WHAT IS “CLOUD”?

Toni Ruottu, Eemil Lagerspetz and Sasu Tarkoma
University of Helsinki
P.O. Box 68 FI-00014 University of Helsinki, Finland
firstname.lastname@cs.helsinki.fi

Abstract

This paper follows the history of the term “cloud” from the beginning of the Internet to the era of cloud computing, and ponders its past and current meaning. We argue that outsourcing is the primary meaning of “cloud”. We discuss characteristics of cloud computing, in particular, elasticity and its significance in cloud computing. Elasticity is a key cost-cutting measure, especially for startup companies, but is not a requirement for cloud systems. We discuss the simple service layer model (the SPI model) and the role of software components on different layers. We refine the model to better capture real-world use cases. This is achieved by dividing the layers into solutions and solutions further into components.

Keywords: Cloud, Software, Definition, Layer model.

1 Introduction

This paper concentrates on cloud computing models and terminology. We will start with a short history of the term cloud. Then we will consider properties commonly associated with cloud computing. These include self-service, availability, resource pooling, measurability, and finally, elasticity, one of the central aspects in modern cloud computing. We'll move on to look at how the different levels of service may and may not be stacked. We take our shot at defining a more accurate stack model with the help of the concepts of Solutions and Components. Finally, we discuss future work, and how taking client-side terminals into account can lead into a more complete understanding of the cloud.

2 On the History of the Cloud

The Internet was built by connecting multiple private networks to each other. The routing was controlled by the participating organizations. With the later introduction of autonomous systems, the management of the network was outsourced to separate companies that concentrated on network maintenance. The private networks were abstracted from each other by "the cloud". System diagrams

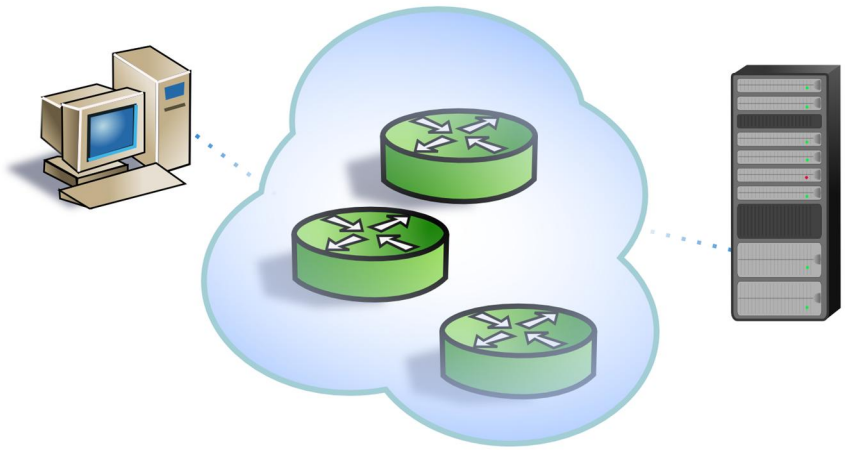

Figure 1. The network cloud separates endpoints from each other.

looked like Figure 1, where a network cloud separated the end-points from each other. Application protocol developers could concentrate on the end-points and stop doing general Internet maintenance.

When multiple computing systems in the network were connected to each other as grids, the end-point became a service or access conduit, instead of a physical machine with a single fixed location. One can think of the grids as the first step towards today's cloud platforms. A grid user would give a task to the whole grid, and not need to concern herself with the details of every participating machine or cluster.

With the introduction of web applications, such as Flickr or Gmail, the term cloud reappeared in a slightly different meaning. Storage of information was outsourced. It was in the cloud. Storing photographs and emails remotely made them universally accessible to the user, only protected by the user's login credentials. Storing the data in actively maintained data centers increased the durability of information. Some services pledged high reliability and an availability guarantee of 99% of the time each year [1]. The chance of data loss was higher on commodity hard disks.

With virtualization, one end-point can split into many virtual machines (VMs) for many customers. While grids are large endpoints with a known number of nodes and processing power, now the physical computer or end-point and its hardware have been outsourced. VMs are end-points that are outsourced to the degree that they only exist in software.

In recent years the term cloud has risen again, this time with the introduction of cloud platforms. Many different definitions of cloud platforms have been discussed [2]. The development and maintenance of application platforms was outsourced. This enabled smaller companies to create competitively scalable cloud systems without buying server hardware upfront. As a side-effect, we have seen improvements in the ways applications are developed for large numbers of servers, and deployed on them. From an end user's point of view the transition to cloud platforms was mostly invisible. For the users, operation of the service was not affected by who owned the server hardware, except in some cases with improved quality of service. However, developers loved the new outsourcing.

