Making Everything Easier!™

Amazon Web Services™ For Dummies®

A Wiley Brand

Learn to:

- Get acquainted with the AWS® toolset
- Implement a website that utilizes AWS
- Manage AWS usage and costs
- Use Amazon Glacier™ to store large quantities of data

Bernard Golden

Author of Virtualization For Dummies

Chapter 1

Amazon Web Services Philosophy and Design

In This Chapter

- Figuring out the cloud
- Watching Amazon grow from retailer to the world's first cloud provider
- Understanding the foundation of Amazon Web Services
- Introducing the Amazon Web Services ecosystem
- Seeing how the network effect helps you
- Comparing Amazon Web Services to other cloud computing providers

You may be forgiven if you're puzzled about how Amazon, which started out as an online bookstore, has become *the* leading cloud computing provider. This chapter solves that mystery by discussing the circumstances that led Amazon into the cloud computing services arena and why Amazon Web Services, far from being an oddly different offering from a retailer, is a logical outgrowth of Amazon's business.

This chapter also compares Amazon's cloud offering to other competitors in the market and explains how its approach differs. As part of this comparison, I present some statistics on the size and growth of Amazon's offering, while describing why it's difficult to get a handle on its exact size.

The chapter concludes with a brief discussion about the Amazon Web Services ecosystem and why it is far richer than what Amazon itself provides — and why it offers more value for users of Amazon's cloud service.

But before I reveal all the answers to the Amazon mystery, I answer an even more fundamental question: What *is* all this cloud computing stuff, anyway?

Cloud Computing Defined

I believe that skill is built on a foundation of knowledge. Anyone who wants to work with Amazon Web Services (AWS, from now on) should have a firm understanding of cloud computing — what it is and what it provides.

IaaS, Paas, SaaS

As a general overview, cloud computing refers to the delivery of computing services from a remote location over a network. The National Institute of Standards and Technology (NIST), a U.S. government agency, has a definition of cloud computing that is generally considered the gold standard. Rather than trying to create my own definition, I always defer to NIST's definition. The following information is drawn directly from it.

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

This cloud model is composed of five essential characteristics:

- **On-demand self-service:** A consumer can unilaterally provision computing capabilities, such as server time and network storage, automatically as needed without requiring human interaction with each service provider.
- **Broad network access:** Capabilities are available over the network and accessed via standard mechanisms that promote use by heterogeneous thin or thick client platforms (such as mobile phones, tablets, laptops, and workstations).
- **Resource pooling:** The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There's a sense of so-called *location independence,* in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (by country, state, or data center, for example). Examples of resources are storage, processing, memory, and network bandwidth.
- **Rapid elasticity:** Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

- **Measured service:** Cloud systems automatically control and optimize resource use by leveraging a metering capability at a level of abstraction that's appropriate to the type of service (storage, processing, bandwidth, or active user accounts, for example). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

Cloud computing is commonly characterized as providing three types of functionality, referred to *IaaS, PaaS,* and *SaaS,* where *aaS* is shorthand for "as a service" and *service* implies that the functionality isn't local to the user but rather originates elsewhere (a location in a remote location accessed via a network). The letters *I, P,* and *S* in the acronyms refer to different types of functionality, as the following list makes clear:

- **Infrastructure as a Service (Iaas):** Offers users the basic building blocks of computing: processing, network connectivity, and storage. (Of course, you also need other capabilities in order to fully support IaaS functionality — such as user accounts, usage tracking, and security.) You would use an IaaS cloud provider if you want to build an application from scratch and need access to fairly low-level functionality within the operating system.
- **Platform as a Service (PaaS):** Instead of offering low-level functions within the operating system, offers higher-level programming frameworks that a developer interacts with to obtain computing services. For example, rather than open a file and write a collection of bits to it, in a PaaS environment the developer simply calls a function and then provides the function with the collection of bits. The PaaS framework then handles the grunt work, such as opening a file, writing the bits to it, and ensuring that the bits have been successfully received by the file system. The PaaS framework provider takes care of backing up the data and managing the collection of backups, for example, thus relieving the user of having to complete further burdensome administrative tasks.
- **Software as a Service (SaaS):** Has clambered to an even higher rung on the evolutionary ladder than PaaS. With SaaS, all application functionality is delivered over a network in a pretty package. The user need do nothing more than use the application; the SaaS provider deals with the hassle associated with creating and operating the application, segregating user data, providing security for each user as well as the overall SaaS environment, and handling a myriad of other details.

REMEMBER

As with every model, this division into *I, P,* and *S* provides a certain explanatory leverage and seeks to make neat and clean an element that in real life can be rather complicated. In the case of IPS, the model is presented as though the types are cleanly defined though they no longer are. Many cloud providers offer services of more than one type. Amazon, in particular, has begun to provide many platform-like services as it has built out its offerings, and has even ventured into a few full-blown application services that you'd associate with SaaS. You could say that Amazon provides all three types of cloud computing.

Private-versus-public cloud computing

If you find the mix of *I, P,* and *S* in the preceding section confusing, wait 'til you hear about the whole private-versus-public cloud computing distinction. Note the sequence of events:

1. Amazon, as the first cloud computing provider, offers *public cloud computing* — anyone can use it.
2. Many IT organizations, when contemplating this new Amazon Web Services creature, asked why they couldn't create and offer a service like AWS to their own users, hosted in their own data centers. This on-premise version became known as *private cloud computing.*
3. Continuing the trend, several hosting providers thought they could offer their IT customers a segregated part of their data centers and let customers build clouds there. This concept can also be considered private cloud computing because it's dedicated to one user. On the other hand, because the data to and from this private cloud runs over a shared network, is the cloud truly private?
4. Finally, after one bright bulb noted that companies may not choose only public or private, the term *hybrid* was coined to refer to companies using both private and public cloud environments.

As you go further on your journey in the cloud, you'll likely witness vociferous discussions devoted to which of these particular cloud environments is the better option. My own position is that no matter where you stand on the private/public/hybrid issue, public cloud computing will undoubtedly become a significant part of every company's IT environment. Moreover, Amazon will almost certainly be the largest provider of public cloud computing, so it makes sense to plan for a future that includes AWS. (Reading this book is part of that planning effort, so you get a gold star for already being well on your way!)

TIP

If you want to drill down further into cloud computing definitions, check out NIST's full description at `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`. The U.S. federal government has been an early adopter of, and hard charger in, cloud adoption, and NIST has been assigned to create this (excellent) government-wide cloud computing resource.

Understanding the Amazon Business Philosophy

Amazon Web Services was officially revealed to the world on March 13, 2006. On that day, AWS offered the Simple Storage Service, its first service. (As you may imagine, Simple Storage Services was soon shortened to S3.) The idea behind S3 was simple: It could offer the concept of object storage over the web, a setup where anyone could put an object — essentially, any bunch of bytes — into S3. Those bytes may comprise a digital photo or a file backup or a software package or a video or audio recording or a spreadsheet file or — well, you get the idea.

S3 was relatively limited when it first started out. Though objects could, admittedly, be written or read from anywhere, they could be stored in only one region: the United States. Moreover, objects could be no larger than 5 gigabytes — not tiny by any means, but certainly smaller than many files that people may want to store in S3. The actions available for objects were also quite limited: You could write, read, and delete them, and that was it.

