Extracted from:

Code in the Cloud

Programming Google AppEngine

The Pragmatic Programmers

Code in the Cloud

Programming Google AppEngine

Mark C. Chu-Carroll

Edited by Colleen Toporek

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at

http://www.pragprog.com

Printed in the United States of America.

ISBN-10: 1-934356-63-8

ISBN-13: 978-1-934356-63-0

Printed on acid-free paper.

B1.0 printing, March 17, 2010

Version: 2010-3-15

Chapter 1

Introduction

Cloud computing is an innovative and exciting style of programming and using computers. It creates tremendous opportunities for software developers: cloud computing can provide an amazing new platform for building new kinds applications. In this chapter, we'll look at the basic concepts: what cloud computing is, when and why you should use it, and what kinds of cloud-based services are available to you as an application developer.

1.1 What's Cloud Computing?

Before we look at how to write cloud programs using AppEngine, let's start at the very beginning, and ask just what we mean by "cloud computing"? What is the *cloud*? How is it different from desktop computing, or old-fashioned client-server computing? And most importantly: why should you, as a software developer, care about the cloud? When should you use it, and what should you use it for?

The Cloud Concept

In the modern world of the Internet and the World Wide Web, there are thousands upon thousands of computers sitting in data centers, scattered around the world. We use those computers constantly—for chatting with other people, sending email, playing games, reading and writing blogs. When we're doing one of these everyday activities, we're accessing a program running on a server, using our browser as a client.

But where is the program actually running? Where is the data? Where is the server? They're *somewhere* out there. Somewhere, in some data

center, somewhere in the world. You don't know where, and more importantly, you don't care: there's absolutely *no reason* for you to care. What matters to you is that you can get to the program and the data whenever you need to.

Let's look at a simple example. A few years ago, I started writing a blog. When I got started, I used Google's "Blogger" service to write it. Every day, I would open up my web browser, go to http://goodmath.blogspot.com/admin, and start writing. When I finished, I'd click on the "Post" button, and the blog post would appear to all of my readers. From my point of view, it just worked. All I needed was my web browser and the URL, and I could write my blog.

Behind the scenes, Blogger is a complex piece of software run by Google in one of its data centers. It hosts hundreds of thousands of blogs, and those blogs are read by millions of users every day. When you look at it this way, it's obvious that the software behind Blogger is running on lots of computers. How many? We don't know. In fact, it's probably not even a fixed number—when not many people are accessing it, it doesn't need to be running on as many machines; when more people start using it, it needs more machines. The number of machines running it varies. But from the point of view of a user—whether that user is a blog author, or a blog reader—none of that matters. Blogger is a *service*, and it works. When I want to write a post, I can go to Blogger and write it, and when people go to my blog's web page, they can read it.

That's the fundamental idea of the cloud: programs and data are on a computer *somewhere* out there—and you neither know nor care where that computer is.

Why call this collection of resources a cloud? A cloud is a huge collection of tiny droplets of water. Some of those droplets fall on my yard, providing the trees and the lawn with water; some run off into the reservoir from which my drinking water comes. And the clouds themselves grow from evaporated water, which comes from all over the place. All I want is enough water in my yard to keep the plants alive, and enough in the reservoir so that I have something to drink. I don't care *which* cloud brings the rain; it's all the same to me. I don't care where on earth that water came from. It's all just *water*—the particular drops are pretty much exactly the same, and I can't tell the difference. As long as I get enough, I'm happy.

So think about the various data centers around the world—where companies have swarms of computers—as clouds. Lots of the biggest companies in network computing, including Google, Amazon, Microsoft, IBM, and Yahoo all have thousands of machines connected to networks, running all sorts of software. Each of those centers is a cloud, and each processor, each disk drive, is a droplet of water in that cloud. In the cloud world, when you write a program, you don't know what computer it's going to run on. You don't know where the disks that store the data are. And you don't need to care. You just need to know how many droplets you need.