From these examples we draw that while the focus of cloud has been constantly changing, from networking through storage, hardware and applications to application platforms, the core meaning has stayed the same – the outsourcing of application independent problems. Regardless of the case, the uninteresting outsourced part of the system is visualized in drawings by hiding it behind a cloud symbol. It is in the cloud – it is someone else's problem.

3 Characteristics Associated with Cloud Platforms and Systems

The NIST definition of cloud computing [3] identifies five essential characteristics of cloud computing. These are on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. We will discuss each of these characteristics below.

On-demand self-service is largely self-explanatory. It requires that a user of a cloud system is able to provision capabilities automatically without human interaction with the service provider. This is a basic requirement of many computing systems. However, systems such as Amazon Web Services (AWS) EC2 require a credit card number that is verified. In the future, as cloud-based attacks increase, we expect that such services will require initial or even periodic human verifications of a customer's identity. The act of provisioning may still stay independent of human interaction.

Broad network access requires that a cloud system be accessible through the Internet without specialized hardware. While this characteristic is desirable, we feel that it is not strictly necessary of a cloud system. There could be a cloud backup and storage service that is ubiquitous on a service provider's computing devices, but inaccessible otherwise. It would be considered a cloud service, since it would outsource the backup and storage functionality. It would also allow use from multiple devices, and creation of accounts, files, and backups automatically.

Resource pooling requires that a cloud system is able to serve multiple customers using a multi-tenant model, with resources dynamically assigned according to customer demand. Dynamic allocation of resources is also one of the primary definitions for cloud systems in the taxonomy of Rimal et al. [25]. We argue that this characteristic is not essential, since the resource assignment model may not even be disclosed by the cloud service provider. For example, a user does not know how resources for her GMail account are provisioned, and does not need to know. Providers with large enough resource pools may opt for multi-user or multi-instance models for ease of design instead of a multi-tenant model.

Measured service means that resource usage throughout the system can be monitored and optimized, also allowing billing customers based on their resource and service usage. This is a characteristic common to services that allow provisioning of costly resources, such as virtual CPUs and memory capacity. Measured use of cloud storage and cloud email requires less granularity and optimization. Users usually pay a flat rate for services of this kind, but they are still cloud services, outsourcing a function and its maintenance, and making the life of users easier.

Elasticity is a concept where a service provider reserves extra capacity letting their customers scale up and down rapidly based on the timely need of the customer. Using an elastic third party cloud platform makes the up-front investment for a company wishing to deploy their first application much smaller than it is in the traditional model.

The focus of elasticity depends on the provided service. Infrastructure providers can have extra servers in store for their users while application platform providers can have capacity for handling sudden peaks in the load. Similarly cloud application providers need to be able to handle a sudden increase in the amount of simultaneous users when their customer have some crisis and they hire a large group of people to work on solving the crisis. If we go further to the end-user device level, a cloud terminal provider could make sure their customers have disposable computers and web browsers available when they are traveling.

Elasticity has been identified as a key property of cloud systems [3,4]. Many cloud platform vendors such as Amazon, OpenStack, and Eucalyptus promote the meaning of the word "Cloud" as a rapid deployment and easy maintenance vehicle for software or software development platforms. On a cloud platform a piece of software may acquire or release machines on-demand. This is to some degree the result of improvements in computer virtualization technologies, but also increase of computing power in general.

Traditionally new hardware needed to be manually brought up and added to the system. On a cloud platform, resource acquisition and release is quick, on the level of seconds and minutes, compared to the traditional days and months [5]. Being able to provisions new computers just by "saying so", is so convenient that it has lead some to believe that cloud and elasticity are the same concept. However large companies with sizable data centers may build cloud services on top of real hardware completely ignoring elasticity aspects. This mirrors the discussion on resource pooling. Elasticity properties may still be used for resource allocation

between different applications within the company, but it is not an essential property for building a cloud system.

While elasticity is an important part of many cloud systems we should not require for all cloud systems to be elastic. This leads into discussions about the minimum criteria of elasticity that is required for a system to be included in the category of cloud systems. If we do not consider all cloud systems to be absolutely elastic, we can ask questions like "How elastic is this platform?" Thus, elasticity has a place as a quality attribute for comparing the quality of competing cloud systems.

We conclude that cloud platforms and applications may be elastic and may have certain characteristics. However, the only ones that really define a cloud system are the outsourcing of a function, and the self-service aspect of usage.