In its first six years, S3 has grown in all dimensions. The service is now offered throughout the world in a number of different regions. Objects can now be as large as 5 terabytes. S3 can also offer many more capabilities regarding objects. An object can now have a termination date, for example: You can set a date and time after which an object is no longer available for access. (This capability may be useful if you want to make a video available for viewing for only a certain period, such as the next two weeks.) S3 can now also be used to host websites — in other words, individual pages can be stored as objects, and your domain name (say, `www.example.com`) can point to S3, which serves up the pages.

S3 did not remain the lone AWS example for long. Just a few months after it was launched, Amazon began offering Simple Queue Service (SQS), which provides a way to pass messages between different programs. SQS can accept or deliver messages within the AWS environment or outside the environment to other programs (your web browser, for example) and can be used to build highly scalable distributed applications.

Later in 2006 came Elastic Compute Cloud (known affectionately as EC2). As the AWS computing service, EC2 offers computing capacity on demand, with immediate availability and no set commitment to length of use.

REMEMBER

Don’t worry if this description of AWS seems overwhelming at first — in the rest of this book, you can find out all about the various pieces of AWS, how they work, and how you can use them to address your computing requirements. This chapter provides a framework in which to understand the genesis of AWS, with details to follow. The important thing for you to understand is how AWS got started, how big of a change it represents in the way computing is done, and why it’s important to your future.

The overall pattern of AWS has been to add additional services steadily, and then quickly improve each service over time. AWS is now composed of more than 25 different services, many offered with different capabilities via different configurations or formats. This rich set of services can be mixed and matched to create interesting and unique applications, limited only by your imagination or needs.

So, from one simple service (S3) to more than 25 in just over six years, and throughout the world — and growing and improving all the time! You're probably impressed by how fast all of this has happened. You're not alone. Within the industry, Amazon is regarded with a mixture of awe and envy because of how rapidly it delivers new AWS functionality. If you're interested, you can keep up with changes to AWS via its What's New web page on the AWS site, at

```
http://aws.amazon.com/about-aws/whats-new
```

This torrid pace of improvement is great news for you because it means that AWS continually presents new things you can do — things you probably couldn't do in the past because the AWS functionality would be too difficult to implement or too expensive to afford even if you could implement it.

Measuring the scale of AWS

Amazon is the pioneer of cloud computing and, because you'd have to have been living under a rock not to have heard about "the cloud," being the pioneer in this area is a big deal. The obvious question is this: If AWS is the big dog in the market and if cloud computing is the hottest thing since sliced bread, how big are we talking about?

That's an interesting question because Amazon reveals little about the extent of its business. Rather than break out AWS revenues, the company lumps them into an Other category in its financial reports.

Nevertheless, we have some clues to its size, based on information from the company itself and on informed speculation by industry pundits.

Amazon itself provides a proxy for the growth of the AWS service. Every so often, it announces how many objects are stored in the S3 service. Take a peek at Figure 1-1, which shows how the number of objects stored in S3 has increased at an enormous pace, jumping from 2.9 billion at the end of 2006 to over 2 trillion objects by the end of the second quarter of 2012. Given that pace of growth, it's obvious that the business of AWS is booming.

Other estimates of the size of the AWS service exist as well. A very clever consultant named Huan Liu examined AWS IP addresses and projected the total number of server racks held by AWS, based on an estimate of how many servers reside in a rack. Table 1-1 breaks down the numbers by region.

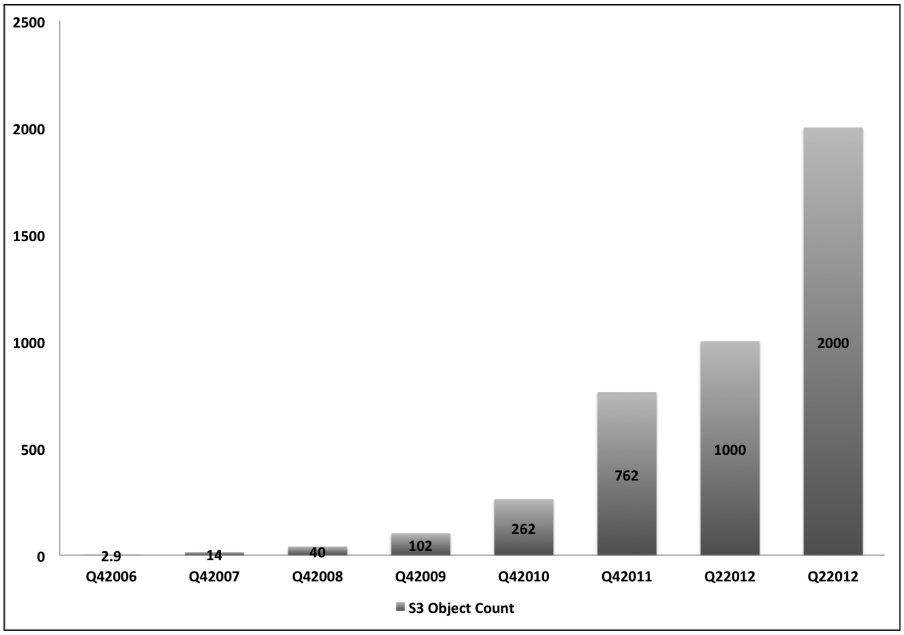

Figure 1-1: Counting S3 objects over the years.

Table 1-1 Total AWS Servers

AWS Region	*Number of Server Racks*	*Number of Servers*
US East	5,030	321,920
US West (Oregon)	41	2,624
US West (California)	630	40,320
EU West (Ireland)	814	52,096
AP Northeast (Japan)	314	20,096
AP Southeast (Singapore)	246	15,744
SA East (Brazil)	25	1,600
Total	**7,100**	**454,400**

That's a lot of servers. (To see the original document outlining Liu's estimates, along with his methodology, go to `http://huanliu.wordpress.com/2012/03/13/amazon-data-center-size`). If you consider that each server can support a number of virtual machines (the number would vary, of course, according to the size of the virtual machines), AWS could support several million running virtual machines.

Amazon publishes a list of public IP addresses; as of May 2013, there are over four million available in AWS. This number is not inconsistent with Liu's estimated number of physical servers; it's also a convenient place to look to track how much AWS is growing. If you're interested, you can look at the AWS numbers at `https://forums.aws.amazon.com/ann.jspa?annID=1701`.

If you're not familiar with the term virtual machines, don't worry: I describe AWS technology in depth in Chapter 4. For an even more detailed discussion of virtual machines and virtualization proper, check out *Virtualization For Dummies,* by yours truly (published by John Wiley & Sons, Inc.).

Checking the bottom line

Though Amazon doesn't announce how many dollars AWS pulls in, that hasn't stopped others from making their own estimates of the size of AWS business — and their estimates make it clear that AWS is a very large business indeed.

Early in 2012, several analysts from Morgan Stanley analyzed the AWS business and judged that the service pulled in $1.19 billion in 2011. (You gotta love the precision that these pundits come up with, eh?) Other analysts from JP Morgan Chase and UBS have calculated that AWS will achieve 2015 revenues of around $2.5 billion.

The bottom line: AWS is big and getting bigger (and better) every day. It really is no exaggeration to say that AWS represents a revolution in computing. People are doing amazing things with it, and this book shows you how you can take advantage of it.

The AWS Infrastructure

If what Amazon is doing with AWS represents a revolution, as I describe in the previous section, how is the company bringing it about? In other words, how is it delivering this amazing service? Throughout this book, I go into the specifics of how the service operates, but for now I outline the general approach that Amazon has taken in building AWS.