Cloud to the Developer

Cloud computing is a fundamental change from how computers and software have worked in the past. Traditionally, if you wanted to run an application, you went out and bought a computer and software, set it up on your own premises, and ran your program. You needed to pick out which operating system you were going to run, handle the installation of your software, and maintain your computer—keeping track of software upgrades, security, backups, and so on.

With cloud computing, you don't do any of that. If you're a user of the cloud, you buy access to the application you want, and then connect to it from anywhere. Installing the software, maintaining the hardware and software where the application runs, making sure that the data is kept safe and secure—none of that is your concern. In the cloud, you buy software *as a service*. If you need more storage than a typical user, you buy extra storage from the service provider. If that means buying and installing a new disk drive, that's up to the provider. You just buy storage-as-a-service from them: how they provide it is their problem. You tell them *what you need*—in both the physical sense ("I need 1 terabyte of storage"), and in less tangible quality of service senses ("I need to guarantee that my storage is transactional, so that after I commit a change, data will never be lost"). You tell them your requirements, and some cloud provider will sell you a service that meets those requirements.

What this means is that when you're developing for the cloud, instead of buying a computer and running software on it, you break things down to basic building blocks, buy those pieces from service providers, and put them together however you want to build a system.

The building blocks are the *resources* you need to run a program or to perform a task. Resources include things like processing time, network bandwidth, disk storage, and memory. As a user of the cloud, you don't need to be concerned about where these resources are located. You know what you need, and you buy that from whomever can provide it to you most conveniently.

For developers, cloud computing introduces an even bigger change. When you develop for the cloud, you're not building a piece of software to sell to your customers—you're building a *service* for your customers to use. Understanding that difference is crucial: you need to design your application around the idea that it's a service you're going to provide to users, not a standalone application that they're going to install on their computers. Your customers are going to choose a service based on the tasks they want to accomplish, so your application needs to be designed with the task in mind, and you must provide it in the most flexible way possible.

For example, if you want to build a to-do list application for a desktop computer, it's a fairly straightforward process. There are lots of variations in how you can arrange the UI, but the basic idea of what you're building is obvious. You would build one UI—after all, why would you need more than one? And you'd build it mainly for a single user. If you are developing this to-do list application for the cloud, though, you'd want multiple UIs; at the very least, you'd want one UI for people accessing your service using their desktop computer, and one for people using a mobile browser on a cell phone. You'd probably want to provide an open interface that other people could use for building clients for other devices. And you'd need to design it for multiple users: if you put an application in the cloud, there's only one program, but it can be used by lots of people. So you need to design it around the assumption that even if users never work together using your application, it's still a multi-user system.

For developers, the most exciting aspect of cloud computing is its *scalability*. When you're developing in the cloud, you can write a simple program to be used by one or two people—and then, without ever changing a line of code, that program can scale up to support millions of users. The program is scale-neutral: you write it so it will work equally well for one dozen users or one *million* users. As you get more users, all you need to do is buy more resources—and your program will just work. You can start with a simple program running on one server somewhere

in the cloud, and scale up by adding resources until you've got millions of users.

Cloud Versus Client-Server

In many ways, the basic style of development for cloud-based software is similar to programming for client-server computing. Both are based on the idea that you don't really run programs on your own computer. Your computer provides a window into an application, but it doesn't run the application itself. Instead of running the program on your computer, all you do on your own computer is run some kind of user interface. The real program is running somewhere else, on a computer called a *server*. You use the server because for whatever reason, the resources necessary to run the program aren't available on your local computer: it's cheaper, faster, or more convenient to run the program somewhere else, where the necessary resources are easy to obtain.

The big difference between cloud and client-server development is in what you know: in traditional client-server systems, you might have a specific computer that is your server, and that's where your stuff is running. The computer may not have be sitting on your desk in front of you, but you know where it is. For example, when I was in college, one of the first big computers I used was a Vax 11/780, named nicknamed "Gold." Gold lived in the Rutgers university computing lab in Hill Center. I used Gold pretty much daily for at least a year before I actually got to see it. The data center had at least 30 other computers: several DEC 20s, a couple of Pyramids, an S/390, and a bunch of Suns. But of those machines, I specifically used Gold. Every program that I wrote, I wrote specifically to run on Gold, and that's the only place that I *could* run it.

