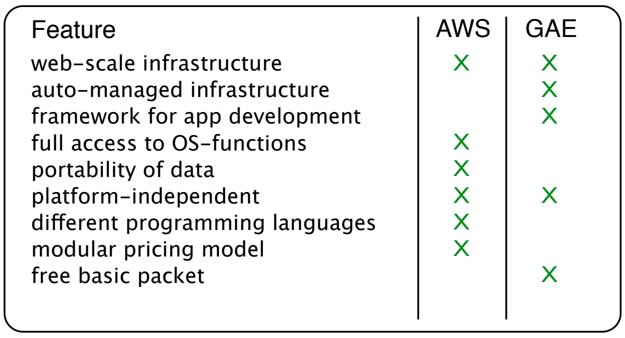
If these drawbacks are no obstacle for web application developers, they get a complete packet with access to Google's web-scale infrastructure and the same benefits as Amazon Web Services meaning scalability, reliability and security. The most attractive advantage especially for individuals is the basic packet which is completely free. There are some restrictions on this basic packet: The used space per application may not exceed 500 MB, an application may not have more than 10 GB of incoming bandwidth per day, an application may not have more than 10 GB of outgoing bandwidth per day, an application may not use more than 200 million megacycles of CPU per day and may not send more than 2,000 emails per day [17].

This offer helps heavily to make web application development attractive for individuals. If their application gets bigger, they might buy additional resources after time. At the time of this paper, Google does not support buying additional resources. This makes Google App Engine unattractive for large Web Applications, but it is very likely that Google will offer different packets in the future.

Google App Engine also has some disadvantages. The system is integrated as a whole, there are no single modules that can be selected by the developers of web applications. If a developer decides to create an application using Google App Engine, he gets the whole infrastructure access with all APIs, even if he does not want to send emails and therefore does not need the Mail API module. Therefore, Google App Engine can't be used just as a data storage system as Amazon S3. A customer always gets all modules.

Because Google does not allow third party software that requires access to the file system or needs system calls, basic databases like MySQL can't be used. If a developer decides to use Google App Engine, but wants to move his software (for instance a blog-system) to another hosting service after a few months, he can't use his data anymore, because his web application uses the proprietary database system of Google App Engine.

In addition, Google does not reveal the structure of Google App Engine. Many developers are interested in the structure of the platform they are developing for.



Feature	AWS	GAE
web-scale infrastructure	X	X
auto-managed infrastructure		X
framework for app development		X
full access to OS-functions	X	
portability of data	X	
platform-independent	X	X
different programming languages	X	
modular pricing model	X	
free basic packet		X

Fig. 2. "Feature Comparison"

Figure 2 compares features of both Amazon Web Services and Google App Engine with each other. Both solutions have features in common, but the differences show the distinct field of application. Amazon Web Services tries to address enterprise corporations whereas Google App Engine focuses on individuals who produce smaller web applications.

The next section draws a conclusion from this comparison and gives a vision for future development.

V. Summary and Future Prospects

As discussed earlier, every web application requires a solid fundament of infrastructure to run. The main purposes of such a fundament are scalability, reliability and security for both individual developers and companies.

The reasons for using web applications instead of desktop applications are savings, because no single-user licenses of desktop software must be bought and web applications can easily be updated due to its centralized nature. The software doesn't need to be distributed, it can be accessed by any modern browser. Also, no extra money is spend on unused computing power by the scalability of a service-oriented infrastructure.

Both, the individual developers and companies can focus on the development of their applications instead of spending time on managing the hardware. Managing a scalable hardware is a quite difficult task.

Further motivations of individuals for using service-oriented infrastructure are the attractive pricing models. Scalable hardware enables charging only for computing resources that are really needed. Often, Individuals want to be creative by developing a web application and don't want to waste time on managing hardware. Another important point is the option to move the web application to another hosting service or, if the demand for the web application increases, buy new infrastructure services in small steps according to the demand.

A company on the other hand focuses primarily on scalability, modularity of the infrastructure and expansion possibilities. They often need full access to operating systems and want to keep control of their data. It is very important for a company to have no limitations upwards.

In general, web applications and infrastructural services heavily support the idea of *users as contributors or developers* of the Web 2.0 vision. Individuals are motivated to create web applications according to their ideas and if the release process of their applications is easy, a lot of new ideas can be released to the public. Also, individuals with great projects can start their own company more easy if they don't have to focus on managing the hardware.

Especially Google's offer to create web applications and the opportunity to use Google's infrastructure for free, help to enhance the web as a platform. Providing a framework helps to create web applications faster, cheaper and keeps people focused on their ideas.

Amazon Web Services on the other hand contribute to the evolvement of the web by offering attractive services especially for larger corporations.

But there are still some drawbacks. Amazon Web Services hardly focuses on individuals. There are still too many tasks for managing the infrastructure. Often, a small web application does not need the scalability that Amazon provides.

Google App Engine on the other hand has heavy restrictions and is unattractive for larger web applications. Those restrictions are limiting the sustainability of web applications.

Another important aspect considers laws of authorship of code and data. Who has the rights for the web application that uses Google's framework? May Google use parts of the code? May Google collect data, if applications use the Users API which connects Google Accounts to the web application? May Google use the data which is generated or gathered by web applications to improve their own services? Those questions must be discussed and have to be reconsidered as the web evolves.

VI. Conclusion

Amazon Web Services and Google App Engine are different approaches which address different types of customers. Both of them help to improve the web as a platform, but both of them still have their flaws. Despite these flaws, both Google and Amazon provide interesting services with trend-setting technology which helps to encourage even more people to participate at the evolvement of the web.

References

[1] E. Schonfeld, "How to lose your cuil 20 seconds after launch," http://www.techcrunch.com/2008/07/29/how-to-lose-your-cuil-20-seconds-after-launch/.

[2] S. Hansell, "No bull, cuil had problems," http://bits.blogs.nytimes.com/2008/07/29/no-bull-cuil-had-problems/.

[3] T. Chen, *Distributed Service-Oriented Software Development*. Kendall Hunt Publishing Company, 2008.

[4] http://en.wikipedia.org/wiki/Service_Oriented_Infrastructure.

[5] http://www.amazon.com/AWS-home-page-Money/b/?node=3435361.

[6] http://solutions.amazonwebservices.com/connect/index.jspa.

[7] http://www.amazon.com/EC2-AWS-Service-Pricing/b/?node=201590011.

[8] http://www.amazon.com/S3-AWS-home-page-Money/b/?node=16427261.

[9] http://www.amazon.com/SimpleDB-AWS-Service-Pricing/b/?node=342335011.

[10] http://www.amazon.com/Simple-Queue-Service-home-page/b/?node=13584001.

[11] http://www.amazon.com/Success-Stories-AWS-home-page/b/?node=182241011.

[12] SmugBlog, "Amazon s3: Show me the money," http://blogs.smugmug.com/don/2006/11/10/amazon-s3-show-me-the-money/.

[13] SmugBlog.com, "Amazon s3: Outages, slowdowns, and problems," http://blogs.smugmug.com/don/2007/01/30/amazon-s3-outages-slowdowns-and-problems/.

[14] http://code.google.com/appengine/docs/whatisgoogleappengine.html.

[15] http://en.wikipedia.org/wiki/Google_App_Engine.

[16] http://code.google.com/appengine/kb/general.html#libraries.

[17] http://code.google.com/appengine/terms.html.