The when, how, and why of enterprise cloud computing

THE Cloud AT Your Service

Jothy Rosenberg
Arthur Mateos

FOREWORD BY ANNE THOMAS MANES

SAMPLE
CHAPTER

MANNING

The Cloud at Your Service

by Jothy Rosenberg
Arthur Mateos

Chapter 2

brief contents

2

Understanding cloud computing classifications

This chapter covers

- Necessary technological underpinnings common to all cloud types
- Classifying the types of clouds and their capabilities
- Choosing the appropriate type of cloud and the best provider for it

Now that we've set the stage for an understanding of what cloud computing is, we can look under the hood and help you understand the different types or classifications of clouds and how they work. Keeping with the under-the-hood analogy, with cloud computing as our vehicle, the modern data center will serve as the engine and virtualization as the vehicle's suspension smoothing out the road. The cloud's API is similar to the dashboard and controls of the vehicle, allowing you to drive it; cloud storage is similar to the trunk, allowing you to transport things; cloud databases are the navigation system (specific information) you need for the trip; and elasticity is the vehicle's transmission that allows the engine's speed to be translated into low- or high-vehicle speeds, which is analogous to allowing your application to support one user and suddenly expand when it needs to support one million. Similar to the variety of vehicles available, there are a wide variety of cloud types. We'll examine the major ones in existence today. Do you need a racing car because you require the speed, or do you need a giant 18-wheeler because of its space capacity?

Let's begin by looking at the six most critical technological underpinnings of the cloud to understand what it's made of. We'll expand on our initial discussion from chapter 1 of the different types of clouds and how they compare and contrast with each other. This will prepare you to make better decisions about which type of cloud you need and how to make best use of it.

2.1 The technological underpinnings of cloud computing

Either through curiosity or because it makes us better drivers and owners, most of us learn the basics of how their car works. Similarly, let's learn about the basic technologies and infrastructure needed to build a cloud, regardless of type, to understand its workings:

- *A cloud needs servers on a network, and they need a home.* That physical home and all the gear in it make up a *data center*.
- *A cloud's servers need to be virtualized.* This is in order to use a large bank of servers effectively. Otherwise, the economics of a huge number of servers won't allow the cloud to be cost effective.
- *A cloud needs an access API.* Without an access API, the virtualized servers in the cloud would be quiet and lonely. Cloud users need a way to access the cloud, provision new virtual servers, get data in and out of storage, start and stop applications on those servers, and decommission servers that are no longer needed. All this needs to be possible remotely, because cloud users never set foot inside the data center.
- *A cloud needs some storage.* It needs to store virtual machine images, users' applications, and persistent data needed by those applications.
- *Cloud applications need a database.* Most applications also need structured data during execution. Consequently, the cloud needs some sort of database.
- *A cloud needs elasticity as a way to expand and contract applications.* A cloud must be dynamically scalable. One of the chief attractions of cloud computing is the ability to have applications that can scale up or down as per the demand the application receives.

In the following six subsections, we'll tackle each of the aforementioned aspects of technology and infrastructure that together form the technological underpinnings of cloud computing.

2.1.1 Achieving high economies of scale with cloud data centers

Revisiting the vehicle analogy, the data center is the car's engine. A *data center*—one that you might find in any large company—is a facility (usually secure) to house a large collection of computers, networking, and communications equipment. But the large internet-based companies, such as Amazon, Yahoo!, Google, Intuit, Apple, and others have, over the years, built up what have to be considered *mega* data centers with thousands of servers. These data centers are the starting point for what is being built out by the cloud providers.

It's useful to understand the structure and the economics of these massive data centers to gauge how much you can scale your operations, how reliable your cloud computing will be, how secure your data will be, and where the economics of public clouds are going. This is particularly important should you decide to build your own *private* cloud. You'll learn more about private clouds later in this chapter, and we've dedicated chapter 4 to the topics of security and private clouds.