In the cloud, you aren't confined to a specific server. You have computing resources—that is, someone is renting you a certain amount of computation on some collection of computers somewhere. You don't know where they are; you don't know what kind of computers they are. You could have 2 massive machines with 32 processors each, and 64 gigabytes of memory; or they could be 64 dinky little single-processor machines with 2 gigabytes of memory. The computers where you run your program could have great big disks of their own, or they could be diskless machines accessing storage on dedicated storage servers. To you, as a user of the cloud, that doesn't matter. You've got the resources you pay for, and where they are makes no difference, as long as you get what you need.

When to Develop for the Cloud

So, now you know what the cloud is. It's a revolutionary way of thinking about computing; it's a universe of servers that you can build an application on; it's a world of services that you can build, or that you can use to build other things. Now, the question is, when should you use it?

You can write almost any application you want in the cloud. In fact, many people strongly believe that *everything* should be in the cloud—that there's no longer any reason to develop applications for standalone personal computers. I don't go quite that far: many applications are well-suited to the cloud, but that doesn't mean that it's the ideal platform for everything. You *can* build anything as a service in the cloud, but it might be a lot harder that developing it as a standalone application.

There are three kinds of applications that it makes sense to build in the cloud:

Collaborative applications. If the application you're building will be used by groups of people to work together, share data, communicate, or collaborate, then you really should build that application in the cloud. Collaboration is the cloud's natural niche.

Services. If you ask "What does my application do?" and the most natural answer sounds like a *service*, then you're looking at a cloud application. The difference between an application and a service can be subtle—you *can* describe almost anything as a service. The key question here is what's the *most natural* description of it? If you want to describe the desktop iTunes application, you could say: "It lets people manage their music collections," which does sound service-like. But it leaves out the key property of what the iTunes desktop application does: it manages a collection of music *files* on the users computer, and lets them sync that music with their iPod using a serial cable. Described the latter way, it's clear that it's a desktop application, not a cloud application.

On the other hand, if you take a look at something like eMusic, you'll come to a different conclusion. eMusic is a subscription-based website that lets users browse an enormous collection of music, and buy a certain number of songs per month. eMusic is clearly a service: it lets people search through a library of hundreds of thousands of musical tracks, providing them with the

ability to listen to snippets, read reviews, comment on things that they've listened to, get suggestions for new things based on what they like, and ultimately select things to purchase. That's clearly a service, and it makes sense to keep it in the cloud.

Large computations. Is your application intended to perform a huge computation, which you could never afford to do if you needed to buy your own computers to run it? If so, the cloud allows you to purchase time on a server farm of computers in an affordable way, and run your application. This is great for people like genetics researchers, who need to run massive computations, but don't have the money or other resources to set up a dedicated data-center for their computations. Instead, they can purchase time on commercial data-centers, which they share with many other users.

1.2 Cloud Computing Programming Systems

There are multiple ways of programming the cloud. Before we start actually writing programs, we'll take a quick look at a few examples, to give you a sense of what the options.

Amazon EC2

Amazon provides a variety of cloud-based services. Their main programming tool is called EC2, Elastic Computing Cloud.

EC2 is really a family of related services. Compared to AppEngine, which provides a single, narrowly focused suite of APIs, EC2 is completely agnostic about programming APIs. It provides *hundreds* of different environments: you can run your application in EC2 using Linux, Solaris, AIX, or Windows Server; you can store data using DB2, Informix, MySQL, SQL Server, or Oracle; you can implement your code in Perl, Python, Ruby, Java, C++, or C#; you can run it using IBM's Websphere or sMash, Apache JBoss, Oracle Weblogic, or Microsoft IIS. Depending on which combination you prefer, and how much of each kind of resource (storage, CPU, network bandwidth), the costs vary from $0.10 per CPU hour and $0.10 per gigabyte of bandwidth to around $2,000 for a dedicated server for one year.