First and foremost, Amazon has approached the job in a unique fashion, befitting a company that changed the face of retail. Amazon specializes in a low-margin approach to business, and it carries that perspective into AWS. Unlike almost every other player in the cloud computing market, Amazon has focused on creating a low-margin, highly efficient offering, and that offering starts with the way Amazon has built out its infrastructure.

Making hard hardware decisions

Unlike most of its competitors, Amazon builds its hardware infrastructure from commodity components. *Commodity,* in this case, refers to using equipment from lesser-known manufacturers who charge less than their brand-name competitors. For components for which commodity offerings aren't available, Amazon (known as a ferocious negotiator) gets rock-bottom prices.

On the hardware side of the AWS offering, Amazon's approach is clear: Buy equipment as cheaply as possible. But wait, you may say, won't the commodity approach result in a less reliable infrastructure? After all, the brand-name hardware providers assert that one benefit of paying premium prices is that you get higher-quality gear. Well . . . yes and no. It may be true that premium-priced equipment (traditionally called enterprise equipment because of the assumption that large enterprises require more reliability and are willing to pay extra to obtain it) is more reliable in an apples-to-apples comparison. That is, an enterprise-grade server lasts longer and suffers fewer outages than its commodity-class counterpart.

The issue, from Amazon's perspective, is how much more reliable the enterprise gear is than the commodity version, and how much that improved reliability is worth. In other words, it needs to know the cost-benefit ratio of enterprise-versus-commodity.

Making this evaluation more challenging is a fundamental fact: At the scale on which an Amazon operates (remember that it has nearly half a million servers running in its AWS service), equipment — no matter *who* provides it — is breaking all the time.

If you're a cloud provider with an infrastructure the size of Amazon's, you have to assume, for every type of hardware you use, an endless round of crashed disk drives, fried motherboards, packet-dropping network switches, and on and on.

Therefore, even if you buy the highest-quality, most expensive gear available, you'll still end up (if you're fortunate enough to grow into a very large cloud computing provider like, say, Amazon) with an unreliable infrastructure. Put another way, at a very large scale, even highly reliable individual components still result in an unreliable overall infrastructure because of the failure of components, as rare as the failure of a specific piece of equipment may be.

REMEMBER

The scale at which Amazon operates affects other aspects of its hardware infrastructure as well. Besides components such as servers, networks, and storage, data centers also have power supplies, cooling, generators, and backup batteries. Depending on the specific component, Amazon may have to use custom-designed equipment to operate at the scale required.

Think of AWS hardware infrastructure this way: If you had to design and operate data centers to deal with massive scale and in a way that aligns with a corporate mandate to operate inexpensively, you'd probably end up with a solution much like Amazon's. You'd use commodity computing equipment whenever possible, jawbone prices down when you couldn't obtain commodity offerings, and custom-design equipment to manage your unusually large-scale operation.

TIP

For more detail on Amazon's data center approach, check out James Hamilton's blog at `http://perspectives.mvdirona.com`. (He's one of Amazon's premier data center architects.) The blog includes links to videos of his extremely interesting and educational presentations on how Amazon has approached its hardware environment.

Examining Amazon's software infrastructure strategy

Because of Amazon's low-margin, highly scaled requirements, you'd probably expect it to have a unique approach to the cloud computing software infrastructure running on top of its hardware environment, right?

You'd be correct.

Amazon has created a unique, highly specialized software environment in order to provide its cloud computing services. I stress the word unique because, at first glance, people often find AWS different and confusing — it is unlike any other computing environment they've previously encountered.

After users understand how AWS operates, however, they generally find that its design makes sense and that it's appropriate for what it delivers — and, more important, for how people use the service.

Though Amazon has an unusual approach to its hardware environment, it's in the software infrastructure that its uniqueness truly stands out. Let me give you a quick overview of its features. The software infrastructure is

- **Based on virtualization:** Virtualization — a technology that abstracts software components from dependence on their underlying hardware — lies at the heart of AWS. Being able to create virtual machines, start them, terminate them, and restart them quickly makes the AWS service possible.

  As you might expect, Amazon has approached virtualization in a unique fashion. Naturally, it wanted a low-cost way to use virtualization, so it chose the open source Xen Hypervisor as its software foundation. Then it made significant changes to the "vanilla" Xen product so that it could fulfill the requirements of AWS.

The result is that Amazon leverages virtualization, but the virtualization solution it came up with is extended in ways that support vast scale and a plethora of services built atop it.

- **Operated as a service:** I know what you're going to say: "Of course it's operated as a service — that's why it's called Amazon Web Services!"

  That's true, but Amazon had to create a tremendous software infrastructure in order to be able to offer its computing capability as a service.

  For example, Amazon had to create a way for users to operate their AWS resources from a distance and with no requirement for local hands-on interaction. And it had to segregate a user's resources from everyone else's resources in a way that ensures security, because no one wants other users to be able to see, access, or change his resources.

  Amazon had to provide a set of interfaces — an Application Programming Interface (API) — to allow users to manage every aspect of AWS. (I cover the AWS API in Chapter 5.)

- **Designed for flexibility:** Amazon designed AWS to address users like itself — users that need rich computing services available at a moment's notice to support their application needs and constantly changing business conditions.

  In other words, just as Amazon can't predict what its computing requirements will be in a year or two, neither can the market for which Amazon built AWS.

  In that situation, it makes sense to implement few constraints on the service. Consequently, rather than offer a tightly integrated set of services that provides only a few ways to use them, Amazon provides a highly granular set of services that can be "mixed and matched" by the user to create an application that meets its exact needs.

  By designing the service in a highly flexible fashion, Amazon enables its customers to be creative, thereby supporting innovation. Throughout the book, I'll offer examples of some of the interesting things companies are doing with AWS.

  Not only are the computing services themselves highly flexible, the conditions of use of AWS are flexible as well. You need nothing more to get started than an e-mail address and a credit card.

- **Highly resilient:** If you took the message from earlier in the chapter about the inherent unreliability of hardware to heart, you now recognize that there is no way to implement resiliency via hardware. The obvious alternative is with software, and that is the path Amazon has chosen.

  Amazon makes AWS highly resilient by implementing resource redundancy — essentially using multiple copies of a resource to ensure that failure of a single resource does not cause the service to fail. For example, if you were to store just one copy of each of your objects within its S3 service, that object may sometimes be unavailable because the disk drive on which it resides has broken down. Instead, AWS keeps multiple copies of an object, ensuring that even if one — or two! — objects become unavailable because of hardware failure, users can still access the object, thereby improving S3 reliability and durability.

In summary, Amazon has implemented a rich software infrastructure to allow users access to large quantities of computing resources at rock-bottom prices. And if you take another look at Figure 1-1 (the one outlining the number of objects stored in S3), you'd have to draw the conclusion that a large number of users are increasingly benefiting from AWS.

The AWS Ecosystem

Thus far, I haven't delved too deeply into the various pieces of the AWS puzzle, but it should be clear (if you're reading this chapter from start to finish) that Amazon offers a number of services to its users. However, AWS hosts a far richer set of services than only the ones it provides. In fact, users can find nearly everything they need within the confines of AWS to create almost any application they may want to implement. These services are available via the *AWS ecosystem* — the offerings of Amazon partners and third parties that host their offerings on AWS.

