amazon web services™

sitepoint®

HOST YOUR WEB SITE IN THE CLOUD

AMAZON WEB SERVICES MADE EASY

BY **JEFF BARR**

SCALABLE, REDUNDANT, AND RELIABLE HOSTING AT A FRACTION OF THE PRICE!

Take Control of the Cloud with Amazon and SitePoint!

These sample chapters will help you get started in the cloud.

Cloud computing is fast becoming the norm for hosting solutions—regardless of whether the business is large or small—so familiarity with the cloud is an essential skill for developers. Web professionals can now save time hacking through documentation and teach themselves how to make the most of this hosting offer with this fantastic step-by-step guide.

"Host Your Web Site in the Cloud: Amazon Web Services Made Easy" reveals how web developers can learn the skills needed to set themselves up with a reliable, scalable, and economical hosting solution.

Here are a few reasons why you should sail up into the cloud:

- Access to a cost-effective server and storage solution
- Offers immediate and reliable server support and scalability
- Provides big savings on time, money, and resources
- Boosts your resume by adding cloud computing to your skill set

Grab yourself a printed copy for only US$39.95 today here[1].

If you'd prefer the electronic version, you can instantly download it for only US$29.95 (includes PDF, EPUB, and MOBI) here[2].

As always, this book is covered by our money-back guarantee. We're sure you'll love this book, but if for any reason you don't, simply return it for a full refund (less postage).

[1] https://sitepoint.com/bookstore/go/263/1ecd9b
[2] https://sitepoint.com/bookstore/go/264/1ecd9b

What's in this excerpt?

Preface

Chapter 1: Welcome to Cloud Computing

In this chapter, you'll learn the basics of cloud computing, and how it both builds on but differs from earlier hosting technologies. You will also see how organizations and individuals are putting it to use.

Chapter 2: Amazon Web Services Overview

This chapter moves from concept to reality, where you'll learn more about the fundamentals of each of the Amazon Web Services. Each web service is explained in detail and key terminology is introduced.

Chapter 3: Tooling Up

By now you're probably anxious to start. But before you jump in and start programming, you'll need to make sure your tools are in order. In Chapter 3, you'll install and configure visual and command line tools, and the CloudFusion PHP library.

Chapter 4: Storing Data with Amazon S3

In Chapter 4, you will write your first PHP scripts. You will dive head-first into Amazon S3 and Amazon CloudFront, and learn how to store, retrieve, and distribute data on a world scale.

Index

What's in the rest of the book?

Chapter 5: Web Hosting with Amazon EC2

Chapter 5 is all about the Elastic Compute Cloud infrastructure and web service. You'll see how to use the AWS Management Console to launch an EC2 instance, create and attach disk storage space, and allocate IP addresses. For the climax, you'll develop a PHP script to do it all in code. To finish off, you'll create your very own Amazon Machine Image.

Chapter 6: Building a Scalable Architecture with Amazon SQS

In this chapter, you will learn how to build applications that scale to handle high or variable workloads, using message-passing architecture constructed using the Amazon Simple Queue Service. As an example of how powerful this approach is, you'll build an image downloading and processing pipeline with four queues that can be independently assigned greater or lesser resources.

Chapter 7: EC2 Monitoring, Auto Scaling, and Elastic Load Balancing

Chapter 7 will teach you how to use three powerful EC2 features—monitoring, auto scaling, and load balancing. These hardy features will aid you in keeping a watchful eye on system performance, scaling up and down in response to load, and distributing load across any number of EC2 instances.

Chapter 8: Amazon SimpleDB: A Cloud Database

In Chapter 8, you'll learn how to store and retrieve any amount of structured or semi-structured data using Amazon SimpleDB. You will also construct an application for parsing and storing RSS feeds, and also make use of Amazon SQS to increase performance.

Chapter 9: Amazon Relational Database Service

In Chapter 9, we'll look at Amazon Relational Database Service, which allows you to use relational databases in your applications, and query them using SQL. Amazon RDS is a powerful alternative to SimpleDB for cases in which the full query power of a relational database is required. You'll learn how to create database instances, back them up, scale them up or down, and delete them when they're no longer necessary.

Chapter 10: Advanced AWS

In this introspective chapter, you'll learn how to track your AWS usage in SimpleDB. You'll also explore Amazon EC2's Elastic Block Storage feature, see how to do backups, learn about public data sets, and discover how to increase performance or capacity by creating a RAID device on top of multiple EBS volumes. Finally, you will learn how to retrieve EC2 instance metadata, and construct system diagrams.

Chapter 11: Putting It All Together: CloudList

Combining all the knowledge gained from the previous chapters, you'll create a classified advertising application using EC2 services, S3, and SimpleDB.

Host Your Web Site in the Cloud: Amazon Web Services Made Easy

by Jeff Barr

Program Director: Lisa Lang
Technical Editor: Andrew Tetlaw
Technical Editor: Louis Simoneau
Editor: Kelly Steele
Expert Reviewer: Keith Hudgins
Printing History:
First Edition: September 2010

Chief Technical Officer: Kevin Yank
Indexer: Fred Brown
Cover Design: Alex Walker

Notice of Rights

Notice of Liability

Trademark Notice

Helmet image on the cover is a Davida Jet and was kindly provided by http://motociclo.com.au.

Published by SitePoint Pty Ltd
Web: www.sitepoint.com
Email: business@sitepoint.com

ISBN 978-0-9805768-3-2
Printed and bound in the United States of America

Chapter 1

Welcome to Cloud Computing

One or two office moves ago, I was able to see Seattle's football and baseball stadiums from the window of my seventh-floor office. Built side-by-side during an economic boom, these expensive and high-capacity facilities sit empty for the most part. By my calculations, these buildings see peak usage one percent of the time at most. On average, they're empty. Hundreds of millions of dollars of capital sit idle. I use this stadium analogy—and have done so many times over the last few years—to help my audiences understand the business value of cloud computing.

Now, instead of a stadium, think of a large-scale corporate data center. It's packed with expensive, rapidly depreciating servers that wait, unutilized, for batch processing jobs, large amounts of data, and a flood of visitors to the company web site. That's because matching predictions and resources for web traffic has historically been problematic. Conservative forecasts lead to under-provisioning and create the risk of a "**success disaster**," where a surge of new users receive substandard service as a result. Overly optimistic forecasts lead to over-provisioning, increased costs, and wasted precious company resources.

As you'll see in this book, cloud computing provides a cost-effective and technically sophisticated solution to this problem. Returning to my opening analogy for a

minute, it's as if a stadium of precisely the right size was built, used, and then destroyed each week. The stadium would have just enough seats, parking spaces, restrooms, and additional facilities needed to accommodate the actual number of attendees. With this scenario, a stadium fit for 50 people would be just as cost-effective as one built for 50,000.

Of course, such a situation is impractical with stadiums; custom, just-in-time resource instantiation is, on the other hand, perfectly reasonable and practical with cloud computing. Data processing infrastructure—servers, storage, and bandwidth—can be procured from the cloud, consumed as needed, and then relinquished back to the cloud, all in a matter of minutes. This is a welcome and much-needed change from yesterday's static, non-scalable infrastructure model. Paying for what you actually need instead of what you *think* you might need can change your application's cost profile for the better, enabling you to do more with less.

Avoiding a Success Disaster

Imagine you're a budding entrepreneur with limited resources. You have an idea for a new web site, one you're sure will be more popular than Facebook[1] or Twitter[2] before too long. You start to put together your business plan and draw a chart to predict your anticipated growth for the first six months. Having already run prototypes of your application and benchmarked its performance, you realize that you'll have to purchase and install one new server every month if all goes according to plan. You never want to run out of capacity, so you allow for plenty of time to order, receive, install, and configure each new server. Sufficient capacity in reserve is vital to handle the users that just might show up before your next server arrives; hence, you find you're always spending money you lack in order to support users who may or may not actually decide to visit your site.

You build your site and put it online, and patiently await your users. What happens next? There are three possible outcomes: your traffic estimates turn out to be way too low, just right, or way too high.

Perhaps you were thinking smallish, and your estimate was way too low. Instead of the trickle of users that you anticipated, your growth rate is far higher. Your initial

[1] http://facebook.com/
[2] http://twitter.com/

users quickly consume available resources. The site becomes overloaded and too slow, and potential users go away unsatisfied.

Then again, maybe you were thinking big and you procured more resources than you actually needed. You geared up for a big party, and it failed to materialize. Your cost structure is out of control, because there are only enough users to keep your servers partially occupied. Your business may fail because your fixed costs are too high.

Of course, you might have guessed correctly and your user base is growing at the rate you expected. Even then you're still in a vulnerable position. Early one morning you wake up to find that a link to your web site is now on the front page of Digg,[3] Reddit,[4] or Slashdot.[5] Or, a CNN commentator has mentioned your site in an offhand way and your URL is scrolling across the headline crawl at the bottom of the screen. This was the moment you've been waiting for, your chance at fame and fortune! Unfortunately, your fixed-scale infrastructure fails to be up to the task, so all those potential new users go away unhappy. The day, once so promising, ends up as yet another success disaster.

As you can see, making predictions about web traffic is a very difficult endeavor. The odds of guessing wrong are very high, as are the costs.

Cloud computing gives you the tools needed to prepare and cope with a traffic onslaught, such as the ones I have just described. Providing you've put the time in up-front to architect your system properly and test it for scalability, a solution based on cloud computing will give you the confidence to withstand a traffic surge without melting your servers or sending you into bankruptcy.

Tell Me about Cloud Computing!

Let's dig a bit deeper into the concept of cloud computing now. I should warn you up-front that we'll be talking about *business* in this ostensibly technical book. There's simply no way to avoid the fact that cloud computing is more than just a new technology; it's a new business model as well. The technology is certainly interesting and I'll have plenty to say about it, but a complete discussion of cloud computing

[3] http://digg.com/
[4] http://reddit.com/
[5] http://slashdot.org/

will include business models, amortization, and even (gasp) dollars and cents. When I was young I was a hard-core geek and found these kinds of discussions irrelevant, perhaps even insulting. I was there for the technology, not to talk about money! With the benefit of 30 years of hindsight, I can now see that a real entrepreneur is able to use a mix of business and technical skills to create a successful business.

What's a Cloud?

Most of us have seen architecture diagrams like the one in Figure 1.1.

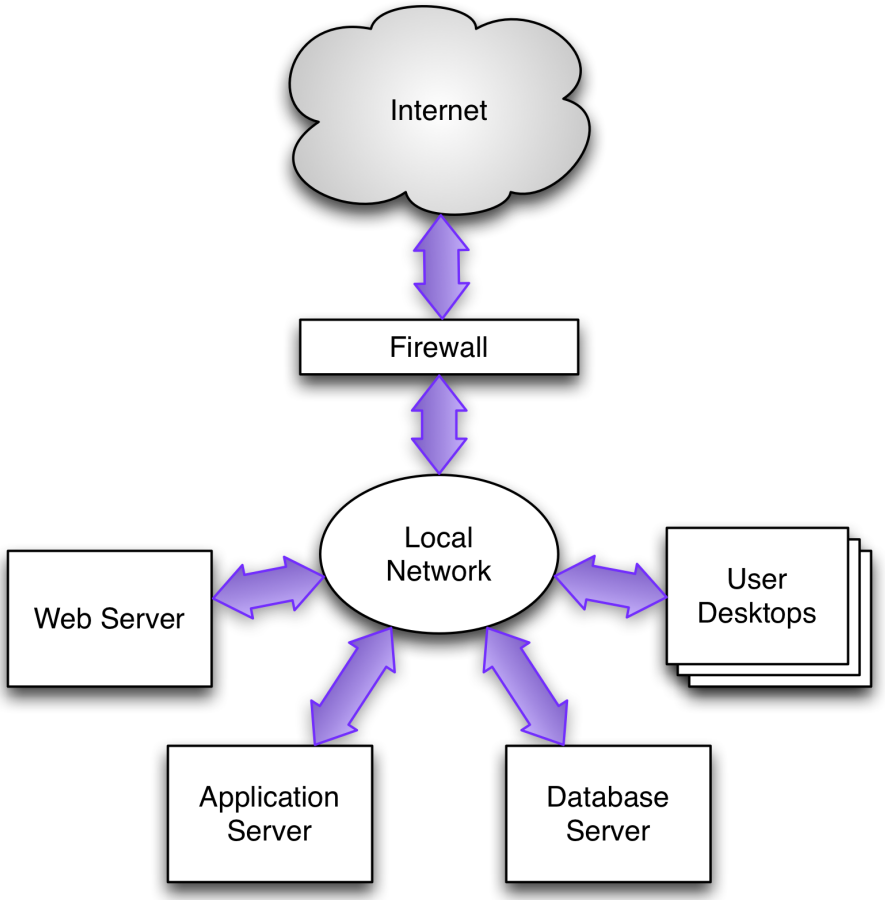

Figure 1.1. The Internet was once represented by a cloud

The cloud was used to indicate the Internet. Over time the meaning of "the Internet" has shifted, where it now includes the resources usually perceived as being *on* the Internet as well as the means to access them.

The term **cloud computing** came into popular use just a few years before this book was written. Some were quick to claim that, rather than a new concept, the term was simply another name for an existing practice. On the other hand, the term has become sufficiently powerful for some existing web applications have to magically turned into examples of cloud computing in action! Such is the power of marketing.

While the specifics may vary from vendor to vendor, you can think of **the cloud** as a coherent, large-scale, publicly accessible collection of compute, storage, and networking resources. These are allocated via web service calls (a programmable interface accessed via HTTP requests), and are available for short- or long-term use in exchange for payment based on actual resources consumed.

The cloud is intrinsically a multi-user environment, operating on behalf of a large number of users simultaneously. As such, it's responsible for managing and verifying user identity, tracking allocation of resources to users, providing exclusive access to the resources owned by each user, and preventing one user from interfering with other users. The software that runs each vendor's cloud is akin to an operating system in this regard.

Cloud computing builds on a number of important foundation-level technologies, including TCP-IP networking, robust internet connectivity, SOAP- and REST-style web services, commodity hardware, virtualization, and online payment systems. The details of many of these technologies are hidden from view; the cloud provides developers with an idealized, abstracted view of the available resources.

The Programmable Data Center

Let's think about the traditional model for allocation of IT resources. In the paragraphs that follow, the resources could be servers, storage, IP addresses, bandwidth, or even firewall entries.

If you're part of a big company and need additional IT resources, you probably find you're required to navigate through a process that includes a substantial amount of person-to-person communication and negotiation. Perhaps you send emails, create an online order or ticket, or simply pick up the phone and discuss your resource

requirements. At the other end of the system there's some manual work involved to approve the request; locate, allocate, and configure the hardware; deal with cables, routers, and firewalls; and so forth. It is not unheard of for this process to take 12–18 months in some organizations!

If you are an entrepreneur, you call your ISP (Internet Service Provider), have a discussion, negotiate and then commit to an increased monthly fee, and gain access to your hardware in a time frame measured in hours or sometimes days.

Once you've gone through this process, you've probably made a long-term commitment to operate and pay for the resources. Big companies will charge your internal cost center each month, and will want to keep the hardware around until the end of its useful life. ISPs will be more flexible, but it is the rare ISP that is prepared to make large-scale changes on your behalf every hour or two.

The cloud takes the human response out of the loop. You (or more likely a management application running on your behalf) make web service requests ("calls") to the cloud. The cloud then goes through the following steps to service your request:

1. accepts the request
2. confirms that you have permission to make the request
3. validates the request against account limits
4. locates suitable free resources
5. attaches the resources to your account
6. initializes the resources
7. returns identifiers for the resources to satisfy the request

Your application then has exclusive access to the resources for as much time as needed. When the application no longer needs the resources, the application is responsible for returning them to the cloud. Here they are prepared for reuse (reformatted, erased, or rebooted, as appropriate) and then marked as free.

Since developers are accustomed to thinking in object oriented terms, we could even think of a particular vendor's cloud as an object. Indeed, an idealized definition for a cloud might look like this in PHP:[6]

[6] This doesn't map to any actual cloud; the method and parameter names are there only to illustrate my point.

```
class Cloud
{

  public function getDataCenters()
  {
    ⋮
  }

  public function allocateServer($dataCenter, $count)
  {
    ⋮
  }

  public function releaseServer($server)
  {
    ⋮
  }

  public function allocateDiskStorage($dataCenter, $gb)
  {
    ⋮
  }

  public function releaseDiskStorage($storage)
  {
    ⋮
  }

  ⋮
}
```

Here's how this idealized cloud would be used. First, we retrieve a list of available data centers (`$d`), and store a reference to the first one in the list (`$d1`):

```
$c = new Cloud();
$d = $c->getDataCenters();
$d1 = $d[0];
```

We can then allocate a server (`$server`) to the data center and create some storage space (`$storage`):

```
$server = $c->allocateServer($d1, 1);
$storage = $c->allocateDiskStorage($d1, 100);
```

The important point is that you can now write a program to initiate, control, monitor, and choreograph large-scale resource usage in the cloud. Scaling and partitioning decisions (such as how to add more server capacity or allocate existing capacity) that were once made manually and infrequently by system administrators with great deliberation can now be automated and done with regularity.

Characterizing the Cloud

Now that you have a basic understanding of what a cloud is and how it works, let's enumerate and dive in to some of its most useful attributes and characteristics. After spending years talking about Amazon Web Services in public forums, I've found that characterization is often more effective than definition when it comes to conveying the essence of the Amazon Web Services, and what it can do.

General Characteristics

Here are some general characteristics of the Amazon Web Services.

Elastic

The cloud allows scaling up and scaling down of resource usage on an as-needed basis. Elapsed time to increase or decrease usage is measured in seconds or minutes, rather than weeks or months.

Economies of scale

The cloud provider is able to exploit economies of scale and can procure real estate, power, cooling, bandwidth, and hardware at the best possible prices. Because the provider is supplying infrastructure as a commodity, it's in its best interest to drive costs down over time. The provider is also able to employ dedicated staffers with the sometimes elusive skills needed to operate at world-scale.

Pay-as-you-go

This is a general characteristic rather than a business characteristic for one very good reason: with cloud-based services, technical people will now be making resource allocation decisions that have an immediate effect on resource consumption and the level of overall costs. Running the business efficiently becomes *everyone's* job.

Business Characteristics

Here are some of the defining characteristics of the Amazon Web Services from a business-oriented point of view:

No up-front investment

Because cloud computing is built to satisfy usage on-demand for resources, there's no need to make a large one-time investment before actual demand occurs.