Amazon S3

Amazon provides another extremely interesting cloud service, which is very different from the other cloud offerings. S3, simple storage services, is a pure storage system. It doesn't provide the ability to run programs; it doesn't provide any filesystem; it doesn't provide any indexing. It's pure block storage: you can allocate a chunk of storage that has a unique identifier, and then you can read and write bytes from that chunk using its identifier.

A variety of systems have been created that use S3 for storage: web-based filesystems, native OS filesystems, database systems, and table storage systems. It's a wonderful example of the cloud's resource-based paradigm: the computation involved in storage is completely separated from the actual data storage itself. When you need storage, you buy a bunch of bytes of storage space from S3. When you need computation, you buy EC2 resources.

S3 is a really fascinating system. It's very focused: it does exactly one thing, and it does it in an incredibly narrow way. But in an important sense, that's exactly what the cloud is about. S3 is a perfectly focused *service*: it stores bytes for you.

S3 charges are based on two criteria: how much data you store, and how much network bandwidth you use storing and retrieving your data. Amazon currently charges 15 cents per gigabyte per month, and about 10 cents per gigabyte uploaded, 17 cents per gigabyte downloaded.

IBM Computing on Demand

IBM provides a cloud service platform based on IBM's suite of web service development that uses Websphere, DB2, and Lotus collaboration tools. The environment is the same as the IBM-based environment on EC2, but it runs in IBM's data centers, instead of Amazon's.

Microsoft Azure

Microsoft has developed and deployed a cloud platform called Azure. Azure is a Windows-based platform that uses a combination of standard web services technologies (such as SOAP, REST, Servlets, and ASPs), and Microsoft's proprietary APIs like Silverlight. As a result, you get the ability to create extremely powerful applications that look very much like standard desktop applications. But the downside is it's closely tied to the Windows platform, so the application clients run primarily on Windows. While there are Silverlight implementations for

other platforms, the applications tend to only be reliable on Windows platforms, and only fully functional in Internet Explorer.

So that's the cloud. Now that we know what it is, we're going to start learning about how to build applications in the cloud. Google has put together a really terrific platform, called AppEngine, for you to build and run your own cloud applications.

In the rest of the book, we're going to look in detail at the key pieces of how to build cloud-based web applications. We'll start off working in Python. Python's great for learning the basics: it lets you see what's going on, and it makes it easy to quickly try different approaches and see what happens.

We'll go through the full stack of techniques that you need for building an AppEngine application in Python, starting with the basic building blocks: HTTP, services, and handlers. Then we'll look at how you work with persistent data in the cloud, using the AppEngine datastore service. And then, we'll look at how to build user interfaces for your applications using HTTP, CSS, and AJAX.

From there, we'll leave Python, and move into Java. For building complex applications, Java can be a lot more convenient. And AppEngine provides access to an absolutely brilliant framework called GWT, which abstracts away most of the boilerplate plumbing of a web-based cloud application, and allows you to focus on the interesting parts. We'll spend some time learning about how to build beautiful user interfaces using GWT, and how to do AJAX style communication using GWTs remote procedure call service.

Finally, we'll spend some time looking at the most complicated aspects of real web development. We'll look at the details of how you can do sophisticated things using the AppEngine datastore service; how to implement server-side processing and computation using things like cron; and how to integrate security and authentication into you AppEngine application.

In the next chapter, we'll start this journey through AppEngine, by looking at how to set up an AppEngine account, and how to set up the software on your computer for building, testing, and deploying AppEngine applications written in Python.

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Code in the Cloud's Home Page
http://pragprog.com/titles/mcappe
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragprog.com/news
Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/mcappe.

Contact Us

Online Orders:	www.pragprog.com/catalog
Customer Service:	support@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com
Contact us:	1-800-699-PROG (+1 919 847 3884)