So, in addition to the 25+ services AWS itself offers, users can find services that

- Offer preconfigured virtual machines with software components already installed and configured, to enable quick use
- Manipulate images
- Transmit or stream video
- Integrate applications with one another
- Monitor application performance
- Ensure application security
- Operate billing and subscriptions
- Manage healthcare claims
- Offer real estate for sale
- Analyze genomic data

- Host websites
- Provide customer support

And really, this list barely scratches the surface of what's available within AWS. In a way, AWS is a modern-day bazaar, providing an incredibly rich set of computing capabilities *from* anyone who chooses to set up shop *to* anyone who chooses to purchase what's being offered.

On closer inspection, you can see that the AWS ecosystem is made up of three distinct subsystems:

- **AWS computing services provided by Amazon:** As noted earlier, Amazon currently provides more than 25 AWS services and is launching more all the time. AWS provides a large range of cloud computing services — you'll be introduced to many of them over the course of this book.
- **Computing services provided by third parties that operate on AWS:** These services tend to offer functionality that enables you to build applications of a type that AWS doesn't strictly offer. For example, AWS offers some billing capability to enable users to build applications and charge people to use them, but the AWS service doesn't support many billing use cases — user-specific discounts based on the size of the company, for example. Many companies (and even individuals) offer services complementary to AWS that then allow users to build richer applications more quickly. (If you carry out the AWS exercises I set out for you later in this book, you'll use one such service offered by Bitnami.)
- **Complete applications offered by third parties that run on AWS:** You can use these services, often referred to as SaaS (Software as a Service), over a network without having to install them on your own hardware. (Check out the "IaaS, Paas, SaaS" section, earlier in this chapter, for more on SaaS.) Many, many companies host their applications on AWS, drawn to it for the same reasons that end users are drawn to it: low cost, easy access, and high scalability. An interesting trend within AWS is the increasing move by traditional software vendors to migrate their applications to AWS and provide them as SaaS offerings rather than as applications that users install from a CD or DVD on their own machines.

WARNING!

As you go forward with using AWS, be careful to recognize the differences between these three offerings within the AWS ecosystem, especially Amazon's role (or lack thereof) in all three. Though third-party services or SaaS applications can be incredibly valuable to your computing efforts, Amazon, quite reasonably, offers no support or guarantee about their functionality or performance. It's up to you to decide whether a given non-AWS service is fit for your needs.

Amazon, always working to make it ever easier to locate and integrate third-party services into your application, has created the Amazon Marketplace as your go-to place for finding AWS-enabled applications. Moreover, being part of the Marketplace implies an endorsement by AWS, which will make you more confident about using a Marketplace application. You can read more about the Marketplace at

```
https://aws.amazon.com/marketplace
```

Counting Up the Network Effects Benefit

The reason the AWS ecosystem has become *the* computing marketplace for all and sundry can be captured in the phrase *network effect,* which can be thought of as the value derived from a network because other network participants are part of the network. The classic case of a network effect is the telephone: The more people who use telephones, the more value there is to someone getting a telephone — because the larger the number of telephones being used, the easier it is to communicate with a large number of people. Conversely, if you're the only person in town with a telephone, well, you're going to be pretty lonely — and not very talkative! Said another way, for a service with network effects, the more people who use it, the more attractive it is to potential users, and the more value they receive when they use the service.

From the AWS perspective, the network effect means that, if you're providing a new cloud-based service, it makes sense to offer it where lots of other cloud users are located — someplace like AWS, for example. This network effect benefits AWS greatly, simply because many people, when they start to think about doing something with cloud computing, naturally gravitate to AWS because it's a brand name that they recognize.

However, with respect to AWS, there's an even greater network effect than the fact that lots of people are using it: The technical aspects of AWS play a part as well.

When one service talks to another over the Internet, a certain amount of time passes when the communication between the services travels over the Internet network — even at the speed of light, information traveling long distances takes a certain amount of time. Also, while information is traveling across the Internet, it's constantly being shunted through routers to ensure that it's being sent in the right direction. This combination of network length and device interaction is called *latency,* a measure of how much of a delay is imposed by network traffic distance.

In concrete terms, if you use a web browser to access data from a website hosted within 50 miles of you, it will likely respond faster than if the same website were hosted 7,000 miles away.

To continue this concept, using a service that's located nearby makes your application run faster — always a good thing. So if your service runs on AWS, you'd like any services you depend on to also run on AWS — because the latency affecting your application is much lower than if those services originated somewhere else.

Folks who build services tend to be smart, so they'll notice that their potential customers like the idea of having services nearby. If you're setting up a new service, you'll be attracted to AWS because lots of other services are already located there. And if you're considering using a cloud service, you're likely to choose AWS because the number of services there will make it easier to build your application, from the perspective of service availability *and* low-latency performance.

The network effects associated with AWS give you a rich set of services to leverage as you create applications to run on Amazon's cloud offering. They can work to reduce your workload and speed your application development delivery by relieving you of much of the burden traditionally associated with integrating external software components and services into your application.

Here are some benefits of being able to leverage the network effects of the AWS ecosystem in your application:

- **The service is already up and running within AWS.** You don't have to obtain the software, install it, configure it, test it, and then integrate it into your application. Because it's already operational in the AWS environment, you can skip directly to the last step — perform the technical integration.
- **The services have a cloud-friendly licensing model.** Vendors have already figured out how to offer their software and charge for it in the AWS environment. Vendors often align with the AWS billing methodology, charging per hour of use or offering a subscription for monthly access. But one thing you don't have to do is approach a vendor that has a large, upfront license fee and negotiate to operate in the AWS environment — it's already taken care of.
- **Support is available for the service.** You don't have to figure out why a software component you want to use doesn't work properly in the AWS environment — the vendor takes responsibility for it. In the parlance of the world of support, you have, as the technology industry rather indelicately puts it, a throat to choke.

- **Performance improves.** Because the service operates in the same environment that your application runs in, it provides low latency and helps your application perform better.

TIP

Before you start thinking about finding a packaged software application to integrate into your application, or about writing your own software component to provide certain functionality, search the Marketplace to see whether one or more applications already provide the necessary functionality.

AWS versus Other Cloud Providers

Nature abhors a vacuum, and markets abhor monopoly providers, so it stands to reason that competitors always enter an attractive market. Cloud computing is no different: There are a plethora of cloud computing providers. Naturally, you'll want to get the lowdown on how AWS measures up.

The most important difference between AWS and almost all other cloud providers revolves around what market they target. To understand that aspect, you must understand the basis of the service they offer.

Now, AWS grew out of the capabilities that Amazon developed to enable its developers to rapidly create and deploy applications. The service is focused on making developers more productive and, in a word, happier.

By contrast, most other cloud providers have a hosting heritage: Their backgrounds involve supporting infrastructure for IT operations groups responsible for maintaining system uptime. A significant part of the value proposition for hosting providers has traditionally been the high quality of their infrastructures — in other words, the enterprise nature of their servers, networks, storage, and so on.

This heritage carries several implications about enterprise cloud providers:

- **The focus is on the concerns of IT operations rather than on the concerns of developers.** Often, this concern translates as, "The service is not easy to use." For example, an enterprise cloud provider may require a discussion with a sales representative before granting access to the service and then impose a back-and-forth manual process as part of the account setup. By contrast, AWS allows anyone with an e-mail address and a credit card access to the service within ten minutes.
- **The service itself reflects its hosting heritage, with its functionality and use model mirroring how physical servers operate.** Often, the only storage an enterprise cloud service provider offers is associated with

individual virtual machines — no object storage, such as S3, is offered, because it isn't part of a typical hosting environment.