4 The Naïve Service Layer Model

The different cloud service types are often presented as a three layered stack [25] (see Figure 2). The Infrastructure as a Service (IaaS) layer sits at the bottom and takes care of abstracting physical hardware. The Platform as a Service (PaaS) layer is in the middle and contains middleware that abstracts multiple computing nodes of the infrastructure into a single application development platform. The Software as a Service (SaaS) layer on the top makes cloud platforms useful to the end users, mapping computations into meaningful things such as an email or a photo album. The upper layers hide the lower layers to some degree, so a company providing software as a service, email for example, does not necessarily need to reveal the used platform to its customer.

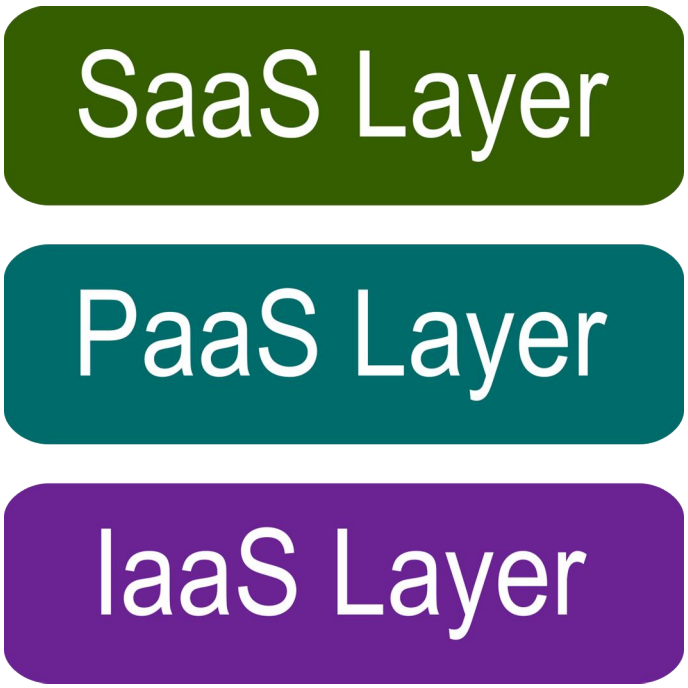

Figure 2. The naïve service layer model.

The layered stack representation implies that implementations of different layers are interchangeable, or that at least that it is possible to build software on a given layer using the abstraction of the layer below. The borders between layers symbolize interfaces that implementations of the different layers use to communicate with each other. The goal of the model is to define these interfaces to allow competition on providing implementations for different layers. However, different use cases demand different applications that need different types of platforms. The platforms in turn need different kinds of support from the infrastructure. Therefore, defining common interfaces is hard.

If we consider the service ecosystem, it may make sense to have competition in defining the interfaces, such as that between Open Cloud Computing Interface (OCCI) [6] and Elastic Compute Cloud (EC2) API [7], in addition to the competition in implementing them, as in

Eucalyptus [8], OpenStack [9] and Amazon Web Services (AWS) [10]. To make this happen, we need to allow incomplete, scenario-specific interfaces to be defined and implemented. Existing platforms already have heterogeneous interfaces, but if we refuse to define a common interface for them we need to update our theoretical model to match reality.

The problems in the stack model become more apparent when we try to imagine implementations of the layers. An implementation of the SaaS layer needs to contain all possible services needed by humankind. The PaaS layer needs to support all those services, and the IaaS layer needs to have the correct abstraction to support all possible services. Defining interfaces this serious may need to be an incremental process, and the interfaces may not become as strict as we might hope them to be.

A good way to test the model is to try placing some existing software products into the model. We spent some time trying to find the right place for database systems in the model. In theory, they could exist on the SaaS layer, if we considered a scenario where an advanced user uses a database directly from the command line, paying for the service that is a remotely usable database command line interface. In practice, a database would typically be used as a backend for some more specific application. It is too minimal to fill requirements of the PaaS layer as it is not a self-contained application development environment. Neither does it match the requirements of the IaaS layer, as it does not provide an environment that makes it possible to execute an arbitrary operating system.

Another problem in the model is its focus on software developers. Software as a Service is the top layer, but people do not interact with software. People interact with devices that project the software into the physical world. Thus for a full, turnkey cloud experience, the user needs someone to take care of client-side hardware. This requirement is not covered in the simple layer model.

The simple layer model has its shortcomings. The model encompasses all services, platforms, and infrastructure, and as such is too general for real use cases. These problems are tackled in detail below.

