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 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 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. Finally, we define a new computing business model. In this model, a company manages the entire computing experience of users. We conclude with discussion of the new business model and the possibilities it offers for users of the cloud.

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'll consider 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. Finally we discuss 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 looked like Figure 1, where a network cloud separated the end-

Figure 1. The network cloud separates endpoints from each other.
points from each other. Application protocol developers could concentrate on the end-points and stop doing general Internet maintenance.

With 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.

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 too much server hardware upfront. As a side-effect, we saw improvements in the ways applications were 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 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 Regarding Elasticity

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 console 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 late 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. 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.

4 The Naïve Service Layer Model

The different cloud service types are often presented as a three layered stack (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.

The layered stack representation implies that implementations of different layers are interchangeable. 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

![Figure 2. The naïve service layer model.](image-url)
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 a place for MySQL in the model. In theory, MySQL could exist on the SaaS layer, if we considered a scenario where an advanced user uses it directly from the command line, paying for the service that is a remotely usable MySQL command line interface. In practice, MySQL 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. It also cannot reflect services that revolve around terminal devices. These two 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 such as MySQL 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. Combining this with common interfaces would allow competition between solutions providing similar functionalities. For example two storage services could have different reliability guarantees.

After understanding that MySQL was a mere component and not a full layer or solution we revisited the question of placing it on one of the layers. If we ignore using MySQL as an end user application, it could be placed in the PaaS layer or in the IaaS layer. The IaaS layer provides persistence, while the PaaS layer provides dynamically scaled transient services. While a platform can provide persistence, this does not happen on the PaaS layer, but on the lower layers underneath. While it was theoretically possible that MySQL would be used for
transient data, that did not seem to make sense. Typically MySQL would be used for persistent storage of dynamic application data which makes it an IaaS component.

![Diagram](image)

**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 MySQL or Eucalyptus to actually become part of a service, some one needs to decide to run them. While MySQL and Eucalyptus are 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 system 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 Towards Computing as a Service

As the simple service layer model did not take into account client-side hardware, we'll use the opportunity to discuss changes needed for extending the model to cover that. A SaaS solution provides a piece of software available for use independent of location and terminal device. However, a user does require a terminal device with certain characteristics for gaining access to the service. The simple model is missing network connectivity, web browser software, and some client-side hardware. Users may want to outsource acquisition, maintenance and release of the software and hardware of the client-side terminal device as well. Outsourcing access completes a cloud system, as the entire computing experience is maintained by third parties. Figure 4 presents a view of the traditional computing model along with a user interacting with a cloud terminal.

A company providing users the full computing experience may take the approach further than mere computer rentals. It is no longer a question of devices, but a question of access to services, or making terminals available at locations where the users tend to be. Google for example has been experimenting with providing Chromebooks at some airports and hotels, for users who are traveling [18]. Chromebook users use Google's cloud services so the user data is not tied to the terminal that the user left at home for the trip. The Chromebooks are based on the Google Chrome OS, a web-centric operating system [19].

Providing users the physical hardware that they interact with also changes the network access picture slightly, as Internet access will be in a way a magical detail inside the product rather than a separate thing. Amazon has made experiments like these earlier by providing users of its e-reader Kindle [20] with free and mostly transparent Internet connectivity.
We call the layer that contains all these client-side technologies the Computing as a Service (CaaS) layer. A CaaS solution consists of a terminal device that is used for accessing applications that live on the Internet. An example of a full stack is shown in Figure 5. This could consist of a laptop with Chrome OS, an internal 3G modem, and some storage for caching. The applications running on the laptop are downloaded from the cloud and retrieved from a cache when Internet connectivity is not available. These could include Google Apps, such as Docs and Blogger.

All applications and created documents are synchronized with the cloud. If any program develops a fault, the CaaS solution provider can fix it on the server side, and updates are pushed to the terminal device. If the device itself develops a fault the user may be provided with a new working terminal device.

The PaaS solution includes Python and the Django [21] extension. The IaaS layer consists of a hypervisor such as Xen [22], and a data store, such as Google's High Replication datastore (HRD) [23]. Note that the IaaS components are examples, and not necessarily the components used by Google.

7 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 clarified the term elasticity, and learned that while it is central to some parts in cloud systems it is not exactly synonymous with the term cloud. We have seen how a cloud stack differs from traditional networking stacks with its vague interfaces. We also noticed that the stack 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. We discussed a computing business model where the missing link between a user and a piece of cloud software is closed. Finally we have learned that taking into account the physical terminals used by end users may lead to a more complete understanding of cloud-based systems. A mobile user may take advantage of computing terminals available at various locations, paying their service provider for access to software through these terminals. The involved service providers make terminals available at the supported locations. This frees users from maintaining and carrying computing devices with them.

Acknowledgements

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

14. KVM. http://www.linux-kvm.org/