- **Enterprise cloud service providers often require a multiyear commitment to resource use with a specific level of computing capacity.** Though this strategy makes it easier for a cloud service provider to plan its business, it's much less convenient for users — *and* it imposes some of the same issues that they're trying to escape from!
- **The use of enterprise equipment often means higher prices when compared to AWS.** I have seen enterprise cloud service providers charge 800 percent more than AWS. Depending on organization requirements and the nature of the application, users may be willing to pay a premium for these providers; on the other hand, higher prices and the long-term commitment that often accompanies the use of an AWS competitor may strike many users as unattractive or even unacceptable.

The rise of shadow IT

Frustration at being unable to get hold of server resources in a timely fashion has led to the phenomenon of *shadow IT:* developers bypassing IT proper and obtaining resources themselves. This phenomenon is powerful and growing — at one conference, I heard a CIO state that he had examined the expense reports submitted to him for reimbursement and found more than 50 different AWS accounts being used by his development staff!

Here's something to consider: Shadow IT is a pejorative term, implying stealth and a definite whiff of illicit behavior. On the other hand, someone engaging in shadow IT might, reasonably enough, think of it as "getting the job done" in the face of existing processes that can stretch out to several months the length of time required to obtain resources.

This conflict is unlikely to subside in the near future. Developers relish the freedom and flexibility that AWS provides, though many IT groups are engaged in a fruitless struggle to go back to "the good old days," where they set the rules.

The conflict will ultimately be resolved in favor of developers. The reason is simple: The application is the way businesses gain value from IT, and applications are often directly tied to revenue-generating offerings (say, a mobile phone app that enables users to order goods or services online). Infrastructure, the province of mainstream IT, is then seen as a necessary evil — the plumbing that supports applications.

The advantage held by developers can be seen in cloud computing market share. One technology analyst told me that, by his estimate, AWS represents 75 percent of the market for cloud service providers. I expect pressure to build on enterprise cloud providers to rapidly improve their offerings to include more developer-friendly services. Amazon's six-year head start may make it too elusive to overtake.

If you analyze how well Amazon matches up against the NIST definition of cloud computing (discussed at the beginning if this chapter) when compared with its competitors, AWS usually emerges victorious. In part, that's because AWS was the pioneer, and because the first entrant into a market typically gets to define it. There's more to it than that, though.

Amazon's stroke of genius was to put together an innovative offering addressing a market poorly served by traditional IT practices. Though hosting companies typically serve IT operations groups well, the emphasis on enterprise equipment and high uptime availability frustrates developers trying to get access to resources. Stories of waiting weeks or months for servers to be provisioned are rife within the industry. As you might imagine, developers (and the application managers and executives they work for) longed for a different way of doing things — and that's what AWS offers.

Getting Ready for the 21st Century

This chapter provides an overview of Amazon Web Services. It lets you see how AWS has grown from Amazon's own computing needs and infrastructure to now represent Amazon's response to this simple hypothesis: If *we* need a flexible, cost-effective, and highly scalable infrastructure, *a lot of other organizations* could probably use one as well."

From that initial insight, Amazon created the computing platform of the 21st century. Targeted at developers, and provided throughout the world, AWS is undergoing explosive growth as more and more people explore how it can enable them to solve problems that were unsolvable by the traditional methods of managing infrastructure.

I hope that you can't wait to jump in to exploring AWS. This book aims to provide you with knowledge you need so that you, too, can leverage the amazing AWS cloud computing offering.

Chapter 2

Introducing the AWS API

In This Chapter

- What APIs are and why they're important
- Taking a closer look at the AWS API
- Knowing when to use the AWS API
- Explaining AWS API security

The AWS environment acts as an integrated collection of hardware and software services designed to enable the easy, quick, and inexpensive use of computing resources. (Chapter 1 gives all the details, if you're curious.) Now, sitting atop this integrated collection is the AWS application programming interface (API, for short): In essence, an *API* represents a way to communicate with a computing resource. (I tell you more about this topic later in this chapter.) With respect to AWS, nothing gets done without using the AWS API. The AWS API is the sole way that external users interact with AWS resources, and there's literally no way to use AWS resources without the API being involved. In fact, if you access AWS through the AWS Management Console or the command line tools, you are actually using tools that make calls to the AWS API.

In this chapter, I start by offering an introduction to the world of APIs and an explanation of why they've become increasingly important in the world of computing. I then discuss the AWS API and outline how it's used. Along the way, I discuss other, third-party services that you may want to use and tell you how they interact with the AWS API. Finally, I describe the AWS API security model, which is critical to understanding how Amazon ensures that only the right people perform acceptable actions within the AWS environment.

APIs: Understanding the Basics

You may consider yourself the kind of person who'd never, ever have to use an API. You'd be wrong. APIs have been important, they are important now, and they'll become even more important. More likely than not, you've been using APIs for years without even knowing it.

With respect to Amazon, the API is the sole external interface to computing resources and services. Without API calls being made, nothing gets done.

API is short for application programming interface, as I mention elsewhere. A good way to describe an API is to say that it represents a way for one program to interact with another via a defined *interface* — in other words, a mechanism by which any other program that communicates with the program can be assured that it will fulfill its role. The idea is that if a calling program provides the right information within the correct syntax, the program with the API will respond in the requested manner.

Understanding APIs

The term *API* traditionally referred to the programming interface offered by one or more routines that were bundled into a library of functions. Someone would supply a library that, say, performs date-and-time manipulation functions. A software engineer would bundle that library into a program and could then call those functions via the API that the library offers.

The API represents the "contract" that the library offers. The API defines the functional interface, the format of any information supplied to the functions (commonly called *arguments* or *parameters)* within the library, the operation to be performed, and the output that each function would return to the calling program.

One benefit of this "contractual" approach is that it offers *encapsulation* within the library — the actual code that implements the contract is hidden from the calling function. The library code can then be modified, updated, or even replaced entirely with another set of code, all without disturbing the calling function — as long as the new library code fulfills the old contract. Encapsulation allows much more flexibility in software environments, because different parts of the overall environment evolve at different rates; changing one part of the environment doesn't require changing everything. As long as the contract is adhered to, every other part of the environment can remain undisturbed.

The meaning of the term *API* has been extended: Rather than be used solely to discuss libraries that are directly attached to other programs, it's now used to refer to software environments in which the different software programs run on different servers and communicate across a network. Furthermore, that network may be contained within a single data center or, quite commonly, extend across the Internet. This network-based API approach is often referred to as a *web services environment* — notice how Amazon's cloud computing offering is named Amazon Web Services? It's no accident.

One critical factor that web services require, but traditional API libraries don't, is the whole notion of security. If two programs are communicating across the Internet, the one calling the service must be able to provide information regarding who it is (its identity), and the called service must be able to validate that whoever is doing the calling is allowed (authorized) to access the service it has requested. I discuss Amazon's approach to its web services security later in this chapter.

Benefiting from Web Services

I've heard people say that we're now living in a web services world, which, on the face of it, seems like an odd thing to say — after all, you may be able to go about much of your life today just as you did a decade ago without giving a moment's thought to something called "web services." However, even though you may not notice the fact, your everyday life is surrounded by web services.