Fixed costs become variable

Instead of making a commitment to use a particular number of resources for the length of a contract (often one or three years), cloud computing allows for resource consumption to change in real time.

CAPEX becomes OPEX

Capital expenditures are made on a long-term basis and reflect a multi-year commitment to using a particular amount of resources. Operation expenditures are made based on actual use of the cloud-powered system and will change in real time.

Allocation is fine-grained

Cloud computing enables minimal usage amounts for both time and resources (for example: hours of server usage, bytes of storage).

The business gains flexibility

Because there's no long-term commitment to resources, the business is able to respond rapidly to changes in volume or the type of business.

Business focus of provider

The cloud provider is in the business of providing the cloud for public use. As such, it has a strong incentive to supply services that are reliable, applicable, and cost-effective. The cloud reflects a provider's core competencies.

Costs are associative

Due to the flexible resource allocation model of the cloud, it's just as easy to acquire and operate 100 servers for one hour as it is to acquire and operate one server for 100 hours. This opens the door to innovative thinking with respect to ways of partitioning large-scale problems.

Technical Characteristics

Here are some of the defining characteristics of the Amazon Web Services from the technical standpoint:

Scaling is quick

New hardware can be brought online in minutes to deal with unanticipated changes in demand, either internally (large compute jobs) or externally (traffic to a web site). Alternatively, resources can be returned to the cloud when no longer needed.

Infinite scalability is an illusion

While not literally true, each consumer can treat the cloud as if it offers near-infinite scalability. There's no need to provision ahead of time; dealing with surges and growth in demand is a problem for the cloud provider, instead of the consumer.

Resources are abstract and undifferentiated

Cloud computing encourages a focus on the relevant details—results and the observable performance—as opposed to the technical specifications of the hardware used. Underlying hardware will change and improve over time, but it's the job of the provider to stay on top of these issues. There's no longer a need to become personally acquainted with the intimate details of a particular dynamic resource.

Clouds are building blocks

The cloud provides IT resources as individual, separately priced, atomic-level building blocks. The consumer can choose to use none, all, or some of the services offered by the cloud.

Experimentation is cheap

The cloud removes the economic barrier to experimentation. You can access temporary resources to try out a new idea without making long-term commitments to hardware.

Some Common Misconceptions

After talking to thousands of people over the last few years, I've learned that there are a lot of misconceptions floating around the cloud. Some of this is due to the

inherent unease that many feel with anything new. Other misconceptions reflect the fact that all the technologies are evolving rapidly, with new services and features appearing all the time. What's true one month is overtaken the next by a new and improved offering. With that said, here are some of the most common misconceptions. Parts of this list were adapted from work done at the University of California, Berkeley.[7]

"The cloud is a fad"

Given the number of once-promising technologies that have ended up on history's scrap heap, there's reason to be skeptical. It's important to be able to respond quickly and cost-effectively to changes in one's operating environment; this is a trend that's unlikely to reverse itself anytime soon, and the cloud is a perfect fit for this new world.

"Applications must be re-architected for the cloud"

I hear this one a lot. While it's true that some legacy applications will need to be re-architected to take advantage of the benefits of the cloud, there are also many existing applications using commercial or open source stacks that can be moved to the cloud more or less unchanged. They won't automatically take advantage of all the characteristics enumerated above, but the benefits can still be substantial.

"The cloud is inherently insecure"

Putting valuable corporate data "somewhere else" can be a scary proposition for an IT manager accustomed to full control. Cloud providers are aware of this potential sticking point, taking this aspect of the cloud very seriously. They're generally more than happy to share details of their security practices and policies with you. Advanced security systems, full control of network addressing and support for encryption, coupled with certifications such as SAS 70,[8] can all instill additional confidence in skeptical managers. I'll address the ways that AWS has helped developers, CIOs, and CTOs to get comfortable with the cloud in the next chapter.

---

[7] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia, *Above the Clouds: A Berkeley View of Cloud Computing* (Berkeley: University of California, 2009), at http://d1smfj0g31qzek.cloudfront.net/abovetheclouds.pdf.

[8] http://www.sas70.com/

"The cloud is a single point of failure"

Some developers wonder what happens if the cloud goes down? Unlike traditional data centers, the AWS cloud offers a wide variety of options for functional and geographic redundancy to ensure high availability.

"The cloud promotes lock-in"

Because you can run existing applications on the cloud, they can be moved off as easily as they can be moved on. Operating systems, middleware, and applications can often be run in a cloud environment with little or no change. Of course, applications can be updated to take advantage of services offered by the cloud and that's what we'll be exploring in this book.

"The cloud is only good for running open source code"

This argument no longer holds water. Commercial operating system and application software vendors now recognize the cloud as a legitimate software environment and have worked to ensure that their applications have the proper cloud-friendly licenses. Forward-thinking vendors are now making their licensed software available on an hourly, pay-as-you-go basis. Instead of buying, for example, a database license for tens or even hundreds of thousands of dollars, you can gain access to the same database for a few dollars per hour.

"Cloud resources are too expensive"

Making a genuine comparison between internal IT resources and equivalent cloud computing resources has proven to be a difficult task.[9] Establishing the complete, all-inclusive cost of internal resources requires a level of tracking and accounting that's absent in most large- or mid-sized organizations. It's far too easy to neglect obvious costs, or to compare internal resources at a permanent hourly cost to scalable cloud resources that cost nothing when idle.

You'll find more detailed explanations in the remaining chapters of this book as to why these are indeed misconceptions.

[9] See, for example, James Hamilton's blog post: *McKinsey Speculates that Cloud Computing May Be More Expensive than Internal IT* at http://perspectives.mvdirona.com/2009/04/21/McKinseySpeculatesThatCloudComputingMayBeMoreExpensiveThanInternalIT.aspx .

Cloud Usage Patterns

Let's now examine some common cloud usage patterns. Armed with this information, you should be in a good position to decide whether your application or workload is a good fit for AWS. Although all these patterns essentially represent usage over time, there are a number of important nuances. In the cases below, "usage" generally represents a combination of common cloud resources—servers, storage, and bandwidth.

Constant usage over time
: common for internal applications where there's little variation in usage or load from day to day or hour to hour

Cyclic internal load
: characteristic for batch or data processing applications run on a predictable cycle, such as close of business for the day or month; the load, both in time and expected resource consumption, is highly predictable.

Cyclic external load
: often applies to web sites that serve a particular market demand; sites related to entertainment and sporting events often fit this pattern.

Spiked internal load
: typical in environments where researchers or analysts can submit large-scale, one-time jobs for processing; the demand is usually unpredictable.

Spiked external load
: seen on the Web when an unknown site suddenly becomes popular, often for a very short time

Steady growth over time
: usually for a mature application or web site; as additional users are added, growth and resources track accordingly.

Cloud Use Cases

Given that you've read this far, you might be wondering how other people are putting clouds to use. In this section I've collected some (but definitely not all) of the most common use cases, starting simple and building to the more complex.

Hosting Static Web Sites and Complex Web Applications

The cloud can easily host a static web site built from static HTML pages, CSS style sheets, and images. In fact, the simplest of such sites can be hosted using only cloud storage, perhaps aided by a content distribution system.

More complex web sites, often with substantial server-side processing and access to a relational database, can also be hosted in the cloud. These sites make use of cloud storage and processing, and often require substantial processing and storage resources to attain the required scale.

Software Development Life Cycle Support

The cloud is a good match for the resource requirements of each phase of the software development life cycle.

During development, using the cloud can ensure that developers have adequate resources for their work. Suppose that a team of developers are building a classic three-tier web application with web, application, and database tiers, each destined to reside on a separate physical server at deployment time. Without AWS, each developer would be supplied with three complete servers, each of which would sit idle for much of the day. Costs grow quickly when new developers are added to the project. Moving to the cloud means that each developer can spin up servers in the morning, develop and test all day, and then return the servers to the cloud at the end of the working day.

The cloud is also valuable during software testing. Developers can spin up testing servers and run unit tests on them without burdening their development servers. If there are numerous unit tests, multiple parallel servers can be used to spread the load around.

The cloud can be used to support a continuous integration environment. In such an environment, each source code commit operation initiates a multistep process of rebuilding, unit testing, and functional testing. If the code is being written for multiple target environments (several different versions or variants of Linux) or platforms (Windows and Linux), the cloud can be a very cost-effective alternative to owning your own infrastructure.

Load and performance testing can be done throughout each development cycle using cloud computing resources. If the application itself will run on the cloud, the testing will ensure that it performs well under a heavy load, adding additional resources as the load grows and removing them as it dissipates.

Testing the performance of a web application intended for public or enterprise deployment becomes easier when the cloud can supply the resources needed to conduct a test at a scale representative of the expected load. Several companies use cloud resources to generate loads that are the equivalent to hundreds of thousands of simultaneous users.

Once the application has been deployed (perhaps also to the cloud), the cloud can supply the resources needed to perform compatibility tests when application middleware layers or common components are updated. Thorough testing can help establish the confidence needed to make substantial upgrades to a production system without the risk of downtime.

Training

The cloud can supply the short-term resources needed to support various types of training programs.

If students are learning how to install, run, and monitor the three-tier application described in the previous section, they can use their own laptops to access cloud resources allocated for the duration of the class. When the class is over for the day the resources are returned to the cloud. The students can start from a single "master" machine image and avoid wasting time (theirs or the instructor's) installing and configuring required packages and applications.

Traditional training classes must impose limits on class size corresponding to the restricted amount of physical hardware that they have available. Leading companies are now conducting online training seminars, backed by per-student cloud-based resources where an additional server is launched as each new student joins the class. This technique has been used by application and database software vendors with impressive results.

Demos

Resources drawn from the cloud can be used to host and deliver demos and trial versions of packaged software applications. Vendors can place demos into the hands of potential customers while the lead is "hot," rather than after the usual protracted delay while a suitable test environment is prepared. Application vendors can create and provide access to a server hosted in the cloud at low cost and on short notice. The sales cycle is shortened and customers have a good experience with the use of cloud-based resources. In some cases, cloud-based demos actually lead to cloud-based deployment.

Data Storage

The cloud is a good place to store private or public data. Scalability, long-term durability, and economy of scale are of paramount importance for this use case. The stored data could be as simple and compact as a few personal files for backup, or it could be as large and complex as a backup of a company's entire digital assets, or anything in between.

Often, use of storage in the cloud turns out to be an excellent first step, a step that inspires confidence and soon leads to considering the cloud for other, more complex, use cases.

Disaster Recovery and Business Continuity

Enterprises with a mission-critical dependence on IT resources must have a plan in place to deal with any setback, be it a temporary or permanent loss of the resources or access to them. The plan must take into account the potential for fires, floods, earthquakes, and terrorist acts to disrupt a company's operations. Many businesses maintain an entire data center in reserve; data is replicated to the backup center on occasion and the entire complex stands ready to be activated at a moment's notice. Needless to say, the cost of building and running a duplicate facility is considerable.

Cloud computing, once again, offers a different way to ensure business continuity. Instead of wasting capital on hardware that will never be put to use under normal circumstances, the entire corporate network can be modeled as a set of cloud resources, captured in template form, and then instantiated when trouble strikes. In this particular use case, you'll need to work with your cloud provider to ensure that the necessary resources will be available when you need them.

Once the corporate network has been modeled for business continuity purposes, other interesting uses come to mind. Traditionally, widespread deployment of updated versions of middleware and shared applications components require substantial compatibility and performance testing. This task is fraught with peril! Many companies find themselves slowly slipping behind: they're unable to deploy the newest code due to limitations in their ability to fully test before deployment, and unwilling to risk facing the consequences of a failed deployment.

Imagine spinning up a full copy (or a representative, scaled-down subset) of the corporate network, along with specified versions of the application components to be tested, and then running compatibility and load tests on it, all in the cloud, and at a very reasonable cost.

Media Processing and Rendering

A number of popular web sites support uploading of media files: music, still images, or videos. Once uploaded the files undergo a number of processing steps, which can be compute-intensive, I/O intensive, or both. Files of all types are scanned for viruses and other forms of malware. Music is fingerprinted (to check for copyright violations) and then transcoded to allow for playback at different bit rates. Images are scaled, watermarked, checked for duplication, and rendered in different formats. Videos are also transcoded and scaled, and sometimes broken into shorter chunks. Finally, the finished objects are stored and made available for online viewing or downloading.

Rendering uses a scene description to generate frames for an animated movie. Each frame can be rendered independently of the others. There's comparatively little input data, but plenty of output data. The process is compute-intensive, since each pixel of each frame must be computed, taking into account light, shadow, color, and motion.

Cloud computing is ideal for processing and rendering use cases due to the amount of storage, processing, and internet bandwidth they can consume.

Business and Scientific Data Processing

Scientific and business data processing often involves extremely large-scale data sets and can consume vast amounts of CPU power. Analysis is often done on an on-demand basis, leading to over-commitments of limited internal resources. In fact,

I'm told that many internal scientific compute grids routinely flip between 0% usage (absolutely no work to be done) and 100% usage (every possible processor is in use). This is a particularly acute problem on university campuses, where usage heats up before the end of the semester and before major conferences.

Business data processing can be ad hoc (unscheduled) or more routine; monthly payroll processing and daily web log processing come to mind as very obvious use cases for cloud computing. A large, busy web site is capable of generating tens of gigabytes of log file data in each 24-hour period. Due to the amount of business intelligence that can be mined from the log files, analysis is a mission-critical function. Gaining access to the usage data on a more timely basis enables better site optimization and a quicker response to changes and trends. The daily analysis process starts to take long and longer, and at some point begins to take almost 24 hours. Once this happens, heavily parallel solutions are brought to bear on the problem, consuming more resources for a shorter amount of time—a perfect case for cloud computing.

Overflow Processing

As companies begin to understand the benefits that cloud computing brings, they look for solutions that allow them to use their existing IT resources for routine work, while pushing the extra work to the cloud. It's like bringing in temporary workers to handle a holiday rush.

Overflow processing allows companies to become comfortable with the cloud. They find more and more ways to use the cloud as their confidence level increases, and as the amount of vital corporate data already present in the cloud grows.

Just Recapping

As you can see, there are a number of different ways to use the cloud to host existing applications, build creative new ones, and improve the cost-effectiveness and efficiency of organizations large and small.

In this chapter we've learned the fundamentals of cloud computing. Using a sporting-venue analogy, we've seen how cloud computing allows individuals and organizations to do a better job of matching available resources to actual demand. We've learned about the notion of a "success disaster" and aim to avoid having one of our own—with the assistance of AWS, of course. From there we covered the character-

istics of a cloud, and proposed that the cloud could be thought of as a programmable data center. We examined the cloud from three sides: general, technical, and business, and enumerated some common misconceptions. Finally, we took a quick look at usage patterns and an extended look at actual use cases.

In the next chapter we'll learn more about the Amazon Web Services, and we'll get ready to start writing some code of our own.

Chapter 2

Amazon Web Services Overview

In the previous chapter we discussed the concept of cloud computing in general terms. We listed and discussed the most interesting and relevant characteristics of the cloud. With that information as background, it's now time to move from concept to reality.

In this chapter I'll introduce Amazon Web Services, or **AWS** for short. After a review of some key concepts I'll talk about each AWS service.

Amazon and AWS Overview

You've probably made a purchase at the Amazon.com[1] site. Perhaps you even bought this book from Amazon.com. One of my first purchases, way back in November 1996, was a book on Perl programming.

Amazon.com Inc. was founded in 1994 and launched in 1995. In order to attain the scale needed to create a profitable online business, the company made strategic investments in world-scale internet infrastructure, including data centers in multiple locations around the world, high-speed connectivity, a plethora of servers, and the

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

creation of a world-class system architecture. With an active customer base in the tens of millions, each and every system component must be reliable, efficient, cost-effective, and highly scalable.

Realizing that developers everywhere could benefit from access to the services that support Amazon's web site, Amazon decided to create a new line of business. In early 2006, the company launched the Amazon Simple Storage Service (S3). Since then Amazon has brought a broad line of infrastructure, payment, workforce, merchant, and web analytic services to market under Amazon Web Services (AWS). In this book I'll focus on the infrastructure services. If you'd like to learn about the other services, please visit the AWS home page.[2]

Building Blocks

AWS consists of a set of building-block services. The services are designed to work independently, so that you can use one without having to sign up for or know anything at all about the others. They are, however, also designed to work well together. For example, they share a common naming convention and authentication system. So, much of what you learn as you use one service is applicable to some or all the other services! This building-block approach also minimizes internal connections and dependencies between the services, which gives Amazon the ability to improve each service independently so that each works as efficiently as possible.

Every function in AWS can be accessed by making a web service call. Starting a server, creating a load balancer, allocating an IP address, or attaching a persistent storage volume (to name just a few actions) are all accomplished by making web service calls to AWS. These calls are the down-to-the-metal, low-level interface to AWS. While it's possible (and simple enough) to make the calls yourself, it's far easier to use a client library written specifically for the programming language of your choice.

Protocols

The web service calls use one of two popular protocols **SOAP** (once short for Simple Object Access Protocol, but now just a pseudo-acronym) and **REST** (an assimilation of Representational State Transfer). Because this book will focus on building useful applications and utilities, I will access AWS using a client library rather than

[2] http://aws.amazon.com/

spending much time at the web service protocol layer. Suffice it to say that SOAP and REST are two different ways to initiate a call (or request) to a web service. Libraries and tools are layered on top of the AWS **API**s (Application Programming Interfaces) to simplify the process of accessing the services.

I guess I have to mention XML here too! XML is a fundamental part of the SOAP protocol. If you access AWS using a SOAP-based library you'll have no dealings with XML tags or elements. However, if you use a REST-based library, you'll have to do some parsing to access the data returned by each call. The examples in this book will use PHP's SimpleXML parser.[3]

Figure 2.1 shows how all the parts that I've outlined in this section fit together. We'll be focusing on building AWS-powered Applications (top-left corner):

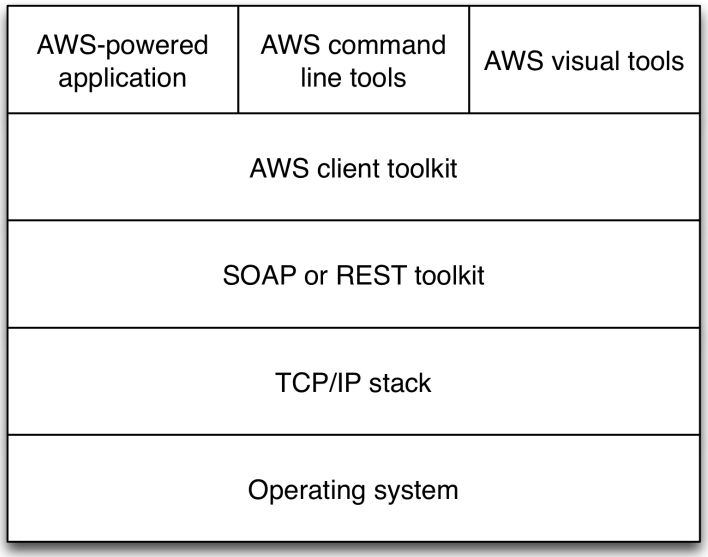

Figure 2.1. Putting the pieces together

The command line tools and visual tools communicate with AWS using the open, published APIs. So, you're able to duplicate what you see any tool do in your own applications. As a consequence of this strict layering of behavior, all developers are on an equal footing.

