What Is a "Cloud"?

The term "cloud" is often used when we know how to use something at a high level but we are conveniently shielded from the detail about how it actually works. We know that we have ways to work with and use things in the cloud, but we don't bother looking at what is going on inside. Sometimes we refer to this concept as "abstraction"; we deal with something complex and detailed in a very simple and abstract way and are unaware of the many intricate details that may be involved. For most of us, the automobile is an abstraction. We use a key, steering wheel, gearshift, gas pedal, and brake to drive around and we seldom worry about the thousands of highly specialized parts that are combined to produce the car. Of course, when your car breaks down or performs badly, those details begin to matter a lot. And when we do not have the skills to address the details, we hire someone (a mechanic), paying the professional to dig into those details and give us back a nicely working "abstraction"—and of course a hefty bill listing all the details within the abstraction that needed some work.

The Internet is another abstraction/cloud. The Internet is often represented as a cloud because although we know that all the computers are connected together, most people are generally unaware of the internal details of the links and routers that make up the Internet at any given moment. So the image of a cloud is a great way of representing all that hidden detail inside. We simply treat the Internet as an "abstraction" and use it—ignoring all the complex internal details.

Cloud or utility computing describes applications running on distributed, computing resources owned and operated by a third party. Some cloud computing providers offer general-purpose computing and storage capabilities, while others provide dedicated or specialized services.

End Users and Cluod Computing.

The end Users utilize the Software as a Service (SaaS) and Platform as a Service (PaaS) computing models. Users interact with applications over standard web browsers without concern about deployment or in-house administration.

- *Service (IaaS),)*
- *Platform as a Service (PaaS),)*
- *Software as a Service (Saas).).*

Each has unique features and benefits.

Any cloud-computing offering should have certain characteristics. Above all, it should be multitenant. A key component of a true cloud-computing platform, multitenancy is a type of software architecture where one instance of the offering is used to serve multiple tenants. The alternative, single tenancy, is how you're probably designing solutions today. Each customer (or business group, or client) gets her own server, database, application layer, and interface. In contrast, a multitenant application would have a single instance of all these layers and would partition the client's data programmatically. Multitenancy is a shared trait among offerings at the IaaS, PaaS, and SaaS layers.

At the lowest level, IaaS offers the physical infrastructure (or virtualized physical infrastructure) to tenants with the ability to pay for what they need in terms of computing power. Instead of purchasing servers, software, and physical location, a tenant of an IaaS offering can pay for these components as needed in a more subscription-based fashion. Leading IaaS vendors like Amazon.com offer “pay per CPU hour” pricing for Linux and Windows platforms. The servers are immediately available and you can spin up dozens of servers in a matter of minutes. At the highest level, SaaS, much like IaaS, offers solutions to the customer on a per-usage model. The major difference is that SaaS offerings completely abstract the physical and application layers from the end user or developer. For example, Salesforce.com (widely consider the best example of a SaaS offering) provides its own customizable user interface and proprietary programming language (Apex) but doesn’t expose to the end user the hardware or software layers that power the application. SaaS offerings have an important characteristic when it comes to application upgrades and maintenance: everything is centrally updated. So, when a new feature is released or a patch or upgrade is provided, it’s immediately available to all customers. In between IaaS and SaaS is the PaaS market. PaaS offers a bit more than IaaS, without providing an actual end-user product. PaaS components are typically building blocks or solution stacks that you can use to build your own applications**. This is where Google App Engine** fits in your cloud-computing portfolio. App Engine is a PaaS offering, currently supporting a Java and a Python runtime to build your scalable web applications without the need for complex underlying hardware and software layers. Google abstracts those layers and lets you concentrate fully on your application. PaaS does have its own set of challenges, however. With PaaS offerings, like App Engine and Force.com, you are restricted by a governor process or application quotas. PaaS governors protect the shared layers of the multitenant platform from being monopolized by one heavy application or runaway code. Application quotas, which Google defines for App Engine applications, define the daily-allotted amount of computing power, space, or bandwidth that any one application is allowed to utilize. With App Engine you have the option to pay for more power or space if needed.

Why Google App Engine? [1]

The stated purpose of the creation of Google App Engine is to make the Web better. By empowering millions of new software developers to produce new applications for the Web, Google is hoping to encourage the growth of the Web. Another advantage of Google letting us see and use their scalable infrastructure is that we will help them find ways to improve their infrastructure and make use of it in novel ways. By opening App Engine up to the public, thousands of new bright developers are poring over every aspect of Google App Engine, testing, checking, poking, prodding, finding problems, and suggesting fixes and improvements. This process greatly builds the community of knowledge around the Google software environment, while keeping Google’s costs low.