As more aspects of our lives move to the Internet — banking, shopping, paying our taxes, collaborating at work, our social lives — people naturally want to be able to combine two or more of them into a new creation. It's the technological equivalent of the musical mash-up that's all the rage these days — a combination of two elements to create a new one that reflects parts of both. An early example of this phenomenon was an Internet application that combined Google Maps with craigslist apartment listings to create a map identifying the location of every available apartment. All the application did was combine (mash up) two basic services, but from that union came an extremely useful result — a guide to apartments in a particular area, making the process of selecting some to view and getting driving directions to them much more efficient.

The huge growth of mobile computing — the brave new world of smartphones and tablets — has worked to fuel the growth of APIs as well as mash-up applications. The "app culture" of mobile computing is a natural place to combine services, especially those tied to location. The apartment map application I just described is even more useful when it can be accessed while you're out and about. Finished looking at one apartment? Pull up the app and let it show you where the next nearest apartment for rent is located.

The next great frontier for web services is the so-called "Internet of Things," a term that refers to computing devices used not by humans, but by each other — interacting to complete useful tasks (smart electric meters that communicate with power company billing systems, for example). Soon, however, you'll be surrounded by all manner of devices that constantly interact with cloud-based applications. How big will the Internet of Things become? One senior Cisco executive predicts that *1 trillion* devices will soon be interacting over the Internet.

As more proof (if more proof were needed) that today's world is a web services world, companies, government agencies, and nonprofit organizations are feverishly making their resources available as online services accessible via APIs. Engineers are combining online web services to create new applications that combine individual services and provide unique and useful capabilities.

This web services revolution that I describe makes possible a number of interesting benefits:

- **Innovation:** Just as musical mash-ups let people combine musical resources into new creations, so, too, do web services foster innovation. Though I may not be able to see the value in a combination of, say, vehicle gas mileage ratings, local gas prices, and state park reviews, someone else may conclude that an application allowing someone to enter the make and model of her automobile to find out which parks she can visit for less than $25 in gas costs would be just the ticket — and a whole lot of people may agree. (In fact, I may put that app on my to-do list!)
- **Niche market support:** In a non-web-services world, the only people who can develop applications are those working for organizations. Only they have access to the computing resources or data — so the only applications that are developed are ones that the company deems useful. However, once those resources and data are made available via web services, anyone can create an application, which allows the development of applications targeted at niche markets. For example, someone can combine Google Maps with a municipal bus schedule in a mobile app to allow users to see when and where the next bus will be available nearby.
- **New sources of revenue:** Companies can provide a web services interface into their business transaction systems and allow outside entities to sell their goods. For example, the large retailer Sears has made it possible for mobile app developers and bloggers to sell Sears goods via a Sears web service. These developers and bloggers reach audiences that Sears may not be able to reach — but Sears can prosper without having to be involved. As another example, Netflix has made its web services interface to its video offerings available, and many device and game manufacturers have used the interface to incorporate Netflix into their products. Netflix can gain new revenues every time someone buys a Wii or an Xbox and decides that it would be cool to use his new toy to access online movies and television.

These examples should give you an understanding of why many people (and I include myself in that number) regard the rise of web services as a true technology revolution.

This revolution is nowhere near the end, either. Mobile computing is still growing extremely rapidly, and it's just getting going in emerging economies. The journey of the Internet of Things has barely begun. But I hope that you recognize just how important web services are to you — even if you're unaware of their presence.

The myth of excess capacity

This is probably as good a place as any to address a persistent myth about AWS. Many people think that the AWS service represents Amazon's effort to rent its excess computing capacity during periods of low demand throughout the year. They believe that because Amazon has to have a lot of capacity on hand to address the heavy load of retail shopping during the Christmas season, it created AWS to encourage people to use its infrastructure the rest of the year.

There's only one problem with that theory — many AWS customers also have Christmas seasonality, so if Amazon is selling its excess capacity the rest of the year, what does it do in December when those AWS customers want access to computing capacity and Amazon doesn't have much excess capacity available?

The truth is that AWS doesn't rely on excess Amazon retail capacity. AWS has its own capacity, thank you very much — and it's a good thing, given that Amazon retail's entire web server functionality now runs on AWS. If AWS depended on excess Amazon retail computing capacity but Amazon retail now runs on AWS . . . well, trying to understand that scenario is like trying to figure out an Escher drawing!

The bottom line: AWS has its own computing capacity, and it is growing dramatically. Early in 2010, I heard Werner Vogels, Amazon's chief technology officer (CTO), say that, because of customer demand, AWS installs, each and every day, as much computing capacity as all of Amazon ran on in 2000, when it was a $2.7 billion company.

Vogels updated his statement in late 2012, saying that, again because of customer demand, AWS now installs as much computing capacity daily as all of Amazon ran on in the year 2003, when it was a $5.2 billion company.

So the next time you hear someone say that AWS is primarily a way for Amazon to rent its off-season excess capacity, you'll know better, and you can @— if you dare! — set that person straight. AWS is its own, significant business, it has its own, significant computing infrastructure, and it is experiencing, by all evidence, very rapid expansion.

An Overview of the AWS API

I've said it before and I'll say it again: The only way you can interact with AWS is via its API. Every service you can ever use is called (and returns data) via its API, so using the API is critical to working with AWS. However, don't worry about having to know the details of a low-level programming interface — you'll likely never have to interact directly with the API. Nevertheless, you *must* understand at least the broad outline of how the AWS API functions, so that's what I describe in this section.

Web services: SOAP or REST?

You can choose from a couple of different schools of thought for how web services should be delivered. The older approach, SOAP (short for Simple Object Access Protocol), had widespread industry support, complete with a comprehensive set of standards. Those standards were too comprehensive, unfortunately. The people designing SOAP set it up to be extremely flexible — it can communicate across the web, e-mail, and private networks. To ensure security and manageability, a number of supporting standards that integrate with SOAP were also defined.

SOAP is based on a document encoding standard known as Extensible Markup Language (XML, for short), and the SOAP service is defined in such a way that users can then leverage XML no matter what the underlying communication network is. For this system to work, though, the data transferred by SOAP (commonly referred to as the *payload*) also needs to be in XML format. Notice a pattern here? The push to be comprehensive and flexible (or, to be all things to all people) plus the XML payload requirement meant that SOAP ended up being quite complex, making it a lot of work to use properly. As you might guess, many IT people found SOAP daunting and, consequently, resisted using it.

About a decade ago, a doctoral student defined another web services approach as part of his thesis: REST, or Representational State Transfer. (Frankly, I think he coined the term REST first, because it sounded easier and more relaxing than SOAP, and then configured the name to fit the acronym.)

REST, which is far less comprehensive than SOAP, aspires to solve fewer problems. It doesn't address some aspects of SOAP that seemed important but that, in retrospect, made it more complex to use — security, for example.

The most important aspect of REST is that it's designed to integrate with standard web protocols so that REST services can be called with standard web verbs and URLs. For example, a valid REST call looks like this:

```
http://search.examplecompany.com/CompanyDirectory/Employee
            Info?empname=BernardGolden
```

That's all it takes to make a query to the REST service of `examplecompany` to see my personnel information. The HTTP verb that accompanies this request is GET, asking for information to be returned. To delete information, you use the verb DELETE. To insert my information, you use the verb POST. To update my information, you use the verb PUT.

For the POST and PUT actions, additional information would accompany the `empname` and be separated by an ampersand (`&`) to indicate another argument to be used by the service.