[3] http://www.php.net/simplexml/

In the section called "Key Concepts" below, I'll discuss the basic functions (for example, `RunInstances`) and the associated command line tools (`ec2-run-instances`). Keep in mind that the same functionality can be accessed using visual tools supplied by Amazon or by third parties, and that you can always build your own tools using the same APIs.

Dollars and Cents

Because AWS is a pay-as-you-go web service, there's a separate cost for the use of each service. You can model your AWS costs during development time to gain a better understanding of what it will cost to operate your site at scale. With sufficient attention to detail you should be able to compute the actual cost of serving a single web page, or performing some other action initiated by one of your users. You can also use the AWS Simple Monthly Calculator[4] to estimate your costs.

With that in mind, let's talk about pricing, metering, accounting, presentment, and billing before we look at the services themselves.

Pricing involves deciding what to charge for, how often to charge, and how much to charge. AWS charges for resource usage at a very granular level. Here are some of the pricing dimensions that AWS uses:

- **Time**—an hour of CPU time.
- **Volume**—a gigabyte of transferred data.
- **Count**—number of messages queued.
- **Time and space**—a gigabyte-month of data storage.

Most services have more than one pricing dimension. If you use an Amazon EC2 (Elastic Compute Cloud) server to do a web crawl, for example, you'll be charged for the amount of time that the server is running and for the data that you fetch.

The web site detail page for each AWS service shows the cost of using the service. Each AWS service is published and visible to everyone. The pricing for many of the services reflects volume discounts based on usage; that is, the more you use the service, the less it costs per event. Pricing for the services tends to decline over time

[4] http://calculator.s3.amazonaws.com/calc5.html

due to the effects of Moore's Law and economies of scale.[5] Pricing also reflects the fact that operating costs can vary from country to country.

Metering refers to AWS measuring and recording information about your use of each service. This includes information about when you called the service, which service you called, and how many resources you consumed in each of the service's pricing dimensions.

Accounting means that AWS tabulates the metered information over time, adding up your usage and tracking your overall resource consumption. You can use the AWS Portal to access detailed information about your resource consumption.

Presentment involves making your AWS usage available so that you can see what you've used and the cost you've incurred. This information is also available from the AWS portal.

Billing indicates that AWS will charge your credit card at the beginning of each month for the resources you consumed in the previous month.

Does any of this seem a little familiar? Indeed, your utility supplier (phone, water, or natural gas) takes on a very similar set of duties. This similarity causes many people to correctly observe that an important aspect of cloud computing is utility pricing.

Key Concepts

Let's review some key concepts and AWS terms to prepare to talk about the services themselves. In the following sections, I include lists of some of the functions and commands that you can use to access the relevant parts of AWS mentioned below. These lists are by no means complete; my intention is to give you a better sense of the level of abstraction made possible by AWS, and also to hint at the types of functions that are available within the AWS API.

[5] Moore's Law refers to the long-term trend where the number of transistors placed on an integrated circuit doubles every two years. It has since been generalized to reflect technology doubling in power and halving in price every two years.

Availability Zone

An AWS **Availability Zone** represents a set of distinct locations within an AWS Region. Each Availability Zone has independent power grid and network connections so that it's protected from failures in other Availability Zones. The zones within a Region are connected to each other with inexpensive, low-latency connections. The Region name is part of the zone name. For example, us-east-1a is one of four zones in the us-east-1 Region.

The mapping of a zone name to a particular physical location is different yet consistent for each AWS account. For example, my us-east-1a is possibly different to your us-east-1a, but my us-east-1a is always in the same physical location. This per-user mapping is intentional and was designed to simplify expansion and load management.

The `DescribeAvailabilityZones` function and the `ec2-describe-availability-zones` command return the list of Availability Zones for a Region.

Region

An AWS **Region** represents a set of AWS Availability Zones that are located in one geographic area. Each AWS Region has a name that roughly indicates the area it covers, but the exact location is kept secret for security purposes. The current Regions are `us-east-1` (Northern Virginia), `us-west-1` (Northern California), `eu-west-1` (Ireland), and `ap-southeast-1` (Signapore). Over time, additional Regions will become available. The `DescribeRegions` function and the `ec2-describe-regions` command return the current list of Regions. You may choose to make use of multiple Regions for business, legal, or performance reasons.

Access Identifiers

AWS uses a number of different **access identifiers** to identify accounts. The identifiers use different forms of public key encryption and always exist in pairs. The first element of the pair is public, can be disclosed as needed, and serves to identify a single AWS account. The second element is private, should never be shared, and is used to create a signature for each request made to AWS. The signature, when transmitted as part of a request, ensures the integrity of the request and also allows AWS to verify that the request was made by the proper user. AWS can use two different sets of access identifiers. The first comprises an Access Key ID and a Secret

Access Key. The second is an X.509 certificate with public and private keys inside. You can view your access identifiers from the AWS portal.[6]

Amazon Machine Image

An **Amazon Machine Image (AMI)** is very similar to the root drive of your computer. It contains the operating system and can also include additional software and layers of your application such as database servers, middleware, web servers, and so forth. You start by booting up a prebuilt AMI, and before too long you learn how to create custom AMIs for yourself or to share, or even sell. Each AMI has a unique ID; for example, the AMI identified by `ami-bf5eb9d6` contains the Ubuntu 9.04 Jaunty server. The `DescribeImages` function and the `ec2-describe-images` command return the list of registered instances. The AWS AMI catalog[7] contains a complete list of public, registered AMIs.

Instance

An **instance** represents one running copy of an AMI. You can launch any number of copies of the same AMI. Instances are launched using `RunInstances` and the `ec2-run-instances` command. Running instances are listed using `DescribeInstances` and `ec2-describe-instances`, and terminated using the `TerminateInstances` function or the `ec2-terminate-instances` command. Before long you will also learn about the AWS Management Console, which is a visual tool for managing EC2 instances.

Elastic IP Address

AWS allows you to allocate fixed (static) IP addresses and then attach (or route) them to your instances; these are called **Elastic IP Addresses**. Each instance can have at most one such address attached. The "Elastic" part of the name indicates that you can easily allocate, attach, detach, and free the addresses as your needs change. Addresses are allocated using the `AllocateAddress` function or the `ec2-allocate-address` command, and attached to an instance using the `AssociateAddress` function or the `ec2-associate-address` command.

[6] http://aws.amazon.com/account
[7] http://aws.amazon.com/amis

Elastic Block Store Volume

An **Elastic Block Store** (EBS) volume is an addressable disk volume. You (or your application, working on your behalf) can create a volume and attach it to any running instance in the same Availability Zone. The volume can then be formatted, mounted, and used as if it were a local disk drive. Volumes have a lifetime independent of any particular instance; you can have disk storage that persists even when none of your instances are running. Volumes are created using the `CreateVolume` function or the `ec2-create-volume` command, and then attached to a running instance using the `AttachVolume` function or the `ec2-attach-volume` command.

Security Group

A **Security Group** defines the allowable set of inbound network connections for an instance. Each group is named and consists of a list of protocols, ports, and IP address ranges. A group can be applied to multiple instances, and a single instance can be regulated by multiple groups. Groups are created using the `CreateSecurityGroup` function and the `ec2-add-group` command. The `AuthorizeSecurityGroupIngress` function and the `ec2-authorize` command add new permissions to an existing security group.

Access Control List

An **Access Control List** (ACL) specifies permissions for an object. An ACL is a list of identity/permission pairs. The `GetObjectAccessControlPolicy` function retrieves an object's existing ACL and the `SetObjectAccessControlPolicy` function sets a new ACL on an object.

AWS Infrastructure Web Services

Now that you know the key concepts, let's look at each of the AWS infrastructure web services.

Amazon Simple Storage Service

The Amazon **Simple Storage Service (S3)** is used to store binary data objects for private or public use. The S3 implementation is fault-tolerant and assumes that hardware failures are a common occurrence.

There are multiple independent S3 locations: the United States Standard Region, Northern California Region,[8] Europe, and Asia.

S3 automatically makes multiple copies of each object to achieve high availability, as well as for durability. These objects can range in size from one byte to five gigabytes. All objects reside in buckets, in which you can have as many objects as you like. Your S3 account can accommodate up to 100 buckets or named object containers. Bucket names are drawn from a global namespace, so you'll have to exercise some care and have a sound strategy for generating bucket names. When you store an object you provide a key that must be unique to the bucket. The combination of the S3 domain name, the globally unique bucket name, and the object key form a globally unique identifier. S3 objects can be accessed using an HTTP request, making S3 a perfect place to store static web pages, style sheets, JavaScript files, images, and media files. For example, here's an S3 URL to a picture of Maggie, my Golden Retriever: http://sitepoint-aws-cloud-book.s3.amazonaws.com/maggie.jpg.

The bucket name is `sitepoint-aws-cloud-book` and the unique key is `maggie.jpg`. The S3 domain name is s3.amazonaws.com.

Each S3 object has its own ACL. By default, each newly created S3 object is private. You can use the S3 API to make it accessible to everyone or specified users, and you can grant them read and/or write permission. I set Maggie's picture to be publicly readable so that you can see her.

Other AWS services use S3 as a storage system for AMIs, access logs, and temporary files.

Amazon S3 charges accrue based on the amount of data stored, the amount of data transferred in and out of S3, and the number of requests made to S3.

Amazon CloudFront

Amazon **CloudFront** is a content distribution service designed to work in conjunction with Amazon S3. Because all Amazon S3 data is served from central locations in the US, Europe, and Asia, access from certain parts of the world can take several hundred milliseconds. CloudFront addresses this "speed of light" limitation with

[8] The Northern California location provides optimal performance for requests originating in California and the Southwestern United States.

a global network of edge locations (16 at press time) located near your end users in the United States, Europe, and Asia.

After you have stored your data in an S3 bucket, you can create a CloudFront Distribution. Each distribution contains a unique URL, which you use in place of the bucket name and S3 domain to achieve content distribution. Maggie's picture is available at the following location via CloudFront:
http://d1iodn8r1n0x7w.cloudfront.net/maggie.jpg.

As you can see, the object's name is preserved, prefixed with a URL taken from the bucket's distribution. The HTTP, HTTPS, and RTMP protocols can be used to access content that has been made available through CloudFront.

CloudFront charges accrue based on the amount of data transferred out of CloudFront and the number of requests made to CloudFront.

Amazon Simple Queue Service

You use the **Simple Queue Service (SQS)** to build highly scalable processing pipelines using loosely coupled parts. Queues allow for flexibility, asynchrony, and fault tolerance. Each step in the pipeline retrieves work units from an instance of the queue service, processes the work unit as appropriate, and then writes completed work into another queue for further processing. Queues work well when the requirements—be it time, CPU, or I/O speed—for each processing step for a particular work unit vary widely.

Like S3, there are separate instances of SQS running in the US and in Europe.

SQS usage is charged based on the amount of data transferred and the number of requests made to SQS.

Amazon SimpleDB

Amazon **SimpleDB** supports storage and retrieval of semi-structured data. Unlike a traditional relational database, SimpleDB does not use a fixed database schema. Instead, SimpleDB adapts to changes in the "shape" of the stored data on the fly, so there's no need to update existing records when you add a new field. SimpleDB also automatically indexes all stored data so it's unnecessary to do your own profiling or query optimization.

The SimpleDB data model is flexible and straightforward. You group similar data into domains. Each domain can hold millions of items, each with a unique key. Each item, in turn, can have a number of attribute/value pairs. The attribute names can vary from item to item as needed.

Like the other services, SimpleDB was built to handle large amounts of data and high request rates. So there's no need to worry about adding additional disk drives and implementing complex data replication schemes as your database grows. You can grow your application to world-scale while keeping your code clean and your architecture straightforward.

SimpleDB charges accrue based on the amount of data stored, the amount of data transferred, and the amount of CPU time consumed by query processing.

Amazon Relational Database Service

The Amazon **Relational Database Service** (RDS) makes it easy for you to create, manage, back up, and scale MySQL database instances. RDS calls these **DB Instances**, and that's the terminology I'll be using in this book.

RDS handles the tedious and bothersome operational details associated with running MySQL so that you can focus on your application. You don't have to worry about procuring hardware, installing and configuring an operating system or database engine, or finding storage for backups. You can scale the amount of processing power up or down, and increase the storage allocation in a matter of minutes, so you can respond to changing circumstances with ease. You can back up your DB Instance to Amazon S3 with a single call or click, and create a fresh DB Instance from any of your snapshots.

RDS also has a Multi-AZ (or Multi-Availability Zone) option that allows you to run a redundant backup copy of your DB Instance for extra availability and reliability.

Amazon RDS charges accrue based on the amount of time that each DB Instance is running, and the amount of storage allocated to the instance.

Amazon Elastic Compute Cloud

The **Elastic Compute Cloud (Amazon EC2)** infrastructure gives you the ability to launch server instances running the AMI (Amazon Machine Instance) of your choice. Instance types are available with a wide range of memory, processing power, and

local disk storage. You can launch instances in any EC2 Region and you can choose to specify an Availability Zone if needed. Once launched, the instances are attached to your account and should remain running until you shut them down.

Each instance is protected by a firewall which, by default, blocks all internal and external connectivity. When you launch instances you can associate any number of security groups with them. The security groups allow you to control access to your instances on a very granular basis.

The EC2 infrastructure provides instances with an IP address and a DNS entry when they're launched. The address and the entry are transient: when the instance shuts down or crashes they are disassociated from the instance. If you need an IP address that will survive a shutdown or that can be mapped to any one of a number of machines, you can use an Elastic IP Address. These addresses are effectively owned by your AWS account rather than by a particular EC2 instance. Once allocated, the addresses are yours until you decide to relinquish them.

The instances have an ample amount of local disk storage for temporary processing. Like the standard IP address and DNS name, this storage is transient and is erased and reused when you're finished with the instance.

Elastic Block Store (EBS) volumes can be used for long-term and more durable storage. You can create a number of EBS volumes, attach them to your instances, and then format the volumes with the file system of your choice. You can make snapshot backups to S3, and you can restore the snapshots to the same volume or use them to create new volumes.

EC2 charges accrue based on the number of hours the instance runs and the amount of data transferred in and out. There is no charge to transfer data to and from other AWS services in the same Region. The charges for EBS volumes are based on the size of the volume (regardless of how much data is actually stored) and there are also charges for I/O requests. To prevent hoarding, you are charged for Elastic IP addresses that you allocate but don't use.

The EC2 **CloudWatch** feature provides monitoring within EC2. It collects and stores information about the performance (CPU load average, disk I/O rate, and network I/O rate) of each of your EC2 instances. The data is stored for two weeks and can be retrieved for analysis or visualization.

The EC2 **Elastic Load Balancer** allows you to distribute web traffic across any number of EC2 instances. The instances can be in the same Availability Zone or they can be scattered across several zones in a Region. The elastic load balancer performs periodic health checks on the instances that it manages, and will stop sending traffic to any instances it determines to be unhealthy. The health check consists of a configurable ping to each EC2 instance.

Finally, the EC2 **Auto Scaling** feature uses the data collected by CloudWatch to help you build a system that can scale out (adding more EC2 instances) and scale in (shutting down EC2 instances) within a defined auto scaling group. Auto scaling lets you define triggers for each operation. For example, you can use Auto Scaling to scale out by 10% when the average CPU utilization across the auto scaling group exceeds 80%, and then scale in by 10% when the CPU utilization drops below 40%.

Amazon Elastic MapReduce

The Elastic **MapReduce** service gives you the ability to use a number of EC2 instances running in parallel for large-scale data processing jobs. This service uses the open source Hadoop framework,[9] an implementation of the MapReduce paradigm. Invented by Google, MapReduce isolates you from many of the issues that arise when you need to launch, monitor, load (with data), and terminate dozens or even hundreds of instances. Elastic MapReduce works just as well for pedestrian tasks, such as log file processing, as it does for esoteric scientific applications, such as gene sequencing.

Other Services

AWS gains new features and services with great regularity. To stay up to date with the latest and greatest happenings, you should check the AWS home page and the AWS Blog[10] (written by yours truly) from time to time.

[9] http://hadoop.apache.org/mapreduce/
[10] http://aws.typepad.com/

What We've Covered

In this chapter, we took a closer look at each of the AWS infrastructure services, reviewing their usage characteristics and pricing models. We also examined a number of key AWS concepts. In the next chapter, we'll tool up in preparation for building our first scripts that make use of all these capabilities.

Chapter 3

Tooling Up

Now that you're familiar with the range of Amazon Web Services and what they're capable of, you may be wondering how we start using them. Before we start coding, however, we'll need to tool up. I'll spend some time on the visual tools (user interfaces and consoles) and command line tools. I'll also discuss the level below the tools: the libraries available for a particular programming language. This book will focus on PHP, but I'll provide some general rules for library selection. This chapter will also lead you through creating of your AWS account, installing CloudFusion, and the necessary command line tools.

Technical Prerequisites

Before we go too much further, I want to ensure that my expectations regarding your programming and system management skills are correct. It's also important that you have the right hardware and software at your disposal.

Skills Expectations

Because this book is targeted at mid-level PHP programmers, I assume that you can already read and write PHP with some skill. I'll avoid using any esoteric features

of PHP, but I'll also avoid explaining the `error_reporting` function or the detailed behavior of `foreach` loops. I also expect you to know the mechanics of writing, debugging, and running PHP code.

On the web technology side, I expect you to know HTML and some CSS, and a tiny bit of JavaScript.

On the systems side, I assume that you're comfortable with the Linux/Unix command line, as there's no time to explain `ls`, `scp`, or `grep`. I assume that you've mastered the ins and outs of your text editor, be it vi, emacs, Notepad, or otherwise.