How the Cloud Runs Your Application

The best explanation of how the Google cloud works internally is that everything is “virtual.” If you look at wired land-line telephones, the prefix of a phone number generally indicates something about the physical geographic location of the phone. On the other hand, many cellular phones have the same prefix, regardless of their physical location. When you make a call to a cellular number from a wired phone, the call is routed across wires into the cellular network and then somehow the cellular network “tracks down” your cellular phone and routes the call to the appropriate cellular tower that is physically near your phone.
In a noncloud environment, the Internet works like the wired phone network. Web servers have a fixed and known location. They are assigned an Internet Protocol (IP) address based on that known location, such as 192.168.0.5. Your IP address is like a phone number—the entire Internet knows where that IP address is located and can route packets across the links that make up the Internet to get the data to that physical server. You also assign the server a domain name, like

www.drchuck.com, which lets Internet software use the Domain Name System (DNS) resolution to look up the numeric IP address (192.168.0.5) associated with the domain name as a convenience.
The Google cloud is more like a cellular network. Programs and data “roam around” the world and the web requests (like cellular calls) somehow find their way to your software, regardless of where in the world your software happens to be running. If you have an App Engine application running at a domain name of cloudcollab.appspot.com, Google can give this domain name a different IP address depending on what region of the world you are coming from. In the eastern United States, you might get one numeric IP address, and in South Africa, you might get another numeric IP address.
Once Google gets your request into the Google network, it figures out which data center(s) can run your application or perhaps which data centers are already running your application. It probably picks a data center that is some combination of reasonably close and not currently overloaded or perhaps the data center where the data for your application is stored. If all of a sudden your application experiences a spike of traffic in the United Kingdom, Google will likely copy your program and some of your data to one of its data centers there and start your application in that data center and pass the incoming requests from the United Kingdom to your program running in the United Kingdom.
If your application is very popular, it might be running in a number of different data centers at the same time. Or if your application gets 10 requests per hour, it probably is not running anywhere most of the time. When Google sees a request for your application, it starts up a single copy somewhere and gives the request to your application. Once your application finishes the request, it is shut back down to conserve resources.
The most important point of this is that your application has absolutely no idea if or when it is running, where geographically it is running, and how many copies of it are running around the world. Google takes care of all those details for you completely and (thankfully) hides them from you. Somehow the requests from your users make it to your application and back to the end user- Google takes all the responsibility for making this happen quickly and efficiently.
Running an application in the cloud is kind of like flying business class across the Pacific Ocean between Australia and the United States. You are vaguely aware that you are going really fast inside of a highly complex device that you barely understand. The pilots, crew, maintenance people, chefs, logistics staff, traffic controllers, and gate agents all are making sure that your trip happens efficiently and comfortably—and that it is uneventful. All you know is that you sit in a recliner, watch a movie, eat a nice filet mignon, have a glass of red wine, lay the seat flat, sleep for a few hours, and wake up refreshed on a different continent.

Components of an App Engine Application [2]

Building scalable applications with Google App Engine for Java (GAE/J) is similar to building Java applications in your typical on-premise environment with one large exception: there’s no need for the network, hardware, operating system, database, or application-server layers of the stack! With Google App Engine for Java, and Platform as a Service offerings in general, you can start to innovate and develop on your application right away and forget about the laborious tasks like setting up the OS and configuring the database. Google App Engine for Java provides a Java 6 JVM and a Java Servlet interface, and supports standard Java technologies like JDO, JPA, JavaMail, and JCache. Google App Engine for Java applications can be developed using the Eclipse IDE, and the Google Plugin for Eclipse even provides a local development server and deployment tools when you’re ready to go live with your App Engine application.
There are a few standard components to any Google App Engine for Java application. Some of these are optional if you’re using other technologies in their place. For example, the Users service is a great way to provide a trusted authentication mechanism to your user base. But, if you are

developing a Facebook application on the App Engine platform, you might be using Facebook Connect from Facebook's native authentication services, in which case the Users service might not be relevant. Table 1 gives you a quick look at the basic core components of a standard Google App Engine application.

GAE / J service	Description
JRE	Google App Engine for Java provides a standard Java 6 JVM and supports Java 5 and later. It also uses the Java Servlet standard, which allows you to serve JSP pages and standard files.
Datastore	Google App Engine for Java provides a persistent, scalable, fast datastore built on the DataNucleus Access Platform. You can use JDO and JPA to interact with the datastore and leverage the Memcache API for transient distributed storage for queries results and calculations.
Schedule Tasks	Google App Engine for Java, via the Administration Console, provides an interface for application owners to create and manage cron jobs on App Engine.
Java Tools	The Eclipse IDE, Google Plugin for Eclipse, the local development server, Apache Ant, and Google Web Toolkit (and much more) are available for use on Google App Engine for Java.

Table 1: Standard App Engine Technology Stack

Google App First Example.

Please download the document google application engine for java.pdf,
or visit the site http://worldims.weebly.com/tutorials

In this document you find the guidelòines to install Eclipse framework and for Signing Up for Google App Engine.

NEXT Tutorial. Servlet in Google APP Engine

by: Daniel Anselmo Bustos.

Bibliography.

[1] Using Google App Engine Charles Severance, Copyright © 2009. Published by O'Reilly Media, Inc.
[2] Beginning Java™ Google App Engine Copyright © 2009 by Kyle Roche and Jeff Douglas