REST imposes no particular formatting requirements on the service payloads; in this respect, it differs from SOAP, which requires XML. For simple interactions, a string of bytes is all you need for the payload; for more complex interactions (say, in addition to returning my employee information, I want to place a request for the employee information of all employees whose names start with G), the encoding convention JSON is used. (JSON, if you're curious, stands for Javascript Object Notation.)

As you might expect, REST's simpler use model, its alignment with standard web protocols and verbs, and its less restrictive payload formatting made it catch on with developers like a house on fire.

WARNING!

AWS originally launched with SOAP support for interactions with its API, but it has steadily *deprecated* (reduced its support for, in other words) its SOAP interface in favor of REST. My recommendation for any use of the AWS API is that you focus on using REST. That way, you won't end up with programs that someday stop working — long after you've forgotten the details of the interaction mechanisms. Unfortunately, I've experienced — many times — the unpleasant task of having to go back into a system and attempt to reconstruct my actions from months or years earlier. There's no sense in tempting fate with AWS — if you want to interact with the AWS API, use REST, which is Amazon's long-term direction.

The AWS API

As you might imagine, given the comprehensiveness of AWS services and the way Amazon has been improving and extending them, the AWS API is one large puppy — the AWS S3 API reference manual is 269 pages. (Think that's a lot of pages? The AWS EC2 API reference manual is *561 pages*.)

However, if you take a quick look at the following example of an API call, you'll quickly see that it closely resembles the (quite simple) REST example in the preceding section:

```
https://ec2.amazonaws.com/?Action=RunInstances
&ImageId=ami-60a54009
&MaxCount=3
&MinCount=1
&Placement.AvailabilityZone=us-east-1b
&Monitoring.Enabled=true
&AUTHPARAMS
```

The call, which is straightforward, instructs AWS to run between one and three instances based on an Amazon machine image of `ami-60a54009` and to place them in the `us-east-1b` availability zone. AWS provides monitoring capabilities, and this call instructs AWS to enable this monitoring. The `AUTHPARAMS` part is a stand-in for the information that AWS uses to implement security in its API. I discuss this topic later in this chapter, so just accept for now that this call has the appropriate security mechanisms in place to ensure its execution. (For more on instances and availability zones, check out Chapter 4.)

The AWS API in real-world use

That's it. That's all there is to the AWS API. Easy, right? Well, even though the concept is easily understood, in practice it can be extremely challenging to use the AWS API properly — as you would expect, given the hundreds of pages devoted to the reference guide.

At this point, you might not feel confident about your ability to successfully use AWS. Understandably, you might feel that interacting with AWS is too complicated and difficult for even the old college try.

Never fear. Though the down-and-dirty details of using the AWS API are quite challenging, they're unlikely to become stumbling blocks to achieving success with AWS.

That's because many clever people have recognized that the API is difficult to use and have created tools to make AWS simpler to use. In Figure 2-1, you can see the four major categories of AWS interaction mechanisms that spare you from the burden of interacting with the AWS API directly.

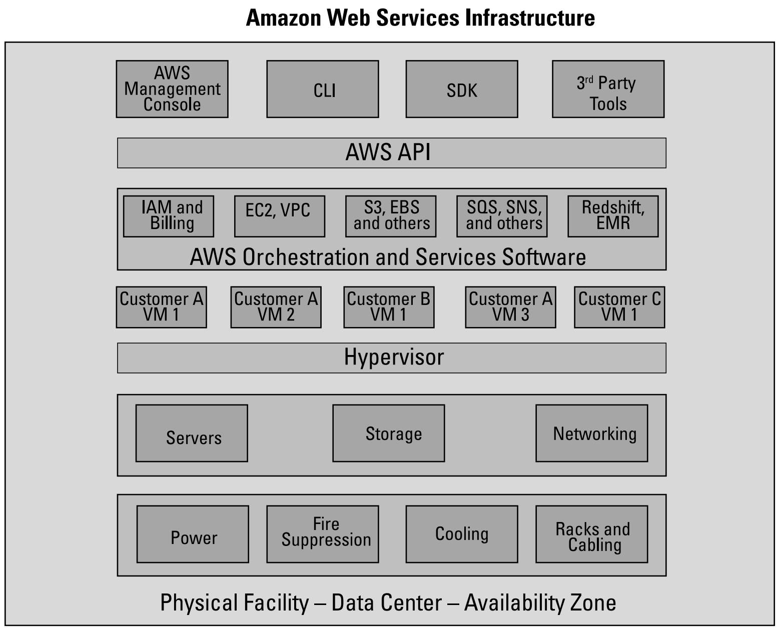

Figure 2-1: The AWS interface tools.
]

- **AWS management console:** Amazon offers a graphical web interface that allows you to interact with service (and your own) computing resources. For many people, the AWS management console is the primary mechanism they use to operate AWS. Even people who use the other two mechanisms to interact with AWS also make heavy use of the management console. I use the management console to explain the examples in this book, and I even focus a complete chapter on it to help you get started with AWS.
- **CLI/SDK:** Many software engineers write applications that need to interact with AWS services directly. Now, calling the web services API directly is complicated and error-prone. *Plumbing* is a common way to refer to this sort of underlying functionality, like the AWS API — just as most of us wouldn't want to have to install a whole new set of pipes only to fill a teapot, most software engineers would prefer not to have to deal with the details of the AWS API. To help them, Amazon and other companies have created language libraries (commonly called SDKs, standing for *S*oftware *D*evelopment *K*its) and a command line interface (commonly called a CLI), which allows commands to be entered in a terminal connected to AWS. The idea here is to offer a simpler programmatic interface to the set of functions that do the heavy lifting of interacting with the AWS API. A software engineer can more easily incorporate library routines into an application, making it easier and faster to build AWS-based applications.
- **Third-party tools:** Many companies build tools that incorporate AWS. Some of these tools extend or simplify AWS itself, similar to what the language libraries do for software engineers. Other tools are products that offer separate functionality or even entire applications. For example, my company provides cloud management software that offers additional functionality not offered by the AWS management console. Other examples include programming environments from companies like Heroku and Engine Yard, data warehousing technology from Informatica and JasperSoft, and load-testing services from SOASTA. What these tools have in common is that they provide functionality to shield users from interacting with the AWS API, making AWS easier and faster to use.

All four of these AWS API interaction mechanisms act as *proxies* on your behalf — under the covers they make the necessary calls to the AWS API to use the AWS functionality for actions you want to perform.

Netflix runs on AWS

If you live in the United States (or, increasingly, in many places throughout the world), there's a good chance that you subscribe to the online video streaming service known as Netflix. If you're a Netflix subscriber, you also use AWS — because the Netflix service runs on AWS.

Yes, that's right: Every time you log on to Netflix, browse its selections, read your personalized recommendations, view your queue, or select a video to watch, it's all running on AWS. And Netflix uses AWS for much more than these functions. All the *transcoding* that Netflix must perform (the process of converting one digital format to another, or several others, because Netflix must create separate versions for different mobile phones, tablets, TVs, and gaming devices) is done using AWS.

All use of AWS by Netflix is via the AWS API. Netflix runs tens of thousands of AWS EC2 instances, and trying to track and manage that number of resources via the AWS Management Console would be unworkable. So Netflix created its own AWS management tools to manage any of its applications running in AWS. Netflix even offers the tools under open source licenses so that other AWS users can take advantage of its work. You can learn more about Netflix and its open source efforts at `www.slideshare.net/adrianco/netflix-and-open-source`.