These days, developers in small organizations are expected to know a thing or two about system administration. I assume that you, too, are quite the "devministrator" and that common system administration tasks, such as installing packages and examining log files, are part of your skill repertoire.

Hardware and Software Expectations

In the sections that follow, I expect you to have a Mac or a PC that is connected to the Internet. Your PC can be running Windows, or it can be running the Linux distribution of your choice.

It's often useful to run some PHP code locally. Packages like WAMP for Windows[1] or MAMP for the Mac[2] make it easy to install and configure PHP, MySQL, and an Apache web server for local development. I'll be talking about this again in the section called "Running the PHP Code in This Book".

You need a web browser. If all you have is Internet Explorer, that's fine—but some of the visual tools take the form of Firefox extensions.

You'll need a good **SSH (Secure Shell)** client. For Windows use, my personal favorite is PuTTY,[3] which is fast and reliable, and its terminal emulation is flawless. For Mac OS X and Linux, you can use the command line SSH clients that come with your operating system.

[1] http://www.wampserver.com/en/
[2] http://www.mamp.info/en/index.html
[3] http://www.chiark.greenend.org.uk/~sgtatham/putty/

If you work in a corporate setting and your desktop has been locked down, the cloud actually gives you a really interesting new option. You can launch an EC2 instance running Windows, access it using the Windows Remote Desktop, and use that for development while leaving your desktop machine untouched.

Optional but Recommended

My final three recommendations are that you have some spare domain names, a good DNS service, and a source code control system.

Back when I was young, domain names were expensive ($75 per year) and complex to register. These days they're cheap to acquire and registration is very simple. I recommend that you have at least one or two freely available as you work through the examples in this book.

You should also have access to a DNS provider, so that you can map your domain names to IP addresses. When I was merely middle-aged I used to run my own DNS server. Since 2004 I've been a very happy customer of ZoneEdit.[4] You can have DNS service for up to five domains at no charge.

You may want to think about a source code control system for your application as well. A source code control system hosted in the cloud will let you keep track of all the bits and pieces of your application, and will allow you to retrieve the newest code at any time. I have been using CVS Dude[5] as my CVS host since 2004. Since CVS is even more archaic than I am,[6] they also host the more modern Subversion system. Depending on the needs of your project, you may also want to consider GitHub,[7] Google Code,[8] or SourceForge.[9] Some of the free systems require you to use a specific open source license, so read the fine print first.

[4] http://www.zoneedit.com/
[5] http://www.cvsdude.org/
[6] In case you haven't noticed, I enjoy poking fun at myself and making a big deal of the fact that I'm not exactly a spring chicken. I hope you enjoy my peculiar brand of humor.
[7] http://github.com/
[8] http://code.google.com/hosting/
[9] http://sourceforge.net/

Tools and Libraries

Now that we've reviewed the AWS infrastructure services and made sure that you have the right skills and some technical prerequisites mastered, it's time to gear up and do some work.

In this section I will tell you about the programming libraries, command line tools, and visual tools that are available to you as an AWS developer. In each case I will provide several alternatives in order to make clear that you have a number of reasonable options. Some of the items I'll describe were created and are maintained by Amazon; others are the work of independent developers.

Tool Considerations

Here are some points to consider when choosing your tools:

Documentation
: Are the tools backed by some good documentation? Does the documentation match the current release? Does the maintainer strive to create a complete product or is documentation always lagging behind the code?

Popularity and Reputation
: What comes up when you do a web search for the name of the tool? Can you locate the tool's community without too much trouble?

Community Support
: Check to see if the tools have a strong community behind them. Is there a discussion forum? See what other developers have to say and whether they're contributing or complaining. Also, see if the original creators or current maintainers are participating in the community, or if they're in hands-off mode. Do they enjoy interacting with their users, or has it become a chore?

Update Frequency
: Take a look at the release and patch history for the tool. Are updates released in sync with new releases of AWS? Does the creator release patches on a timely basis in response to problems? Is there a fixed release schedule?

Style

Is the tool a good fit for your working style? Are the command, function, or data structure names consistent and logical? Are your educated guesses correct more often than not?[10]

Security

You'll be trusting these tools with your AWS public and secret keys, so caution is advised. At the very least, you should understand how and where the tool stores your keys and how it protects them from accidental disclosure or casual discovery.

Language Libraries

Language libraries occupy the space between your application and the web service calls. The libraries provide an adapter that matches the structure and style of the language, so that your application code will look and feel *native*. Some libraries provide an object oriented interface to AWS. All the libraries take care of the details involved in making a call, including handling of default parameter values, managing private and public keys, signing the requests, making the web service calls, checking for error conditions, and parsing returned values into native objects. Some libraries also provide functions to retry calls that have failed due to a transient error; others will accumulate data returned by multiple calls into a single return value.

The official Amazon libraries for PHP are very nice, but as of this writing they lack support for some of the services—S3, for example. If you have no requirement for S3 support, these libraries may fit the bill for you. You can find the EC2 libraries[11] (base EC2, CloudWatch, Auto Scaling, and Elastic Load Balancing), as well as a number of additional libraries for PHP and other languages, at the same link.

In this book we'll be using the CloudFusion library written by Ryan Parman. This library supports all the AWS services. CloudFusion supports multi-threaded access and bulk operations, and is supplied in open source form under the BSD license.[12] CloudFusion also includes a number of high-level utility functions to make AWS even easier to use.

---

[10] I could write an entire book on why I'll never use the word "intuitive" to describe this property, but I have to finish this one first.

[11] http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=85

[12] http://www.opensource.org/licenses/bsd-license.php

Here's some code to illustrate how you can use CloudFusion to create an S3 bucket. I'm including this code to show you that CloudFusion is clean and easy to use:

```
#!/usr/bin/php
<?php

error_reporting(E_ALL);
require_once('cloudfusion.class.php');

$s3  = new AmazonS3();
$res = $s3->create_bucket("jeff_barr_bucket");

if ($res->isOK())
{
  print("Bucket created\n");
}
else
{
  print("Error creating bucket\n");
}
?>
```

First, we create a new `AmazonS3` object from which we can access the functions we need. Creating a bucket involves a single method call to the `create_bucket` function, supplying the new bucket's name. We can then test to ensure that there were no errors by using the `isOK` method.

No need to worry about the details right now—we'll be jumping in before too long. The CloudFusion project maintains a GitHub repository[13] and a web site.[14] We'll dive deeper into CloudFusion in the section called "Installing CloudFusion" at the end of this chapter, so it's unnecessary to download or configure it just yet.

Command Line Tools

Not so long ago, typing commands into a shell window was considered the state of the art in human-computer interaction. Although visual tools have obviated much of the need to do this on a routine basis, there are still a good number of reasons to use the command line from time to time. Command line tools are easier to use as part of a script. Routine operations that involve a sequence of commands and some

[13] http://github.com/cloudfusion/cloudfusion
[14] http://getcloudfusion.com/

decision-making can be automated. The output from two or more command line tools can be blended together. All things considered, these tools still serve a purpose.

Amazon supplies a number of command line tools. The first set is called API tools because there's one tool for each function in the EC2 API. For example, the `ec2-run-instances` command is a wrapper around the EC2 `RunInstances` function. The tools are written in Java; the source, however, is unavailable. The EC2 API tools are divided into four packages:

1. The Amazon EC2 API Tools[15] provide access to the core EC2 API functions.
2. The Amazon CloudWatch API Tools[16] provide access to the CloudWatch API functions.
3. The Auto Scaling API Tools[17] provide access to the Auto Scaling API functions.
4. The Elastic Load Balancing API Tools[18] provide access to the Elastic Load Balancing API functions.

Amazon also supplies a set of AMI tools.[19] More special-purpose in nature, you use these tools to create, upload, and register customized Amazon Machine Images (AMIs).

Members of the AWS developer community have created a number of command line tools. Some of these focus on providing access to a single service; others aim to be all-encompassing. A good example of the latter is Tim Kay's top-rated `aws` command.[20] This aspiring Swiss Army Knife now provides functions for EC2 services, S3, and SQS. Written in Perl, this script runs on Linux and Windows.

Visual Tools

At the top of the stack we find the visual tools. Developers have really exercised their creativity in this area and there are many tools to choose from; I'll cover a few of my favorites here. There are at least four types of tools:

[15] http://developer.amazonwebservices.com/connect/entry.jspa?externalID=351&categoryID=251
[16] http://developer.amazonwebservices.com/connect/entry.jspa?externalID=2534&categoryID=251
[17] http://developer.amazonwebservices.com/connect/entry.jspa?externalID=2535&categoryID=251
[18] http://developer.amazonwebservices.com/connect/entry.jspa?
[19] http://developer.amazonwebservices.com/connect/entry.jspa?externalID=368
[20] http://timkay.com/aws/

1. Dynamic Ajax-powered applications—the AWS Management Console is a good example of this type of application.
2. Browser extensions—the ElasticFox and the S3Fox are both extensions to the Firefox web browser.
3. Standalone desktop applications—CloudBerry Explorer, Bucket Explorer, and the SimpleDB Explorer are all in this category.
4. Device-specific applications—the iPhone applications from Ylastic and Direct Thought are both in this category.

AWS Management Console

The AWS Management Console[21] is effectively part of AWS. Because it's part of the AWS site, there's no configuration work required. Once you've logged in, you have full access to all EC2 services, CloudFront, and Elastic MapReduce. You can see your running instances and launch new ones. You can manage security groups, Elastic Block Store volumes, Elastic IP addresses, and identity key pairs. Figure 3.1 shows what it looks like.

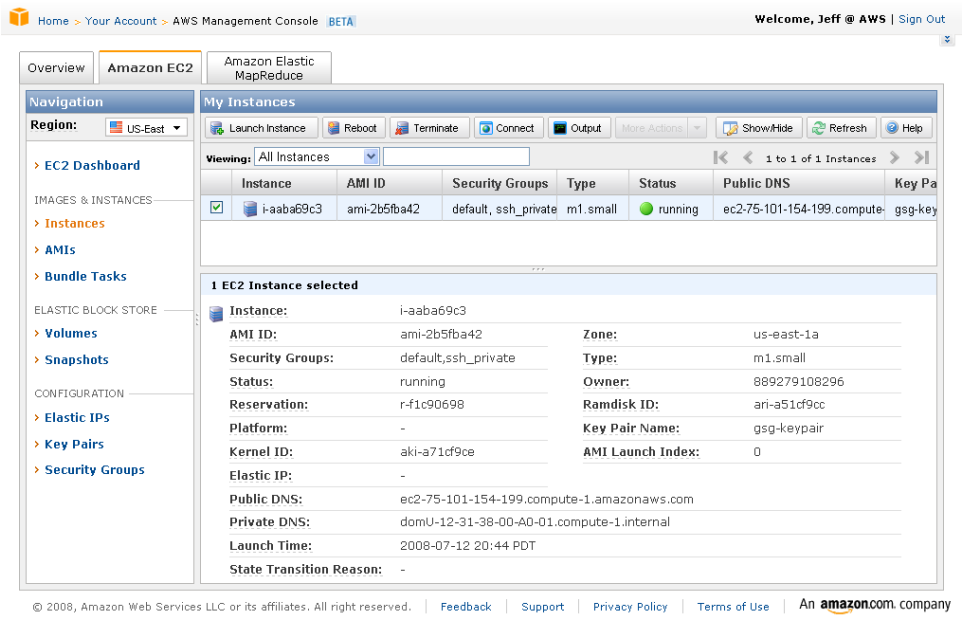

Figure 3.1. The AWS Management Console

[21] http://console.aws.amazon.com

ElasticFox

ElasticFox[22] is a Firefox extension available for free. After installing the extension and restarting your browser, you can access ElasticFox through the **Tools** menu. ElasticFox is an open source tool and has a SourceForge project of its very own.

Figure 3.2 shows ElasticFox listing available AMIs.

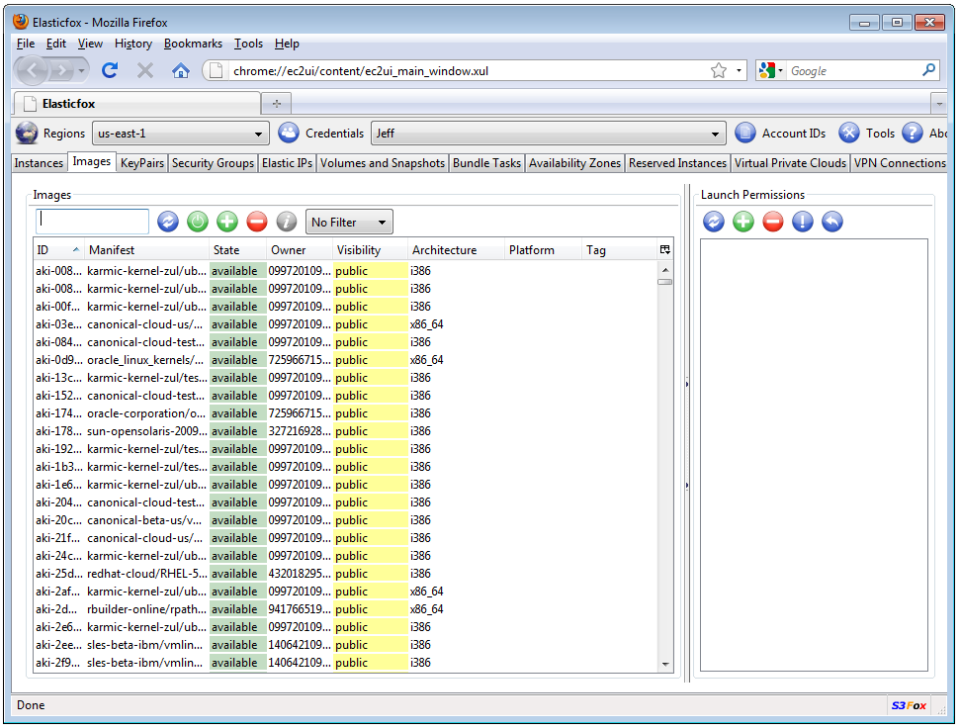

Figure 3.2. ElasticFox displaying a list of available AMIs

[22] http://developer.amazonwebservices.com/connect/entry.jspa?externalID=609

S3Fox

S3Fox[23] is a free S3 and CloudFront extension for Firefox, and a product of Suchi Software Solutions. This too makes it easy to copy files from your desktop to Amazon S3 (and vice versa) using a two-paned interface. Figure 3.3 shows what the S3Fox looks like; the local disk is in the left pane, while the S3 bucket is on the right.

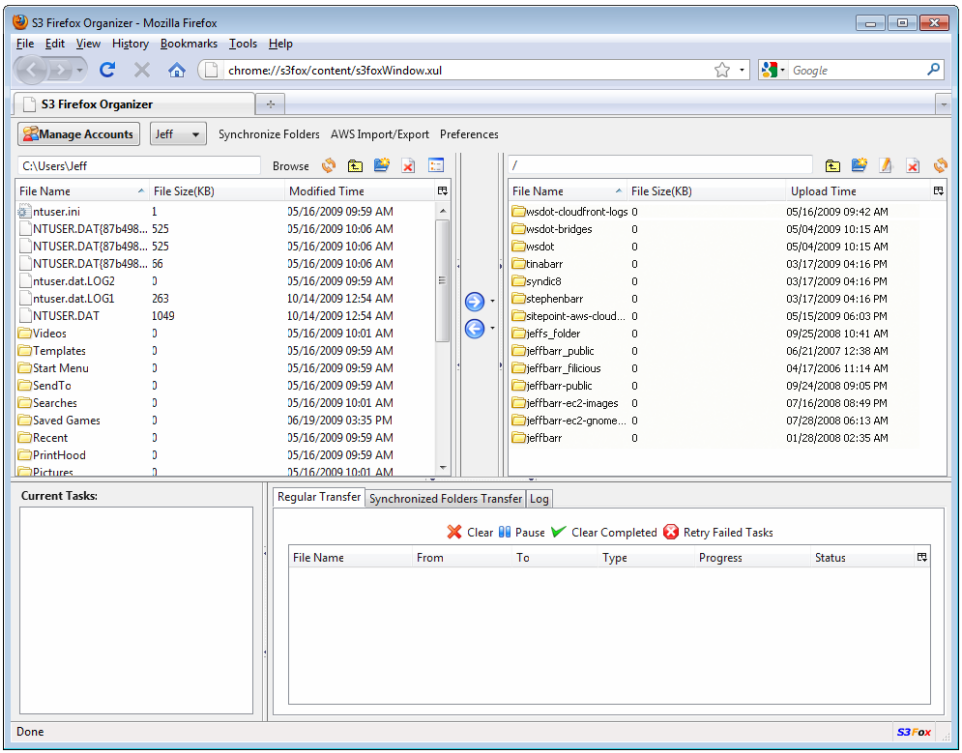

Figure 3.3. The S3Fox Organizer with its dual-paned approach

[23] http://www.s3fox.net/

CloudBerry Explorer

CloudBerry Explorer[24] is a free, standalone desktop application and a product of CloudBerry Lab. You can manage your S3 buckets, copy files into and out of Amazon S3, and create CloudFront distributions. There's also a professional version that supports file encryption, data compression, and access to FTP servers. A screenshot from the free version is shown in Figure 3.4.

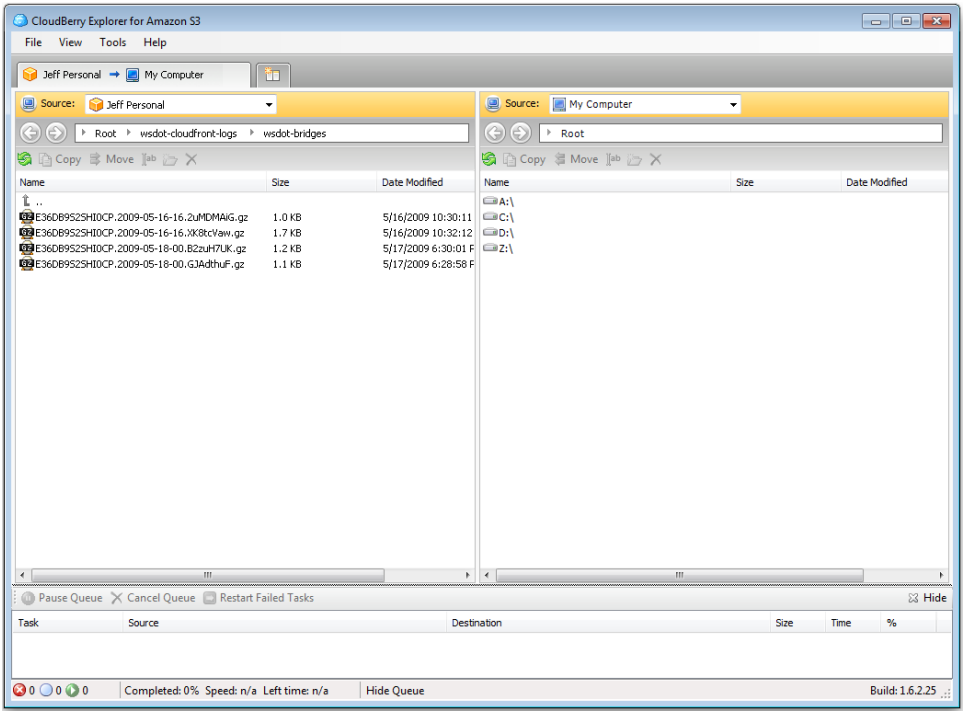

Figure 3.4. CloudBerry Explorer displaying the S3 bucket (left) and local disk (right)

[24] http://cloudberrylab.com/

Bucket Explorer

Bucket Explorer[25] is a commercial desktop application from Chambal.com. You can download free trial versions for Windows, Mac, and Linux. In addition to managing S3 buckets and objects, Bucket Explorer supports shared buckets, local and remote synchronization, and versioning. Figure 3.5 shows a screenshot of Bucket Explorer in action.