5 The Layer, Component and Solution Model

From our experiments with the simple stack model we learned that each layer may have different kinds of partial implementations to support different use cases. We decided we would call implementations of partial layers solutions. We also learned that there are pieces of software that can not really be said to alone support even one use case. We would call such pieces of software components. A solution or a component could live on any layer in the stack model. Depending on the use case, the layer a piece of software was placed on could be different.

While defining an interface for full layers seemed impossible, defining interfaces for solutions was possible, and had to some degree already happened. Multiple cloud platforms were accessible on the IaaS layer via the EC2 or OCCI API. The solution’s interface allows solutions on the layer above use the provided service without knowing its internal structure. While solutions may overlap in functionality and interfaces, their non-functional qualities can still be different. For example two storage services could have different reliability guarantees. Combining this with common interfaces would allow competition between solutions providing similar functionalities.

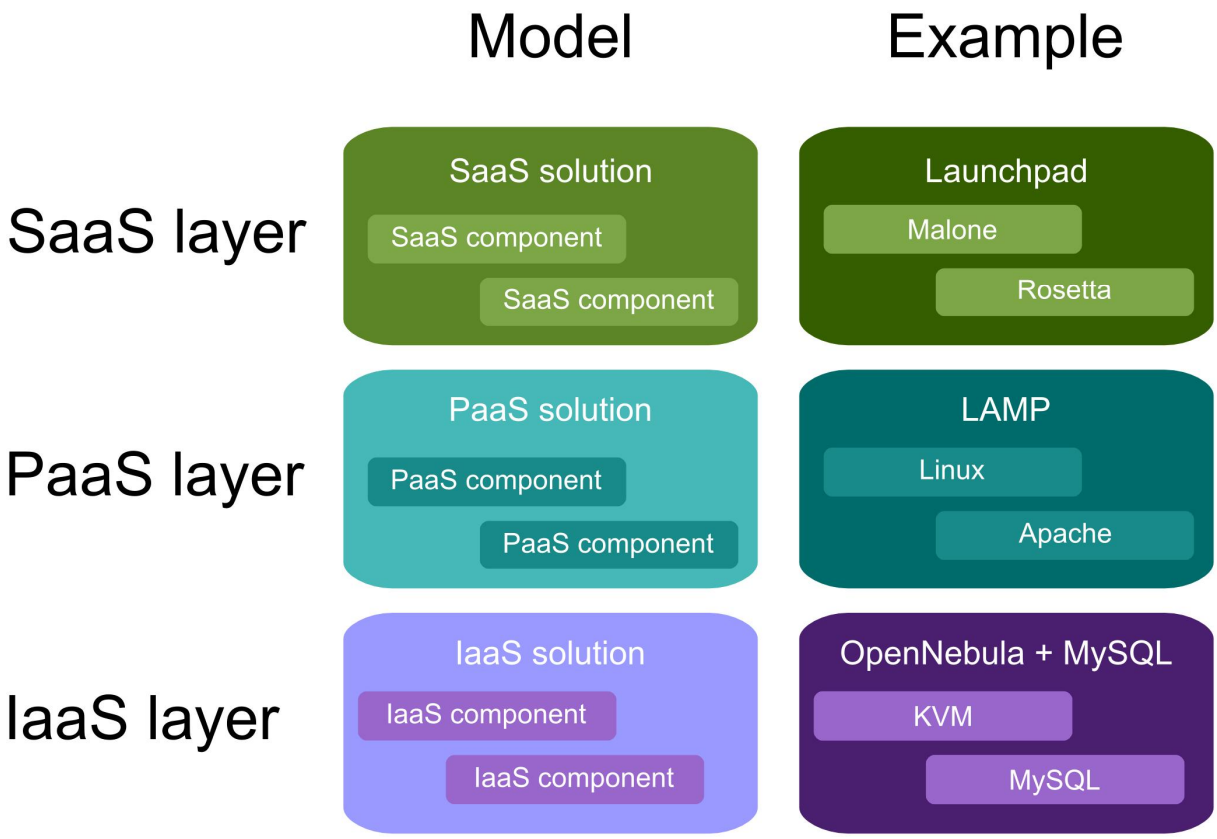

Figure 3. The refined model with a practical example.

When we talk about components and solutions we are talking about the technical parts that are used by a service provider to provide a service. For a piece of software to become part of a service, someone needs to run it. While the software may be publicly available, all components and solutions need not be published. Some companies may have their in-house developed parts, and that should not affect this classification.