How broadly does Netflix use AWS? Well, consider this: Netflix has over 29 million subscribers. At peak viewing times (evenings, when hard-working people generally "kick back" by watching videos), Netflix accounts for a staggering *30 percent* of total Internet traffic. Of all Internet-connected TVs, 40 percent are used to access Netflix shows. And Netflix is growing — fast.

In fact, this level of growth is the reason that Netflix has chosen to rely on AWS as its infrastructure environment. Netflix realizes that it doesn't excel at building and operating data centers (its core business is media and video streaming, in case you didn't notice), so offloading the infrastructure work to AWS allows Netflix to focus on its core business.

Many people find this decision strange. Doesn't the Amazon video streaming service compete with Netflix? Yes, but Netflix has concluded that AWS is now *the* most advanced cloud provider and that Netflix support by AWS won't be tainted by the parent company's own video offering. If another cloud provider comes along and offers the scale and functionality of AWS, Netflix might reevaluate its commitment to AWS, but for now, if you use Netflix, you use AWS.

AWS API Security

Here's an obvious question when dealing with third-party proxies: If these tools act on your behalf, how does AWS know that the person on whose behalf they're acting is in fact *you?* In other words, how can AWS authenticate your identity to ensure that the commands it receives are from you? In fact, the same question is valid even if you interact with the AWS API directly. How can AWS validate your identity to ensure that it executes commands only for you?

One way, of course, is for you to include your account username and password in the API calls. Though some cloud providers take this approach, Amazon does not. Rather than rely on a username and password, it relies on two other identifiers to authenticate its API service calls: the access key and the secret access key. It uses these keys in service calls to implement security in a way that's much more secure than using only your username and password.

So how does it work? When you sign up for an account with AWS, you have the opportunity to create an access key and to have a secret access key sent to you. Each one is a lengthy string of random characters, and the secret access key is the longer of the two. When you download the secret access key, you should store it somewhere very secure because it is the key (sorry — bad pun) to implementing secure service calls. After you do this, both you and Amazon have a copy of the access key and the secret access key. Retaining a copy of the secret access key is crucial because it's used to encrypt information sent back and forth between you and AWS, and if you don't have the Secret Access Key, you can't execute any service calls on AWS.

The way the two keys are used is conceptually simple, although somewhat challenging in detail.

Essentially, for every service call you want carried out, you (or a tool operating on your behalf) do the following:

1. Create the service call payload.

   This is the data you need to send to AWS. It may be an object you want to store in S3 or the image identifier of an image you want to launch. (You'll also attach other pieces of information to the payload, but because they vary according to the specifics of the service call, I don't list them here. One piece of data is the current time.)

2. Encrypt the payload using the secret access key.

   Doing so ensures that no one can examine the payload and discover what's in it.

3. Digitally sign the encrypted payload by adding the secret access key to the encrypted payload and performing a digital signature process using the secret access key.

   Secret access keys are longer and more random than typical user passwords; the lengthy secret access key makes the encryption performed with it more secure than it would be if it were performed with a typical user password.

4. Send the total encrypted payload, along with your access key, to AWS via a service call.

   Amazon uses the access key to look up your secret access key, which it uses to decrypt the payload. If the decrypted payload represents readable text that can be executed, AWS executes the service call. Otherwise, it concludes that something is wrong with the service call (perhaps that it was called by a malevolent actor) and doesn't execute the service call.

In addition to the encryption just described, AWS has two other methods it uses to ensure the legitimacy of the service call:

- The first is based on the date information included with the service call payload, which it uses to determine whether the time associated with the making of the service call is appropriate; if the date in the service call is much different from what it should be (much earlier or later than the current time, in other words), AWS concludes that it isn't a legitimate service call and discards it.
- The second additional security measure involves a checksum you calculate for the payload. (A *checksum* is a number that represents the content of a message.) AWS computes a checksum for the payload; if its checksum doesn't agree with yours, it disallows the service call and doesn't execute it. This checksum approach ensures that no one tampers with the contents of a message and prevents a malevolent actor from intercepting a legitimate service call and changing it to perform an unacceptable action. If someone tampers with the message, when AWS calculates a checksum, that checksum no longer matches the one included in the message, and AWS refuses to execute the service call.

If, like most AWS users, you use a proxy method to interact with AWS — the AWS management console, a language library, or a third-party tool — you need to provide your access key and secret access key to the proxy. When the proxy executes AWS service calls on your behalf, it includes the access key in the call and use the secret access key to perform payload encryption.

Because of the critical role that these keys fulfill in AWS, you should share them *only* with entities you trust. If you want to try out a new third-party tool and you don't know much about the company, set up an AWS test account for the trial instead of using your production AWS account credentials. That way, if you decide not to go forward with the tool, you can drop it, terminate the test AWS account, and move forward, unconcerned about potential security vulnerabilities in your main production accounts. Of course, you can always create new access keys and secret access keys, but using your production keys for tests and then changing the keys creates a lot of work, because you need to update every place that makes reference to your existing keys. If you're like many other AWS users, you'll use a number of tools and libraries, and going back to them to update your keys is a pain. You're better off using nonproduction accounts to test new tools.

TECHNICAL STUFF

AWS offers a service to make managing keys/secret access keys easier. It's called IAM and is covered in Chapter 8. IAM allows you to assign keys and secret access keys to individuals or to applications, making it much easier to avoid wholesale changes when one person leaves an organization; it's also a great help when you need to give each application access to the AWS services and resources that it needs.

Contents at a Glance

Table of Contents

Enterprise Applications/General

Go to the cloud to build dynamic websites, take payments, store objects, and do so much more

Here's your ticket to the cloud… Amazon cloud, that is! Let this book serve as your tour guide as you discover all of the amazing things you can do with Amazon Web Services. From building dynamic websites and creating a blog to launching an e-commerce store and developing rich applications, you'll never want to come back down.

- ***Decisions, decisions* — from S3 to EBS, determine which AWS storage service to use**
- ***Take a big stretch* — find out how to use Amazon Elastic Compute Cloud, including choosing images, selecting deployment options, and designing new EC2 images**
- ***Lock it up* — put the best security measures in place as you explore the Cloud Computing Trust Boundary, AWS security groups, and the AWS Virtual Private Cloud**
- ***Leveraging the AWS ecosystem* — get up to speed on using third party services, adding application functionality, and integrating partner products**

"This is a great resource for anyone considering the jump into cloud computing. Golden accurately explores the roster of AWS services while clearly illustrating ways for developers to make applications easier to build and manage. He manages to address both business requirements and technical content in a way that will appeal to almost any audience."

— Jeff Barr, Sr. Technology Evangelist, Amazon Web Services

Bernard Golden is vice president in charge of enterprise solutions at Dell Enstratius Networks, a global cloud computing management software company. He was named one of the Top Ten Influential People of Cloud Computing by *Wired* magazine in 2012.

Open the book and find:

- **All about the Amazon business philosophy**
- **Basics of the AWS API**
- **Details for setting up AWS storage**
- **Hints for using Amazon Elastic Compute Cloud**
- **How to use AWS core services**
- **Tips for building a simple blog**
- **Steps for creating an effective e-commerce site**
- **Ways to leverage the AWS ecosystem for your needs**

Cover Image: ©iStockphoto.com/scanrail

FOR DUMMIES®
A Wiley Brand

Also available as an e-book

ISBN:978-1-118-57183-5
52999
9 781118 571835