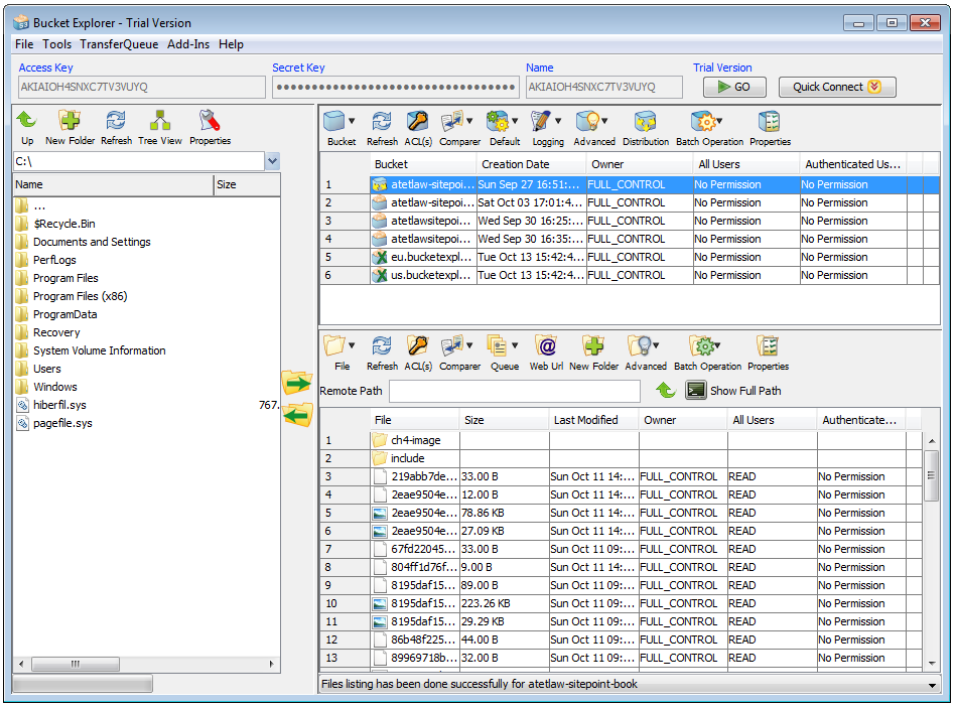

Figure 3.5. Bucket Explorer with a bucket list on top and contents at the bottom

[25] http://www.bucketexplorer.com/

SimpleDB Explorer

SimpleDB Explorer,[26] shown in Figure 3.6, is also produced by Chambal.com and is available in versions for Windows, Mac, and Linux. It features complete read-write access to SimpleDB domains and metadata, along with querying, sorting, and pagination through large result sets.

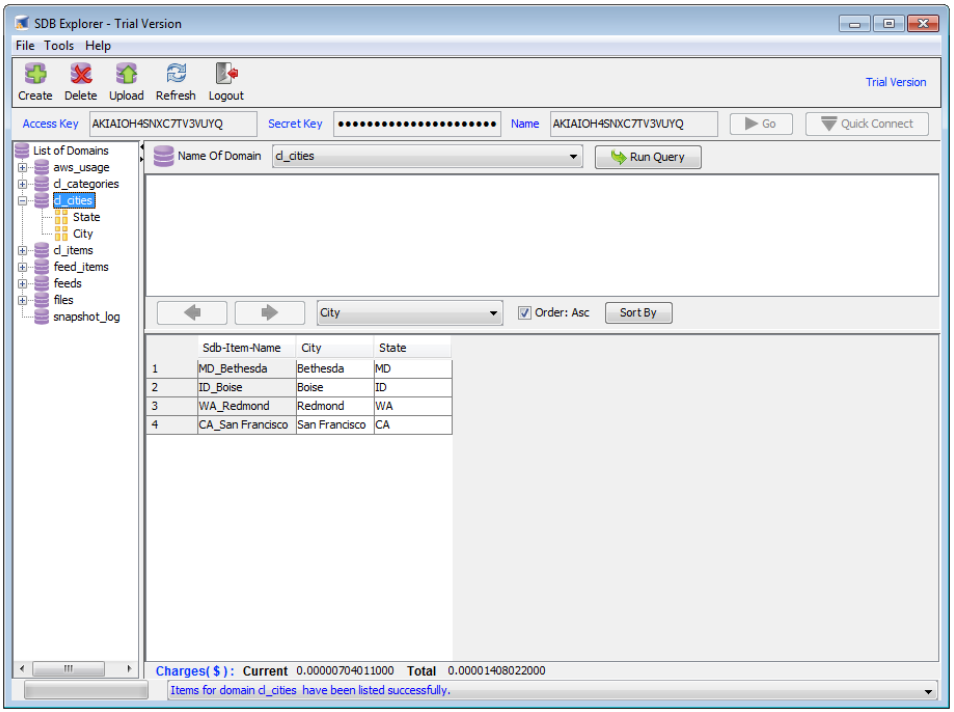

Figure 3.6. The SimpleDB Explorer interface

[26] http://www.sdbexplorer.com/

Ylastic

Ylastic[27] supports complete management of EC2 services, SQS, S3, CloudFront, and SimpleDB on Apple iPhone and phones based on the Google Android operating system (the latter can be seen in Figure 3.7). While it's advisable to use a more capable device as your primary AWS management tool, it can certainly be useful to have a portable management tool at hand.

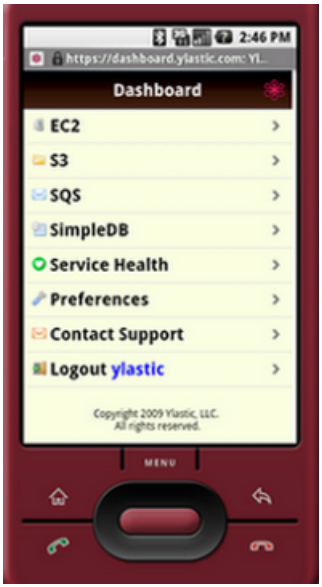

Figure 3.7. Ylastic on the Android phone

[27] http://ylastic.com/

DirectEC2

The Direct Thought company has created DirectEC2,[28] a native interface to EC2 services for the iPhone and the iPod Touch. The application features complete control of each aspect of EC2 functionality: AMIs, running instances, EBS volumes, EBS volume snapshots, Security Groups, IP addresses, and Keypairs. Figure 3.8 demonstrates what it looks like.

Figure 3.8. DirectEC2 on the iPhone

Creating an AWS Account

You're going to need an AWS account in order to run the code in this book. The accounts are free and you'll only be charged for the services that you actually use, but you'll need an email address and credit card in order to create your account.

You can skip this section if you already have an active AWS account.

[28] http://www.directthought.com/launch.html#directEC2

Start by visiting http://aws.amazon.com and click on the button labeled **Sign Up Now**, as shown in Figure 3.9.

Figure 3.9. The Sign Up Now button

On the next page, enter your email address and select the **I am a new user** option, then click on the **Sign in using our secure server** button. Proceed to the registration page. Fill in your name, enter your email address a second time (for verification), choose a secure password, and click on the **Create account** button.

Enter your contact information (including your street address) on the next page, then scroll down a bit so that the AWS license agreement is visible. Read all the fine print and check the checkbox to indicate that you've read and agree to the terms of the license. Scroll down some more and enter the security check code, and click the **Continue** button.

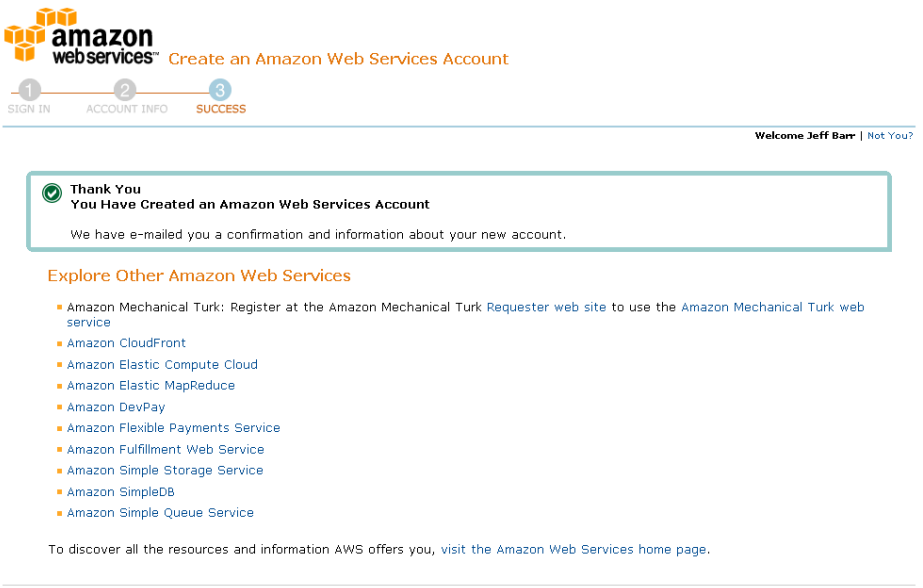

Figure 3.10. This page confirms you've successfully created your AWS account

Once you've done this, you'll see a page like the one shown in Figure 3.10, indicating that you now have an AWS account. Congratulations, and welcome to the club!

There's still a little more left to do, though. We've yet to reach the part where you enter your credit card number. The next step is to sign up for an actual AWS service. Click on the **Amazon Elastic Compute Cloud** button, and then click on the **Sign Up For Amazon EC2** button. You may need to enter your email address and password to access the next page.

The next page includes complete pricing information on EC2 services and S3; you need to use S3 to store your Amazon Machine Images, so this form covers both. Review the pricing information, then scroll down to the bottom of the page and enter your credit card information.[29] Click the **Continue** button and on the next page, select **Use This Address** if the address you already entered is also the billing address for your credit card. Otherwise, enter the appropriate address and click on the **Continue** button. Finally, scroll down to the bottom of the confirmation page and click on the **Complete Sign Up** button.

You'll see the "Thank you for signing up …" page and a notice informing you that you'll receive a sequence of confirmation emails after the sign-up process:

1. the "Welcome to Amazon" email that indicates that you now have an Amazon account
2. the "Welcome to AWS" email that indicates that you now have an AWS account
3. the final pair of emails indicating that you've signed up for EC2 services and S3

Obtaining Your AWS Keys

Once you've created an AWS account, the next step is to access your public and private AWS keys.

Go to the AWS portal at http://aws.amazon.com and select **Security Credentials** from the **Account** menu, as shown in Figure 3.11.

[29] You may want to review the section on Payment Safety and Security on the Amazon web site at http://www.amazon.com/gp/help/customer/display.html?ie=UTF8&nodeId=518224&#safe.

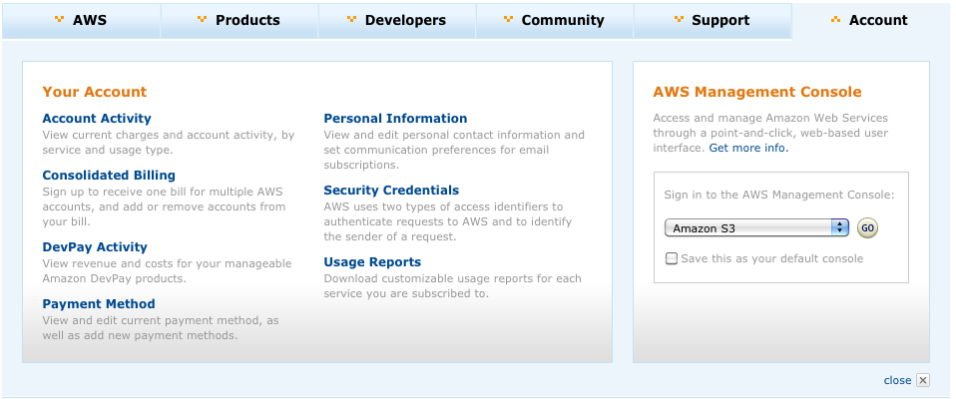

Figure 3.11. Account menu

Enter your email address and password again if necessary. You'll now see the **Security Credentials** page. Scroll down to find the section labeled **Your Access Keys**, as shown in Figure 3.12.

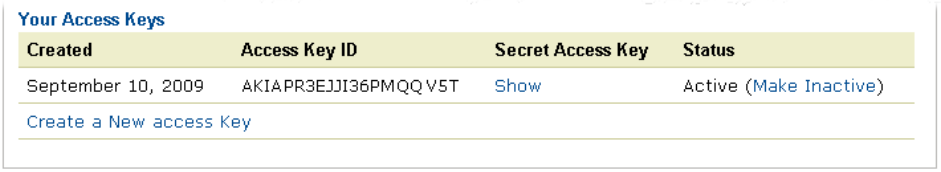

Figure 3.12. Your Access Keys

You're going to need this key in the next section, so select your Access Key ID and then copy and paste it into a scratch file (Notepad or TextEdit is always good for this).

Now click on the **Show** link to see your **Secret Access Key**, like the one shown in Figure 3.13. Again, select it, copy it, and paste it to your scratch file.

Figure 3.13. The Secret Access Key

With this information in hand, you're all set to download and install CloudFusion in the next section.

Running the PHP Code in This Book

All the code in this book is PHP code. We'll be running PHP on the command line and on a web server. Along with the book's assumption about your PHP knowledge, is the assumption that you can install and run PHP and a web server on your specific platform. However, there's just enough room in this chapter to provide a nudge in the right direction.

If you use Windows, look no further than the WampServer package[30] to have you up and running in no time at all. Similarly, MAMP[31] is a one-click installer for Mac OS X. Most Linux distributions already have Apache and PHP installed, but they're also available via your distribution's package manager. If you need more information, have a look at Chapter 1 of Kevin Yank's book *Build Your Own Database Driven Web Site Using PHP and MySQL*, freely available on sitepoint.com.[32]

GD Not Included in Mac OS X

Some of the code in the next chapter makes extensive use of PHP's GD library[33] for image manipulation. Unfortunately, while PHP is included as part of Mac OS X, the GD extension is excluded. To remedy this situation you can either use MAMP, mentioned above, or install the Mac OS X PHP package linked from the PHP web site under the heading **Binaries for other systems**.[34]

We're about to get serious with PHP, but before we do I have to mention this vital information. The PHP programs in this book that commence with a shebang (`#!/usr/bin/php`) are meant to be run from the command line, for example:

```
#!/usr/bin/php
<?php

 ⋮ php script

?>
```

[30] http://www.wampserver.com/en/
[31] http://www.mamp.info/
[32] http://articles.sitepoint.com/article/php-amp-mysql-1-installation/
[33] http://www.php.net/gd/
[34] http://www.php.net/downloads.php

Running PHP scripts from the command line takes the form:

```
$ path_to_php_executable php_script_file [php_script_arguments]
```

The `$` in this example represents the command prompt and is not typed in. Here's an example of running a PHP script on a Linux machine:

```
$ /usr/bin/php list_buckets.php
```

Here's another example, this time on a Mac OS X machine using MAMP:

```
$ /Applications/MAMP/bin/php5/bin/php list_buckets.php
```

Here's an example of running these scripts on a Windows machine from the Windows Command Prompt:

```
C:\> C:\PHP5\php list_buckets.php
```

Further details about running PHP from the Windows command line can be found in the PHP Manual.[35]

Running PHP as a Shell Script

If you're running these scripts on a Linux or Mac OS X machine you can run them like a shell script, but you need to make the files executable first. You can do so using the `chmod` command:

```
$ chmod +x list_buckets.php
```

After doing that you can run your scripts like so:

```
$ ./list_buckets.php
```

In order for this to work, the shebang line in all your scripts must indicate the location of the PHP executable on your system.

Scripts that are meant to be run on your web server will have no shebang:

[35] http://www.php.net/manual/en/install.windows.commandline.php

```
<?php

 ⋮ php script

?>
```

PHP 5.3.0 and Time Zones

If you're using PHP 5.3.0, there is an important change from previous versions. PHP 5.3.0 will now issue a warning-level log message whenever you use a date-related function without specifying a time zone. PHP has always defaulted to the time zone of the machine it's running on, but that behavior is now deemed to be unreliable and so the warning is emitted.

To avoid seeing the warning messages you must specify a time zone when using date functions, or else set a default time zone.[36] The easiest way to set a default time zone is to edit your **php.ini** file, specifically the following line:

```
; Defines the default timezone used by the date functions
;date.timezone =
```

Uncomment the line and add a valid time zone value. You can find a list of time zone values on the PHP web site.[37] Here's an example:

```
; Defines the default timezone used by the date functions
date.timezone = America/Los_Angeles
```

Installing CloudFusion

Many of the examples in this book can be run locally—that is, from your PHP-equipped desktop machine running Windows, Mac OS, or Linux—or remotely on a server or an EC2 instance. Wherever you plan to run them from, that's where you'll need to install CloudFusion. Here are the steps involved:

1. As at the time of writing, the latest version of CloudFusion was 2.5. You can download it from the project's web site at http://getcloudfusion.com. Unzip

[36] http://www.php.net/manual/en/migration52.datetime.php
[37] http://www.php.net/manual/en/timezones.php

the downloaded file archive and move the resulting **cloudfusion** directory to the location where you'd like CloudFusion to be installed.