In Figure 3, the service layer model has been refined into layers, solutions and components. The Figure shows the theoretical stack model next to some examples of solutions and components on different layers. The components within each solution in the Figure are examples and do not compose the whole solution. For example, the OpenNebula solution also uses SSH and may use storage services such as NFS instead of the local disk. The example consists of a solution for each of the three layers. On the top is a software project management solution called Launchpad [11]. It used by software developers for bug tracking, translation, and release planning. Malone is a bug tracking application while Rosetta is used in translation. On the PaaS layer we have a Linux, Apache, MySQL and Perl/Php/Python (LAMP) server. It consists of a Linux-based operating system, the Apache web server and other components. The LAMP server runs as a virtual machine on top of OpenNebula [12], on the IaaS layer. The database service used by the LAMP server is provided by an IaaS component, MySQL. OpenNebula manages the virtual machines using libVirt [13] and Kernel Virtual Machine (KVM) [14].

We use the LAMP stack as an example of a PaaS solution, since it is well-known and needed by many existing legacy solutions. In a way, it is not a “proper” PaaS solution; it performs poorly in abstracting the computer instances below the platform. Some modern alternatives include Heroku[15], Cloudfoundry [16] and AppEngine [17].

6 Conclusion

We considered the meaning of the term “cloud” in the history of the Internet and in current cloud computing. We learned how the term cloud is related to outsourcing parts of a computing system to third parties. We discussed the characteristics commonly associated with cloud computing, and concluded that only outsourcing and self-service are required for a system to be called a cloud system. We clarified the term elasticity, and learned that while it is central to some parts in cloud systems it is not synonymous with the term cloud. We have seen how a cloud service stack differs from traditional networking stacks with its vague interfaces. Deploying a cloud service restricts these interfaces, and limits the variation available for each layer after deployment. We noticed that the stack model may be improved by discussing specific parts of the layers. A layer is divisible into solutions that solve partial problems related to a layer, and to components used for building those solutions.

Acknowledgements

The authors would like to thank Ville Palkosaari and Sami Saada for constructive feedback on the original draft.

References

1. Glotzbach, M. (10/30/2008) What we learned from 1 million businesses in the cloud. The Official Google Blog. Retrieved from http://googleblog.blogspot.com/2008/10/what-we-learned-from-1-million.html
2. Vaquero L. M., Rodero-merino L., Caceres J., Lindner M. A Break in the Clouds : Towards a Cloud Definition. *Computer Communication Review*. 2009;39(1):50-55.
3. Mell P., Grance T. (2009). The NIST Definition of Cloud Computing. Special Publication 800-145 , The National Institute of Standards and Technology (NIST).
4. Armbrust M., Joseph A. D., Katz R. H., Patterson D. A. (2009). Above the Clouds : A Berkeley View of Cloud Computing. Technical Report No. UCB/EECS-2009-28 , University of California at Berkeley.
5. Coté , M. (6/2/2011) Cloud = Speed, or, How to do cloud marketing. Coté's People Over Process (Blog at RedMonk.com). Retrieved from http://www.redmonk.com/cote/2011/06/02/how-to-do-cloud-marketing/
6. The OCCI Cloud Protocol and API. http://occi-wg.org/
7. The EC2 API Reference. http://docs.amazonwebservices.com/AWSEC2/latest/APIReference/
8. Eucalyptus. http://www.eucalyptus.com/
9. OpenStack. http://www.openstack.org/
10. Amazon Web Services. http://aws.amazon.com/
11. Launchpad. https://launchpad.net/
12. OpenNebula. http://www.opennebula.org/
13. LibVirt. http://libvirt.org/
14. KVM. http://www.linux-kvm.org/
15. Heroku Cloud Application Platform. http://www.heroku.com/
16. CloudFoundry. http://www.cloudfoundry.com/
17. Google App Engine. http://code.google.com/appengine/
18. Kim, H. (6/30/2011) Summer travel with a Chromebook. Google Chrome Blog. Retrieved from http://chrome.blogspot.com/2011/06/summer-travel-with-chromebook.html
19. Google. (11/18/2009). What is Google Chrome OS? A video at http://www.youtube.com/watch?v=0QRO3gKj3qw
20. Amazon Kindle e-Reader. http://www.amazon.com/Kindle

21. The Django Web Framework. https://www.djangoproject.com/
22. The Xen Hypervisor. http://xen.org/
23. The Google High Replication Datastore.
    http://code.google.com/appengine/docs/python/datastore/hr/overview.html
24. Rimal, B.P. and Eunmi Choi and I. Lumb. A Taxonomy and Survey of Cloud Computing Systems.