2. Modify the PHP `include_path` setting in your **php.ini** file; this is to include the full path to the location of the CloudFusion library files.

   On a Linux system, this file is found in the **/etc** directory (as well as on Mac OS X). I put CloudFusion in the **/mnt/cloudfusion** directory, so I edited the `include_path` to look like this:

   ```
   include_path = ".:/php/includes:/mnt/cloudfusion"
   ```

   If you're on a Windows machine your **php.ini** file will be found in your PHP installation folder. If you've placed the CloudFusion files in **c:\cloudfusion**, this is what your `include_path` statement will look like:

   ```
   include_path = ".;c:\php\includes;c:\cloudfusion"
   ```

   You should also create a folder for the source code for all the examples in this book; then, download the code and append that directory name to the `include_path` as well.

3. Now turn your attention to the CloudFusion directory. Copy the file **config-sample.inc.php**, rename it **config.inc.php**, and open this new file for editing. Locate the following statements:

   ```
   define('AWS_KEY', '');
   define('AWS_SECRET_KEY', '');
   ```

   Edit the first statement so that your access key ID is between the quotes. Edit the second statement so that your secret access key is between quotes, like so:

   ```
   define('AWS_KEY', 'your_access_key_id');
   define('AWS_SECRET_KEY', 'your_secret_access_key');
   ```

   If other people can access the computer where you've stored your keys, take appropriate steps to protect both the computer and this file from disclosure.

4. Delete your scratch file.

Where We've Been

In this chapter we talked about the technical prerequisites that you'll need in order to gain the most possible value from the rest of the book. We started tooling up, reviewing a few of the many programming libraries, command line tools, and visual tools available for AWS. We stepped through the AWS registration process and obtained an account with a key to access the web services. We also installed the CloudFusion library. We're now ready to begin coding.

Chapter 4

Storing Data with Amazon S3

In this chapter, we'll dive headfirst into Amazon S3, the Simple Storage Service. After a quick overview of the most important S3 concepts, we'll spend the greater part of this chapter reviewing the code needed to manipulate buckets and objects in S3 and its content-distribution sibling, CloudFront.

S3 Overview

S3 is an Internet-scale data storage service. All data is stored redundantly to guard against problems brought on by temporary connectivity issues or permanent hardware failures. S3 can scale to handle vast amounts of data and deal with a very large number of concurrent accesses. At the time of writing, S3 held over 124 billion objects and was handling over one trillion requests per year.[1]

S3 lets you store your data in containers called **buckets**. A bucket is a named storage entity attached to a particular AWS account. You can create up to 100 buckets in your AWS account. The bucket names must be globally unique, so you'll have to choose with care and keep on trying until you find available names.

[1] http://aws.amazon.com/about-aws/whats-new/2009/03/30/celebrating-3-years-of-amazon-s3-with-3-months-of-transfer-in-for-3-centsgb/

Buckets are used to group any number of S3 objects. When I say *any number*, I really mean it—it is perfectly reasonable to store millions or tens of millions of objects in a single S3 bucket. Depending on your needs, you may choose to use a number of buckets for a single application, or you may choose to use a single bucket for each application.

You can store any type of data you like in S3. It could be text files such as HTML pages, CSS style sheets, or C source code, or binary files such as JPEG images, tar backup files, or even encrypted data (you can encrypt sensitive data before storing it in S3, if you'd like). S3 is ideal for storing web data. Each S3 object has its own URL and S3 can handle a very high request rate.

S3 can store objects of up to 5GB in size. This is a limit that you'll probably never approach in practice, due to various factors. Depending on the speed of your internet connection, it could take hours or even days to upload an object of this size. If the connection was interrupted, you'd have to restart the upload from the beginning. Also, some of the S3 tool kits attempt to read the entire object into memory before uploading it.

Every S3 object has a unique URL formed by concatenating these components:

- protocol (`http://` or `https://`)
- bucket name ending with "`.`"
- S3 endpoint (`s3.amazonaws.com`)
- object key starting with "`/`"

The object key can itself contain "`/`" characters. These characters are simply part of the object's name and have no special meaning to S3. S3 is not a hierarchical file system and has no concept of a subfolder. The S3 tools listed in Chapter 3 maintain the polite fiction that the "`/`" is special and allows the user to traverse of the contents of an S3 bucket as if it contained actual subfolders.

S3 can be accessed using either one of two APIs. The SOAP API contains functions such as `ListAllMyBuckets`, `CreateBucket`, and `DeleteBucket`. The HTTP API uses the standard HTTP verbs (`GET`, `PUT`, `HEAD`, and `DELETE`) as the basis for all operations. The `GET` operation retrieves the contents of a bucket or an object, the `PUT` operation creates a new bucket or object, the `DELETE` operation removes a bucket or an object, and the `HEAD` operation retrieves information about an object.

You can attach metadata in the form of key-value pairs when you create a new S3 object. The metadata is returned in the form of HTTP headers when the object is retrieved using an HTTP `GET` request.

Access to each S3 bucket and object is regulated by an **ACL** or Access Control List. An ACL contains a series of up to 100 **grants**, with each grant consisting of a grantee and a permission. Using the ACL, you can grant access to specific users or to groups of users.

Specific users represent an individual AWS account. The account can be specified using an email address or a canonical user ID, but it is always returned as a `CanonicalUser` object—the email address is simply a convenience.

Groups represent predefined classes of users, any AWS user, or anonymous users (anyone at all).

The permission component of a grant in an ACL specifies the access given to the related grantee. You can grant permissions to buckets or to objects—but be aware that the semantics are slightly different for each, as described in Table 4.1.

Table 4.1. S3 Permission Grants When Applied to Buckets and Objects

Permission	When applied to a bucket	When applied to an object
`READ`	Permission to list the contents of the bucket	Permission to read the object or its metadata
`WRITE`	Permission to create, replace, or delete any object in the bucket	Unsupported for objects
`READ_ACP`	Permission to read the bucket's ACL	Permission to read the object's ACL
`WRITE_ACP`	Permission to overwrite the bucket's ACL	Permission to overwrite the object's ACL
`FULL_CONTROL`	All of the above	All of the above

The owner of a bucket or an object always has implicit `READ_ACP` permission; you can always access the ACL of your own objects. Similarly, the owner also has implicit `WRITE_ACP` permission, so you can always change the ACL of your own object too. The `FULL_CONTROL` permission is provided for convenience and has the same effect as applying the `READ`, `WRITE`, `READ_ACP`, and `WRITE_ACP` permissions.

Permissions on buckets and objects are distinct. The bucket's permissions have no influence on the permissions of newly uploaded objects; ACLs must be set separately for each new object.

The S3 Pricing Model

Your S3 usage is charged in three dimensions:

- amount of data stored
- amount of data transferred in and out of S3
- the number of requests made to S3

Let's examine each of these dimensions in detail.

Your S3 storage charges are based on a unit known as a **gigabyte-month**. If you store exactly one gigabyte for exactly one month, you'll be charged for one gigabyte-month, which is $0.15 (fifteen cents).[2] Time and space can be traded for one another, so you could also store two gigabytes for half of a month for $0.15, or even 30 gigabytes for one day if you'd like. Internally, S3 uses a more fine-grained billing unit, a **byte-hour**. This allows AWS to measure your usage with a very high degree of accuracy and to bill you accordingly.

Your data transfer charges are based on the amount of data transferred in (uploaded) and out (downloaded) from S3. Data transferred in to S3 is charged at a rate of $0.10 per gigabyte. Once again, this amount is prorated. Data transferred out of S3 is charged on a sliding scale starting at $0.17 per gigabyte and decreasing based on volume, reaching $0.10 per gigabyte for all outgoing data transfer in excess of 150 terabytes per month. As an important special case, there is no charge to transfer data between S3 and an Amazon EC2 instance in the same Region.

There are also nominal (yet important) charges for each request made to S3. HTTP `GET` requests are charged at the rate of $0.01 for every 10,000 requests. `PUT`, `COPY`, `LIST`, and `POST` requests are charged at the rate of $0.01 for every 1,000 requests.

Putting it all together, this pricing model means that you can probably prototype and then run your S3-powered application for next to nothing, paying for more storage and more data transfer only as your application becomes popular.

[2] All AWS prices are subject to change. The latest prices for S3 can be found at http://aws.amazon.com/s3.

Watch That Meter!

You need to keep in mind that the meter is always running! If you write a program to poll one of your S3 buckets every second (a `LIST` request) you'll be spending 86 cents every day, whether there's anything new in the bucket or otherwise. You should also check your loop terminating conditions with extreme care. It's better to avoid writing an infinite loop which costs you money with every iteration.

You can (and should) check your S3 usage at any point during the month to verify that it's in accord with your expectations. You can do this by logging into the AWS portal and selecting the **Account Activity** option from the **Your Account** menu.

CloudFront Overview

CloudFront is a web service for content delivery, allowing you to distribute web content at high speed with low latency. CloudFront integrates with S3, making it very easy to distribute any public S3 object to CloudFront's network of 14 global edge locations.[3] As of this writing, the edge locations are spread through the United States, Europe, and Asia, as shown in Table 4.2.

Table 4.2. Edge Locations in the CloudFront Network

United States	Europe	Asia
Ashburn, Virginia	Amsterdam	Hong Kong
Dallas/Fort Worth, Texas	Dublin	Signapore
Los Angeles, California	Frankfurt	Tokyo
Miami, Florida	London	
Newark, New Jersey		
New York, New York		
Palo Alto, California		
Seattle, Washington		
St. Louis, Missouri		

CloudFront is very easy to use. You simply create a distribution for any of your S3 buckets and CloudFront does the rest.

[3] The number of edge locations, and the locations themselves, may change over time.

The CloudFront Pricing Model

Your CloudFront usage is charged in two dimensions:

- data transfer
- the number of requests made to CloudFront

Data transferred in and out of CloudFront is charged on a sliding scale. Starting at a rate of $0.17 per gigabyte, it decreases according to volume, reaching $0.05 per gigabyte for all outgoing data transfer in excess of 1,000 terabytes per month. You're also charged for the data transfer from S3 to CloudFront. The latest prices for CloudFront can be found at http://aws.amazon.com/cloudfront.

There's a charge of $0.013 for every 10,000 HTTP `GET` requests processed by CloudFront.

Programming S3 and CloudFront

With the preliminaries out of the way, it's time to start on the fun part—the code! In this section, you'll learn how to list your buckets, create new buckets, and list the objects in a bucket in several different ways. We'll also cover how to process the contents of a bucket (performing some simple image processing along the way), and how to distribute your content using CloudFront.

Running the Code

As we mentioned in the section called "Running the PHP Code in This Book" in Chapter 3, the programs in this section that start with a shebang (`#!/usr/bin/php`) are meant to be run from the command line. The others are meant to be run via a web server.

Creating an S3 Bucket

Moving right along, let's go ahead and create a new bucket. Before we begin, though, we'll create a new PHP file and call it **book.inc.php**. This will contain a lot of common definitions and functions we'll be using throughout this book. I've placed it in a subfolder called **include**.

The first definition we'll add to our **book.inc.php** file is the name of our S3 bucket:

chapter_04/include/book.inc.php *(excerpt)*

```
<?php

define('BOOK_BUCKET', 'sitepoint-aws-cloud-book');

?>
```

Here we've created a new constant named `BOOK_BUCKET` and given it the value `sitepoint-aws-cloud-book`. Of course, you'll need to decide on your own distinct bucket name; as I mentioned earlier bucket names must be globally unique, and I've already nabbed this one!

Here's the code for creating a new S3 bucket:

chapter_04/create_bucket.php *(excerpt)*

```
#!/usr/bin/php
<?php

error_reporting(E_ALL); (1)

require_once('cloudfusion.class.php'); (2)
require_once('include/book.inc.php');

$s3  = new AmazonS3(); (3)
$res = $s3->create_bucket(BOOK_BUCKET);

if ($res->isOK()) (4)
{
  print("'${bucket}' bucket created\n");
}
else
{
  print("Error creating bucket '${bucket}'\n");
}
?>
```

Let's examine the code statement by statement:

1. We begin by setting the error reporting level. This statement instructs PHP to report all errors and potential errors. This important programming discipline lets you know about problems sooner rather than later. You'll win in the long term because most of the potential bugs will have been shaken out earlier in the piece.

2. Next we include the required files. The first statement pulls the **cloudfusion.class.php** file into memory. This file contains the CloudFusion library used throughout this book. The second statement pulls the **book.inc.php** file into memory.

3. Now we come to the main part of our script. First, we create a new `AmazonS3` object and then call the `create_bucket` method—the key statement in this script—to create a new bucket. If you've customized the definition of `BOOK_BUCKET` in **book.inc.php**, it will create a bucket just for you (assuming that the bucket doesn't already exist).

   One point you should be aware of, is that the CloudFusion library is using your AWS key ID and secret key to access the web service. We configured these values back in the section called "Installing CloudFusion" in Chapter 3.

4. The last statement checks that the call was made to S3, and that S3 returned a status code indicating that the operation was processed successfully. If the bucket exists and you own it, the `create_bucket` call will succeed. If the bucket exists but it belongs to another user, the `create_bucket` call will fail and the `isOK` method will return `FALSE`. Our statement prints an appropriate status message to indicate success or failure.

We can make this script a little bit more useful by allowing the specification of a bucket name on the command line, like so:

chapter_04/create_bucket.php *(excerpt)*

```
#!/usr/bin/php
<?php

error_reporting(E_ALL);

require_once('cloudfusion.class.php');
require_once('include/book.inc.php');

if ($argc != 2) (1)
{
  exit("Usage: " . $argv[0] . " bucket name\n");
}

$bucket = ($argv[1] == '-') ? BOOK_BUCKET : $argv[1]; (2)

$s3  = new AmazonS3();
$res = $s3->create_bucket($bucket);

if ($res->isOK())
{
  print("'${bucket}' bucket created\n");
}
else
{
  print("Error creating bucket '${bucket}'\n");
}

?>
```

(1) First, we check to see how many arguments are supplied. If there are not exactly two, we exit the script and display a helpful usage message.

(2) This program expects the first argument to be a bucket name. If the bucket name is a dash character (`-`), the default bucket (`BOOK_BUCKET`) is used instead.

Running this script with the default bucket's name gives the following output:

```
$php create_bucket.php -
'sitepoint-aws-cloud-book' bucket created
```

Listing Your S3 Buckets

Here's how to list your S3 buckets:

chapter_04/list_buckets.php *(excerpt)*

```
#!/usr/bin/php
<?php

error_reporting(E_ALL);

require_once('cloudfusion.class.php');

$s3 = new AmazonS3();
$buckets = $s3->get_bucket_list(); (1)

foreach ($buckets as $bucket) (2)
{
  print($bucket . "\n");
}

exit(0);
?>
```

The above code should be relatively easy to understand:

(1) This is the key statement that retrieves the list of buckets associated with your account. The method returns an array with one string element for each bucket.

(2) Once we have the list, we can loop through the array and print the name of each bucket using the `print` statement. If you're running this for the first time, you may only see one name: the name of the bucket you created in the last section.

Here's what I see when I run this program from my personal AWS account:

```
$php list_buckets.php
andybarr
aws-dev-relations
biancabarr
carmenbarr

⋮ many more buckets
```

```
sitepoint-aws-cloud-book
sitepoint-aws-cloud-book-thumbs
```

This is a simple yet powerful piece of code. In a few statements we established a connection to S3, retrieved a list of buckets, iterated over the result, and printed the name of each bucket on a line of its own. If all this makes sense, you're well on your way to mastering S3.

Bucket Listing as a Web Page

Since this is a book about web programming, let's get a bit fancier and output the list of buckets as a web page. This version of the script is very similar to the previous **list_buckets.php** script, as we're still using the `get_bucket_list` method:

chapter_04/list_buckets_page.php *(excerpt)*

```
<?php

error_reporting(E_ALL);

require_once('cloudfusion.class.php');

$s3 = new AmazonS3();
$buckets = $s3->get_bucket_list();

$output_title = 'Chapter 3 Sample - List of S3 Buckets';
$output_message = 'A simple HTML list of your S3 Buckets';
include 'include/list_buckets.html.php';

exit(0);
?>
```

The last three statements are new. We set two variables, `$output_title` and `$output_message` with informative content that will be appearing on the web page. The final `include` statement includes the HTML template for our bucket list:

chapter_04/include/list_buckets.html.php *(excerpt)*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title><?php echo $output_title ?></title>
</head>
<body>
  <h1><?php echo $output_title ?></h1>
  <p><?php echo $output_message ?></p>
  <ul>
    <?php foreach($buckets as $bucket): ?>
      <li><?php echo $bucket ?></li>
    <?php endforeach ?>
  </ul>
</body>
</html>
```

In the code above, `$output_title` becomes the page title and top-level heading, while `$output_message` is output as an informative note at the top of the page within a HTML paragraph. The list of buckets in the `$buckets` array is output as an HTML unordered list, as shown in Figure 4.1, with the help of a `foreach` loop.

- andybarr
- aws-dev-relations
- biancabarr
- carmenbarr
- faces
- faces-r
- gracebarr
- hidden-ridge
- jbarr-work
- jbarr_demo
- jbarr_ruby
- jeff_barr_bucket
- jeffbarr
- sitepoint-aws-cloud-book
- sitepoint-aws-cloud-book-thumbs

Figure 4.1. A simple list of S3 buckets

Listing Objects in a Bucket

Now that you know how to create buckets and list these buckets from the command line and from a web page, let's look at the contents of a particular bucket. In this

script we list all the objects within the first bucket we created (using the constant `BOOK_BUCKET`):

chapter_04/list_bucket_objects.php *(excerpt)*

```
#!/usr/bin/php

error_reporting(E_ALL);

require_once('cloudfusion.class.php');
require_once('book.inc.php');

$s3  = new AmazonS3();
$objects = $s3->get_object_list(BOOK_BUCKET);

if($objects) {
  foreach ($objects as $object)
  {
    print($object . "\n");
  }
} else {
  print("No objects found in " . BOOK_BUCKET . "\n");
}

exit(0);
?>
```

Here's the key statement in the above example:

```
$objects = $s3->get_object_list(BOOK_BUCKET);
```

The `get_object_list` method is a simple shortcut method. When given no extra parameters, it will return a list of the first (alphabetically speaking) 1,000 object keys in a bucket. Optional parameters can be used to start returning keys alphabetically after a given key or to filter the list of keys after they've been retrieved from S3. Like `get_bucket_list`, this method returns an array with one key in string form per element.

You're probably wondering why this function returns just 1,000 keys. Remember that buckets can contain millions of objects, but returning such a long list all at once would be problematic on several levels—simply transmitting the lists would take

a long time. As we'll see in the next section, a web-scale system like S3 must provide a way to access long lists in chunks of a reasonable size.

Of course, your newly created bucket has no objects in it at this time, so this script will do little more than alert you to that fact. To give it a more complete test run, you can either add some objects to your bucket using one of the tools we saw in the section called "Visual Tools" in Chapter 3 (such as S3Fox or CloudBerry Explorer), or skip ahead to the section called "Uploading Files to S3" later in this chapter.

Finally, here's a challenge: based on the **create_bucket.php** script we developed above, it should be fairly easy for you to change the **list_bucket_objects.php** script, so that you can specify a bucket name on the command line.

Processing Complex CloudFusion Data Structures

It's time to take a little detour.

The CloudFusion functions that I have told you about thus far all return simple data structures—arrays of strings. However, the functions that I will use later in this chapter return a more complex data structure called a `ResponseCore`. S3 returns its results in the form of an XML document; CloudFusion parses the XML using PHP's SimpleXML package and includes the parsed objects in the response where they can be referenced by name.[4]

The following code calls S3 to list the first 1,000 objects in the bucket `BOOK_BUCKET`, and then calls PHP's handy `print_r` function to display the resulting object tree:

chapter_04/list_bucket_objects_raw.php *(excerpt)*

```
#!/usr/bin/php
<?php

error_reporting(E_ALL);

require_once('cloudfusion.class.php');
require_once('include/book.inc.php');

$s3 = new AmazonS3();
$res = $s3->list_objects(BOOK_BUCKET);
```

[4] Complete documentation for SimpleXML can be found at http://www.php.net/simplexml.

```
print_r($res);
exit(0);
?>
```

The resulting output is far too long to display in its entirety (465 lines for my buckets). Let's look at some excerpts instead. Here's the first part:

```
$php list_bucket_objects_raw.php
ResponseCore Object
[header] => Array
(
  [x-amz-id-2] => Ya7yAuUClv7HgR6+JJpzOsYDM1m4/Zy+dORmk5cSAu+qV+v+6
➥9gLSHlytlD77wAn
  [x-amz-request-id] => 14AA13F3FOB76032
  [date] => Thu, 28 May 2009 06:51:26 GMT
  [content-type] => application/xml
  [transfer-encoding] => chunked
  [connection] => close
  [server] => AmazonS3
  [_info] => Array
  (
    [url] => https://sitepoint-aws-cloud-book.s3.amazonaws.com/
    [content_type] => application/xml
    ⋮
```

The first line indicates that the data is of type `ResponseCore`. Further on, we find some standard PHP arrays. If we need to, we can access the data like this:

```
$res->header['transfer-encoding']
$res->header['_info']['url']
```

`$res` is an object and `header` is one of the object's instance variables, so it's accessed using the `->` operator. The `header` instance variable is a PHP array, so its members are accessed using the array syntax.

In the second line the `_info` member of `header` is itself an array, so a second set of brackets are used to access the `url` value inside.

A little bit further down in the output, we find the following:

```
[body] => SimpleXMLElement Object
(
  [Name] => sitepoint-aws-cloud-book
  ⋮
```

The `body` instance variable is of type `SimpleXMLElement`. It starts out with a `Name` instance variable, which can be accessed as `$res->body->Name`.

Even further down we finally find what we came here for—the list of objects in the bucket:

```
[Contents] => Array
(
  [0] => SimpleXMLElement Object
  (
    [Key] => images/2008_shiller_housing_projection.jpg
    [LastModified] => 2009-05-22T23:44:58.000Z
    [ETag] => "e2d335683226059e7cd6e450795f3485"
    [Size] => 236535
    [Owner] => SimpleXMLElement Object
    (
      [ID] => 7769a42be4e57a034eeb322aa8450b3536b6ca56037c06ef19b1e1
➥eabfeaab9c
      [DisplayName] => jeffbarr
    )
    [StorageClass] => STANDARD
  )
  ⋮
```

You can see that `body` contains an instance variable called `Contents`, which is another array containing all the files in the bucket. Each file in the bucket is represented by a `SimpleXMLElement` object; each has `Key`, `ETag`, `Size`, `Owner`, and `StorageClass` members, accessed like this:

```
$res->body->Contents[0]->Key
$res->body->Contents[0]->ETag
$res->body->Contents[0]->Size
$res->body->Contents[0]->Owner->ID
$res->body->Contents[0]->Owner->DisplayName
$res->body->Contents[0]->StorageClass
```

Of course, you're free to use intermediate variables to make this code shorter or more efficient.

You may be wondering where the object names (`Contents`, `Key`, `Size`, and so forth) come from. The `list_objects` method makes an HTTP `GET` request to S3 to fetch a list of the first 1,000 objects in the bucket. The request returns an XML document, and CloudFusion parses and returns it as the `body` object. The object names are taken directly from the XML tags in the document.

If we were to modify the previous script to print out some of these values, it may look like this example:

```
#!/usr/bin/php
<?php

error_reporting(E_ALL);

require_once('cloudfusion.class.php');
require_once('include/book.inc.php');

$s3 = new AmazonS3();
$res = $s3->list_objects(BOOK_BUCKET);

print("Bucket Url: " . $res->header['_info']['url'] . "\n");
print("Bucket Name: " . $res->body->Name   . "\n");
print("First Key:  " . $res->body->Contents[0]->Key . "\n");
print("Second Key: " . $res->body->Contents[1]->Key . "\n");
exit(0);
?>
```

In the above example we output the bucket's URL and name, followed by the keys of the first two items in the bucket.

We have now come to the end of the detour. I hope that the ride was scenic, yet educational. Next, we will use this newfound knowledge to create a very handy utility function.

Listing Objects in a Bucket as a Web Page

Before we can write a script that outputs a list of all the objects in a bucket within a web page, we first have to write a rather complex function. We'll add this function to our **book.inc.php** file and call it `getBucketObjects`:

chapter_04/include/book.inc.php *(excerpt)*

```
function getBucketObjects($s3, $bucket, $prefix = '') 1
{
  $objects = array();
  $next = '';

  do 2
  {
    $res = $s3->list_objects($bucket,
        array('marker' => urlencode($next),
            'prefix' => $prefix)
      ); 3

    if (!$res->isOK()) 4
    {
      return null;
    }

    $contents = $res->body->Contents; 5
    foreach ($contents as $object)
    {
      $objects[] = $object;
    }

    $isTruncated = $res->body->IsTruncated == 'true'; 6

    if ($isTruncated)
    {
      $next = $objects[count($objects) - 1]->Key; 7
    }
  }
  while ($isTruncated);

  return $objects; 8
}
```

This function is more complex than anything you've seen so far, but there's no need to worry. Earlier in this chapter I told you that one "list bucket" request to S3 will return at most 1,000 keys, regardless of how many keys are in the bucket. Our `getBucketObjects` function simply calls `list_objects` again and again until S3 says that there are no more objects to return:

1. Our function accepts three arguments: an `AmazonS3` object, an S3 bucket, and a prefix value that defaults to an empty string.
2. We use a `do … while` loop, so that the body of the loop always runs at least once.
3. Each time I call `list_objects`, I pass in a value called `$next`. The first time through the loop, `$next` is an empty string, and `list_objects` starts at the beginning (alphabetically speaking) of the bucket. On subsequent loop iterations, `$next` is set to the final key returned on the previous iteration. This tells S3 to start retrieving keys alphabetically following the previous iteration's final key.
4. If the `list_objects` call fails, the function returns null.
5. We retrieve the `Contents` array from the body of the response returned to our `list_objects` call, then loop through the values storing each one in the `$objects` array. This array will eventually be our return value.
6. The data returned by a call to `list_objects` includes an element named `IsTruncated`. If this value is the string `"true"`, the listing is incomplete and there are more objects to be found. This condition is also used to control the loop.
7. If the list is incomplete, we set the `$next` value ready to begin the next iteration.
8. When the loop terminates, the `$objects` array is returned.

Put it together and this function fetches all the objects in the bucket, puts them all into one array, and returns the array.

Avoid Going Loopy

I will freely admit that I failed to correctly state the termination condition when I first wrote this code. I knew that this would be tricky, so I used a print statement at the top to ensure that I avoided creating a non-terminating loop that would spin out of control and run up my S3 bill. I advise you to do the same when you're building and testing any code that costs you money to execute.

With this function in hand, creating a list of the objects in the bucket becomes easy. Here's all we have to do:

chapter_04/list_bucket_objects_page.php *(excerpt)*

```
<?php

error_reporting(E_ALL);

require_once('cloudfusion.class.php');
require_once('include/book.inc.php');

$bucket = IsSet($_GET['bucket']) ? $_GET['bucket'] : BOOK_BUCKET; (1)

$s3 = new AmazonS3();

$objects = getBucketObjects($s3, $bucket); (2)

$fileList = array();

foreach ($objects as $object) (3)
{
  $key = $object->Key;
  $url = $s3->get_object_url($bucket, $key);
  $fileList[] = array('url' => $url, 'name' => $key,
                      'size' => number_format((int)$object->Size)); (4)
}

$output_title = "Chapter 3 Sample - List of S3 Objects in Bucket' .
    '${bucket}'";
$output_message = "A simple HTML table displaying of all the' .
    ' objects in the '${bucket}' bucket.";
include 'include/list_bucket_objects.html.php'; (5)

exit(0);
?>
```

This code generates a web page and can accept an optional `bucket` argument in the URL query string. Let's rip this one apart and see how it works:

(1) This code checks to see if the `bucket` argument was supplied. If it was, then it's used as the value of `$bucket`. Otherwise, the default value, the `BOOK_BUCKET` constant, is used.

(2) Here we call our custom `getBucketObjects` function that fetches the list of objects in the given bucket and stores them in the `$objects` array.

3. The next step is to iterate over the array and process each one.

4. We store three values for each object in the `$fileList` array: the object's URL, key (which we store as `name`), and size (converted to an integer and formatted like a number).

5. We include our HTML template to output the values in the `$fileList` array.

Here's what the **list_bucket_objects.html.php** HTML template looks like:

chapter_04/include/list_bucket_objects.html.php *(excerpt)*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title><?php echo $output_title ?></title>
</head>
<body>
  <h1><?php echo $output_title ?></h1>
  <p><?php echo $output_message ?></p>
  <table>
    <thead>
      <tr><th>File</th><th>Size</th></tr>
    </thead>
    <tbody>
    <?php foreach($fileList as $file): ?>
      <tr>
          <td><a href="<?php echo $file['url'] ?>">
              <?php echo $file['name'] ?></a>
          </td>
          <td><?php echo $file['size'] ?></td>
      </tr>
    <?php endforeach ?>
    </tbody>
  </table>
</body>
</html>
```

The template iterates over the `$fileList` array and creates a table row for each file, placing a link to the file in the first column and the file size in the second column.

Figure 4.2 shows what it looks like (I had already uploaded some files to my bucket).

images/2008_shiller_housing_projection.jpg	236,535
images/3DoorScion.jpg	61,918
images/AndyBass.jpg	73,388
images/AndyWorkstation.jpg	80,841
images/Jeffbase.gif	37,561
images/Jeffbase.psd	545,419
images/Jeffbasebright.gif	37,452
images/Jeffbasebright.psd	532,308
images/STP65708.JPG	1,716,926
images/Scion XB Stretch.jpg	66,598
images/Scion XB.jpg	70,882

Figure 4.2. Listing objects in an S3 bucket

You may have spotted the fact that we now have all the parts needed to make a simple S3 file browser. I'll leave that as a challenge to you. With just a little bit of work you should be able to connect **list_buckets_page.php** and **list_bucket_objects_page.php**.

Uploading Files to S3

Now that we know how to obtain information about buckets and their objects from S3, let's figure out how to put new objects into S3. This is quite easy; we just need two more utility functions that we'll add to our **book.inc.php** file.

The first function is called `uploadObject`:

chapter_04/include/book.inc.php *(excerpt)*

```
function uploadObject($s3, $bucket, $key, $data,
    $acl = S3_ACL_PRIVATE, $contentType = "text/plain")
{
  $res = $s3->create_object($bucket,
      array(
        'filename'    => $key,
        'body'        => $data,
        'acl'         => $acl,
```

```
        'contentType' => $contentType
      ));
  return $res->isOK();
}
```

The `uploadObject` function accepts between four and six parameters. The first four specify the S3 access object, the destination bucket, the desired object key, and the data to be stored in the object. The final two specify a non-default ACL and a content type for the stored object.

Web browsers use the object's content type to figure out how to display the object. For example, a content type of `image/png` tells the browser that the object is an image and that it is in PNG format.

As you can see, the `uploadObject` function simply calls the `create_object` method and checks the returned value to ensure that the object was actually stored in S3.

If S3 is unable to store the object, it will return an `HTTP 500` (internal server error) code. This is almost always a recoverable condition; the proper response is to make several attempts with increasing time delays between attempts (sometimes known as **exponential backoff**). Here's a more sophisticated version of the `uploadObject` function, this one with a retry mechanism:

```
function uploadObject($s3, $bucket, $key, $data,
    $acl = S3_ACL_PRIVATE, $contentType = "text/plain")
{
  $try = 1;
  $sleep = 1;
  do
  {
    $res = $s3->create_object($bucket,
        array(
          'filename'    => $key,
          'body'        => $data,
          'acl'         => $acl,
          'contentType' => $contentType
        ));

    if ($res->isOK()) {
      return true;
    }
```

```
    sleep($sleep);
    $sleep *= 2;
  }
  while(++$try < 6);
  return false;
}
```

This version of our function will try up to six times to create a new object in a bucket. Each time it tries, the length of the pause (before the next try) doubles.

Our next function helps us determine a file's content type:

chapter_04/include/book.inc.php *(excerpt)*

```
function guessType($file)
{
  $info = pathinfo($file, PATHINFO_EXTENSION);
  switch (strtolower($info))
  {
    case "jpg":
    case "jpeg":
      return "image/jpg";

    case "png":
      return "image/png";

    case "gif":
      return "image/gif";

    case "htm":
    case "html":
      return "text/html";

    case "txt":
      return "text/plain";

    default:
      return "text/plain";
  }
}
```

Given a filename, this function uses the file's extension to make a very simple guess as to the content type of the file. There's no inspection of the file's content at all,

and the function expects the file extension to accurately reflect the contents. To serve its purpose for this chapter the function handles just a few types.

Putting it all together with some argument processing, looping, and error checking, we have a handy command to upload one or more files to S3:

chapter_04/upload_file.php *(excerpt)*

```
#!/usr/bin/php
<?php

error_reporting(E_ALL);

require_once('cloudfusion.class.php');
require_once('include/book.inc.php');

if ($argc < 3) 1
{
  exit("Usage: " . $argv[0] . " bucket files...\n");
}

$bucket = ($argv[1] == '-') ? BOOK_BUCKET : $argv[1]; 2

$s3  = new AmazonS3();

for ($i = 2; $i < $argc; $i++) 3
{
  $file        = $argv[$i];
  $data        = file_get_contents($file);
  $contentType = guessType($file);

  if (uploadObject($s3, $bucket, $file, $data, 4
       S3_ACL_PUBLIC, $contentType))
  {
    print("Uploaded file '${file}' to bucket '{$bucket}'\n");
  }
  else
  {
    exit("Could not upload file '${file}'" .
         " to bucket '{$bucket}'\n");
  }
}
```

```
exit(0);
?>
```

This script is designed to be run from the command line like so:

```
$php upload_file.php bucket_name file_name […]
```

Let's take a closer look:

1. First, we check to see if there are any arguments supplied; if there are none we display a helpful usage message and exit the script.
2. This program expects the first argument to be a bucket name. If the bucket name is a dash character (`-`), the default bucket (`BOOK_BUCKET`) is used instead.
3. The remaining arguments are considered to be the files to upload. We then loop through all the inputted filenames.
4. Within each loop we call our `uploadObject` function. I chose to use the path name to each object exactly as supplied, to enable usage of key names with embedded slash characters.

Here's what this program looks like in action:

```
$php upload_file.php - images/catatonia_album.jpg
Uploaded file 'images/catatonia_album.jpg' to bucket
➥'sitepoint-aws-cloud-book'
```

We have really covered quite a bit of ground in just a few pages. You should now understand how to upload images to an S3 bucket, browse the bucket, and then view the images by clicking on a link. In the next section, you will learn how to generate thumbnail versions of all the images in a bucket.

Creating and Storing Thumbnail Images

Our next utility function will take an in-memory image, figure out the appropriate height and width for a thumbnail, and then create the thumbnail.

Firstly, in our **book.inc.php** file, we need to create two constants; one to store the desired thumbnail size, and one to store the default name for the bucket that will store our thumbnails. I want the thumbnail images to be 200 pixels on the longest side, and I want to define a suffix to add to my default bucket name in order to create the thumbnail bucket name, so I'll add the following:

chapter_04/include/book.inc.php *(excerpt)*

```
define('THUMB_SIZE', 200);
define('THUMB_BUCKET_SUFFIX', '-thumbs');
```

Of course, before we can use the thumbnail bucket we have to make sure it exists! You can use the **create_bucket.php** script we developed earlier to do that.

Here's the code for the `thumbnailImage` function; once again, this goes into our **book.inc.php** file:

chapter_04/include/book.inc.php *(excerpt)*

```
function thumbnailImage($imageBitsIn, $contentType)
{

  $imageIn = ImageCreateFromString($imageBitsIn);
  $inX = ImageSx($imageIn);
  $inY = ImageSy($imageIn);

  if ($inX > $inY)
  {
    $outX = THUMB_SIZE;
    $outY = (int) (THUMB_SIZE * ((float) $inY / $inX));
  }
  else
  {
    $outX = (int) (THUMB_SIZE * ((float) $inX / $inY));
    $outY = THUMB_SIZE;
  }

  $imageOut = ImageCreateTrueColor($outX, $outY);
  ImageFill($imageOut, 0, 0,
      ImageColorAllocate($imageOut, 255, 255, 255));
  ImageCopyResized($imageOut, $imageIn,
       0, 0, 0, 0,
       $outX, $outY, $inX, $inY);
```

```
  $fileOut = tempnam("/tmp", "aws") . ".aws";

  switch ($contentType)
  {
    case "image/jpg":
      $ret = ImageJPEG($imageOut, $fileOut, 100);
      break;

    case "image/png":
      $ret = ImagePNG($imageOut, $fileOut, 0);
      break;

    case "image/gif":
      $ret = ImageGIF($imageOut, $fileOut);
      break;

    default:
      unlink($fileOut);
      return false;
  }

  if (!$ret)
  {
    unlink($fileOut);
    return false;
  }

  $imageBitsOut = file_get_contents($fileOut);
  unlink($fileOut);
  return $imageBitsOut;
}
```

I'll refrain from going through this code in detail—this chapter focuses on S3, rather than graphics programming. The code makes extensive use of PHP's GD library.[5] Put simply, it creates a copy of an image and resizes it so that its longest dimension is equal to the number of pixels specified in the `THUMB_SIZE` constant, while preserving the width-height ratio of the image.

Generating the thumbnail for a good-sized image can take a substantial fraction of a second due to the number of pixels to be moved around; also, the `ThumbNailImage`

[5] http://www.php.net/gd

function must write the new image to a temporary file and then read it back into memory.

With this code in hand it's now a simple matter to do some argument processing and thumbnail each image in the given bucket. Here's how to do it:

chapter_04/thumbnail_bucket.php *(excerpt)*

```
#!/usr/bin/php
<?php

error_reporting(E_ALL);

require_once('cloudfusion.class.php');
require_once('include/book.inc.php');

if ($argc != 3) 1
{
  exit("Usage: " . $argv[0] . "in-bucket out-bucket\n");
}

$bucketIn  = ($argv[1] == '-') 2
             ? BOOK_BUCKET
             : $argv[1];

$bucketOut = ($argv[2] == '-') 3
         ? $bucketIn . THUMB_BUCKET_SUFFIX
             : $argv[2];

print("Thumbnailing '${bucketIn}' to '${bucketOut}'\n");

$s3  = new AmazonS3();
$objectsIn = getBucketObjects($s3, $bucketIn); 4

foreach ($objectsIn as $objectIn)
{
  $key = $objectIn->Key;
  print("Processing item '${key}':\n");

  if (substr(guessType($key), 0, 6) == "image/") 5
  {
    $startTime   = microtime(true); 6
    $dataIn      = $s3->get_object($bucketIn, $key); 7
    $endTime     = microtime(true);
    $contentType = guessType($key);
```

```
    printf("\tDownloaded from S3 in %.2f seconds.\n",
     ($endTime - $startTime));

    $startTime = microtime(true);
    $dataOut   = thumbnailImage($dataIn->body, $contentType); 8
    $endTime   = microtime(true);

    printf("\tGenerated thumbnail in %.2f seconds.\n",
     ($endTime - $startTime));

    $startTime = microtime(true);
    if (uploadObject($s3, $bucketOut, $key, $dataOut, 9
         S3_ACL_PUBLIC, $contentType))
    {
      $endTime = microtime(true);

      printf("\tUploaded thumbnail to S3 in %.2f seconds.\n",
       ($endTime - $startTime));
    }
    else
    {
      print("\tCould not upload thumbnail.\n");
    }
  }
  else
  {
    print("\tSkipping - not an image\n");
  }
  print("\n");
}
exit(0);
?>
```

Let's go through this code step by step:

1. First, we need to ensure that we have the minimum requirement of three arguments: the name of the script, the image bucket name, and the thumbnail bucket name.

2. Much like the previous example, this code allows the input and output buckets to be set to default values by using the "-" character for the first two command line arguments.

3. The default bucket name for our thumbnail bucket will be our default bucket name plus the suffix we defined earlier.

4. Here we use our `getBucketObjects` function to retrieve all the objects in our bucket.

5. We'll make use of our `guessType` function to ensure we only process objects that have a content type beginning with "`image/`".

6. This code might look a bit cluttered because I've wrapped each major operation in timing code like this:

```
$startTime   = microtime(true);
⋮ code operation
$endTime     = microtime(true);
```

   This allows us to print useful time-related information as our script runs. Subtracting `$startTime` from `$endTime` will provide us with the elapsed time that we can format for output:

```
printf("Completed in %.2f seconds.\n", ($endTime - $startTime));
```

7. There's just one new S3 call here:

```
$dataIn = $s3->get_object($bucketIn, $key);
```

   The `get_object` method downloads the object from S3 and returns its data as a string.

8. Here our `thumbnailImage` function generates a thumbnail and stores it in `$dataOut`.

9. Our `uploadObject` function uploads the generated thumbnail to our thumbnail bucket.

Here's an example of the output produced when a single file is "thumbnailed":

```
$php thumbnail_bucket.php - -
Processing item 'images/a380_factory.jpg':
  Downloaded from S3 in 0.78 seconds.
  Generated thumbnail in 0.19 seconds.
  Uploaded thumbnail to S3 in 0.09 seconds.
```

The combination of S3, PHP, and GD lets you do some powerful graphics processing without a whole lot of work. You could easily modify the thumbnail code to reduce the quality of the image, paint a watermark over it, or map colors to grayscale values.

Creating a CloudFront Distribution

Before we can work with CloudFront Distributions, you have to make sure you've activated the feature in your AWS account. Visit http://aws.amazon.com/cloudfront/ and then click the **Sign Up for Amazon CloudFront** button, activation is only one more click away.

Although it's possible to write code to create a CloudFront distribution for an S3 bucket (and you're welcome to, of course), I'm choosing to sidestep it. Instead, it's easier to just use a graphical interface to create your distribution. You will be able to do this in any of the tools mentioned in the section called "Visual Tools" in Chapter 3. The AWS Management Console[6] makes it very simple; select the **Amazon CloudFront** tab and click the **Create Distribution** button, as shown in Figure 4.3.

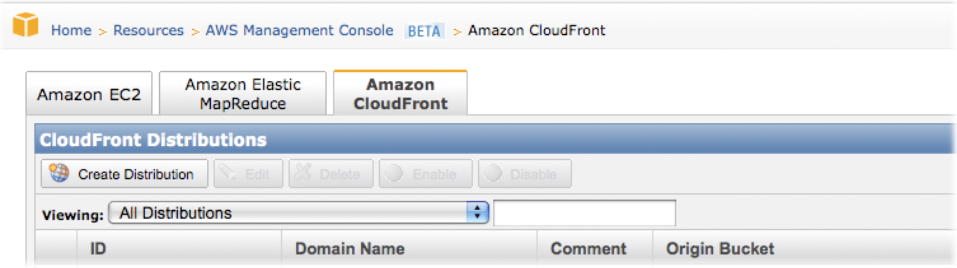

Figure 4.3. Click the Create **Distribution** button

Once set up, you'll have to wait several minutes while the CloudFront distribution is created.

[6] http://console.aws.amazon.com

Let's wrap up this chapter by making an image browser. We'll use CloudFront for efficient global distribution of the original and thumbnailed images.

Listing CloudFront Distributions

Here's a simple script to list all of your CloudFront distributions:

chapter_04/list_distributions.php *(excerpt)*

```
#!/usr/bin/php
<?php

error_reporting(E_ALL);

require_once('cloudfusion.class.php');
require_once('include/book.inc.php');

$cf = new AmazonCloudFront();
$res = $cf->list_distributions();

if (!$res->isOK())
{
  exit("Could not retrieve list of CloudFront distributions\n");
}

$distributions = $res->body->DistributionSummary;

printf("%-16s %-32s %-40s\n", "ID", "Domain Name", "Origin");
printf("%'=-16s %'=-32s %'=40s\n", "", "", "");

foreach ($distributions as $distribution)
{
  $id         = $distribution->Id;
  $domainName = $distribution->DomainName;
  $origin     = $distribution->Origin;

  printf("%-16s %-32s %-40s\n", $id, $domainName, $origin);
}
exit(0);
?>
```

The code structure should be familiar to you by now. The `list_distributions` method returns an array of objects, which are then iterated over and printed. Notice, though, that we instantiate a new `AmazonCloudFront` object instead of an `AmazonS3`

object like the previous scripts, and the `list_distributions` method to retrieve the details of the distributions.

The response object can be queried in the same way as our previous S3 scripts. In the response returned to the `list_distributions` method, the `$res->body->DistributionSummary` property will contain an array of `SimpleXMLElement` objects, one for each CloudFront distribution. The script above simply iterates over this array and extracts the `Id`, `DomainName`, and `Origin` property of each for display in the output, formatted into a table similar to the following:

```
$php list_distributions.php
Id               Domain Name                        Origin
=============== ================================= ==============
➥==========================
nnnnnnnnnnnnnn   nnnnnnnnnnnnnn.cloudfront.net    sitepoint-aws-
➥cloud-book.s3.amazonaws.com
```

When you run this script with your AWS account, the `ID` and `Domain Name` columns will show your unique values.

Listing S3 Files with Thumbnails

Okay, time for the last script in this chapter! We first need one more utility function to make this work: the `findDistributionForBucket` function that will return the CloudFront distribution for a given S3 bucket. You guessed it, we'll put this one in our **book.inc.php** file:

chapter_04/include/book.inc.php *(excerpt)*

```
function findDistributionForBucket($cf, $bucket)
{

  $res = $cf->list_distributions();

  if (!$res->isOK())
  {
    return null;
  }

  $needle = $bucket . ".";
  $distributions = $res->body->DistributionSummary;
```

```
  foreach ($distributions as $distribution)
  {
    if (substr($distribution->Origin, 0, strlen($needle)) ==
        $needle)
    {
      return $distribution;
    }
  }

  return null;
}
```

This function accepts a CloudFront access object and the name of a bucket. It fetches the list of CloudFront distributions and attempts to match each one to the supplied bucket name. If a match is made, the distribution object is returned.

The code below is an enhanced version of **list_bucket_objects_page.php**, as seen earlier in this chapter. It adds a thumbnail to the table for all the image objects in the bucket that also have a corresponding image in the thumbnail bucket. It also uses the CloudFront URL if available:

chapter_04/list_bucket_objects_page_thumbs.php *(excerpt)*

```
<?php

error_reporting(E_ALL);

require_once('cloudfusion.class.php');
require_once('include/book.inc.php');

$bucket = IsSet($_GET['bucket']) ? $_GET['bucket'] : BOOK_BUCKET;
$bucketThumbs = $bucket . THUMB_BUCKET_SUFFIX;

$s3 = new AmazonS3(); (1)
$cf = new AmazonCloudFront();

$dist = findDistributionForBucket($cf, $bucket); (2)
$thumbsDist = findDistributionForBucket($cf, $bucketThumbs);

$objects = getBucketObjects($s3, $bucket); (3)
$objectThumbs = getBucketObjects($s3, $bucketThumbs);

$thumbs = array(); (4)
```

```
foreach ($objectThumbs as $objectThumb)
{
  $key = (string) $objectThumb->Key;

  if ($thumbsDist != null) ❺
  {
    $thumbs[$key] = 'http://' . $thumbsDist->DomainName
        . "/" . $key;
  }
  else
  {
    $thumbs[$key] = $s3->get_object_url($bucketThumbs, $key);
  }
}

$fileList = array(); ❻
foreach ($objects as $object)
{
  $key = (string) $object->Key;

  if ($dist != null)
  {
    $url = 'http://' . $dist->DomainName . "/" . $key;
  }
  else
  {
    $url = $s3->get_object_url($bucket, $key);
  }

  $thumbURL = IsSet($thumbs[$key]) ? $thumbs[$key] : ''; ❼
  $fileList[] = array('thumb' => $thumbURL, 'url' => $url,
      'name' => $key, 'size' => number_format((int)$object->Size));
}

$output_title = "Chapter 3 Sample - List of S3 Objects in Bucket"
    . " '${bucket}'";
$output_message = "A simple HTML table displaying of all the objects"
    . "in the '${bucket}' bucket with thumbnails.";

include 'include/list_bucket_objects_thumbs.html.php';

exit(0);
?>
```

Let's take a look at this code:

1. First, we instantiate new `AmazonS3` and `AmazonCloudFront` objects.
2. Here we determine whether there's a CloudFront distribution for each of our buckets.
3. Once again we use our `getBucketObjects` function to retrieve all the objects from our two buckets.
4. We iterate through all the objects in our thumb bucket, and populate the `$thumbs` array using the object keys as array keys and storing their URLs.
5. If there's a CloudFront distribution for our thumbnails, we use its URL; otherwise, we use the standard S3 URL.
6. We do the same operation for all the objects in the specified bucket; this is stored in the `$fileList` array and will be used in the HTML output.
7. This is a new addition from the **list_bucket_objects_page.php** script to the `$fileList` array. The thumbnail URL is saved if there's a matching thumbnail, otherwise an empty string is stored.

Here's the HTML template that generates the output:

chapter_04/list_bucket_objects_thumbs.html.php *(excerpt)*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title><?php echo $output_title ?></title>
</head>
<body>
  <h1><?php echo $output_title ?></h1>
  <p><?php echo $output_message ?></p>
  <table>
    <thead>
      <tr><th>File</th><th>Size</th></tr>
    </thead>
    <tbody>
    <?php foreach($fileList as $file): ?> (1)
      <tr>
        <td>
          <?php if($file['thumb'] != ''): ?> (2)
          <a href="<?php echo $file['url'] ?>">
              <img src="<?php echo $file['thumb'] ?>"/></a>
          <?php endif ?>
        </td>
        <td><a href="<?php echo $file['url'] ?>">
          <?php echo $file['name'] ?></a>
        </td>
        <td><?php echo $file['size'] ?></td>
      </tr>
    <?php endforeach ?>
    </tbody>
  </table>
</body>
</html>
```

There's a couple of interesting points in the code above:

(1) This template iterates over the `$fileList` array, adding a table row for each element of the array, and a table cell for the thumbnail image, the filename, and file size.

(2) The contents of the thumbnail table cell are only added if there's a thumbnail URL specified for that file, otherwise the cell is left empty.

Figure 4.4 shows what the output looks like.

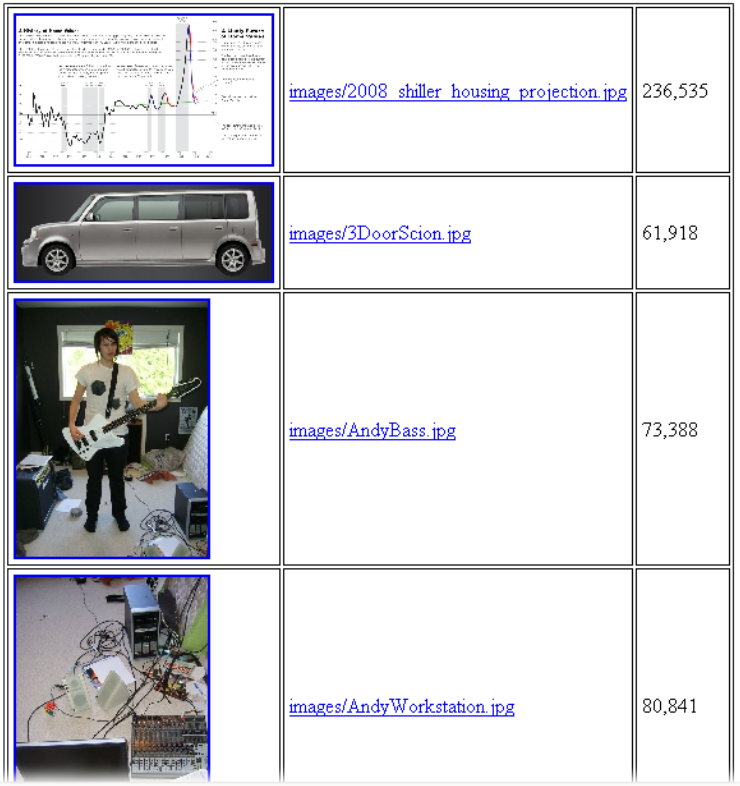

Figure 4.4. Object listing with thumbnails

Finally

This has been a long and code-heavy chapter. Starting from scratch you've learned how to create S3 buckets, list your buckets, and upload files to S3. You have discovered how to process the contents of one bucket into another, and how to use CloudFront for efficient content distribution. With all these new skills in hand, you should be ready to create some intriguing S3 applications of your very own.

Take the Next Step with Us

Stop wasting time, money, and resources on servers that can't grow with you.

Let Jeff Barr–Amazon's own "cloud guy"–show you how cloud computing can give you the ultimate freedom, scalability and speed, for probably less than you're paying right now.

So, why order the full book?

- Understand the Amazon Web Services universe
- Come to master visual and command-line tools
- Manage monitoring, load balancing, and scaling
- Four great formats (print, EPUB, MOBI, PDF)
- 100% money back guarantee

ORDER EBOOK
iPhone + iPad + Kindle + PDF

ORDER PRINTED BOOK

100% Satisfaction Guarantee

We want you to feel as confident as we do that this book will deliver the goods, so you have a full 30 days to play with it. If in that time you feel the book falls short, simply send it back and we'll give you a prompt refund of the full purchase price, minus shipping and handling.

ALL SOURCE CODE AVAILABLE FOR DOWNLOAD

EXPERIENCE THE POWER AND FREEDOM OF THE NEW GENERATION OF WEB SERVICES!

With Amazon Web Services you can launch new servers in minutes, distribute data through a global high-speed network, and store terabytes of data at a fraction of the normal cost.

Host Your Web Site in the Cloud is your step-by-step guide to this revolutionary approach to hosting and managing your web applications.

Authored by Amazon Web Services Senior Evangelist, Jeff Barr, this book's straightforward, practical approach and easy-to-read style will have you running your web site in the cloud in no time.

Grow your web site's capacity to meet demand and never worry about hardware again!

SO, WHAT IS AWS?

Amazon Web Services is an exciting web services platform that offers easy, programmable access to scalable, reliable computing resources on a pay-as-you-go basis. Taking advantage of Amazon's global computing infrastructure, AWS provides your web application with on-demand processor power, storage, and an extensive suite of services as you need them.

amazon web services™

JEFF BARR
Senior Evangelist, Amazon Web Services

Jeff Barr joined Amazon in 2002 when he realized it was destined to become the next great developer platform, and that he could help to make it so. In his role as the Amazon Web Services Senior Evangelist, Jeff speaks to developers at conferences and user groups all over the world. Before coming to Amazon, Jeff ran his own consulting practice and has held management and development positions at Microsoft, eByz, KnowNow, and Visix Software.

SITEPOINT BOOKS

- ✓ Advocate **best practice** techniques
- ✓ Lead you through **practical** examples
- ✓ Provide **working code** for your web site
- ✓ Make learning **easy** and **fun**

WEB DEVELOPMENT
ISBN-13:978-0-9805768-3-2

USD $39.95 CAD $49.95

Visit us on the Web at **sitepoint.com** or for sales and support email **books@sitepoint.com**