THE STRUCTURE OF A DATA CENTER

A data center can occupy one room of a building, one or more floors, or an entire building. Most of the equipment is often in the form of servers mounted in 19-inch rack cabinets, which are usually placed in single rows with corridors between them. This allows people access to the front and rear of each cabinet. Servers differ greatly in size, from 1U servers (which occupy one of 42 slots in a standard rack) to large freestanding storage silos that occupy many tiles on the floor. Mainframe computers and storage devices may be as big as the racks themselves and are placed alongside them. Large data centers may use shipping containers packed with 1,000 or more servers each; when they need to repair or upgrade, they replace the whole container (rather than repairing individual servers).

Clean, unwavering power—and lots of it—is essential. Data centers need to keep their computers running at all times. They should be prepared to handle brownouts and even power outages. The power must be conditioned, and backup batteries and diesel generators must be available to keep power flowing no matter what.

As you can imagine, all that power generates a lot of heat. Data centers must cool their racks of equipment. The most common mode of cooling is air-conditioning; water-cooling is also an option when it's easily available, such as at some of the new data centers along the Columbia River in Washington State. Air-conditioning not only cools the environment but also controls humidity to avoid condensation or static electric buildup.

Network connectivity and ample bandwidth to and from the network backbones are vital, to handle the input and output from the entire collection of servers and storage units. All these servers will be idle if no one can access them.

Another important aspect is physical and logical security. Bigger data centers are targets for hackers all over the world. Some freestanding data centers begin with security through obscurity and disguise the fact that a data center even exists at that location. Guards, mantraps, and state-of-the-art authentication technology keep unauthorized people from physically entering. Firewalls, VPN gateways, intrusion-detection software, and so on keep unauthorized people from entering over the network. (More on all aspects of cloud security in chapter 4.)

Finally, data centers must always assume the worst and have disaster recovery contingencies in place that avoid loss of data and experience the minimum loss of service in case of disaster.

DATA CENTERS: SCALING FOR THE CLOUD

A traditional, large data center dedicated to a single large corporation costs approximately \$100-200 million.[1] Contrast that to the total cost of building the largest mega data centers that provide cloud services: \$500 million or more.[2,3] What is going into that much higher cost, and what can the biggest cloud data centers do that normal companies can't do with their dedicated data centers?

The largest data-center operators like Google, Amazon, and Microsoft situate their data centers in geographic proximity to heavy usage areas to keep network latency to a minimum and to provide failover options. They also choose geographies with access to cheap power. The northwest is particularly advantageous because the available hydropower is the cheapest power in the country and air-conditioning needs are low to zero. Major data centers can use a whopping amount of wattage and cost their owners upward of \$30 million a year for electricity alone, which is why data-center power consumption across the U.S. represents 1.2 percent of total power consumption in the country—and it's rising. The positive side is that cloud data centers use so much power and have so much clout that they can negotiate huge power volume discounts.

Additionally, these giant data centers tend to buy so much hardware that they can negotiate huge volume discounts far beyond the reach of even the largest company that's building a dedicated data center. For example, Amazon spent about \$90 million for 50,000 servers from Rackable/SGI in 2008,[4] which, without the massive volume discounts, would have cost \$215 million.

Servers dominate data-center costs. This is why Google and others are trying to get cheaper servers and have taken to building their own from components. Google relies on cheap computers with conventional multicore processors. A single Google data center has tens of thousands of these inexpensive processors and disks, held together with Velcro tape in a practice that makes for easy swapping of components.

To reduce the machines' energy appetite, Google fitted them with high-efficiency power supplies and voltage regulators, variable-speed fans, and system boards stripped of all unnecessary components, such as graphics chips. Google has also experimented with a CPU power-management feature called *dynamic voltage/frequency scaling*. It reduces a processor's voltage or frequency during certain periods (for example, when you don't need the results of a computing task right away). The server executes its work more slowly, reducing power consumption. Google engineers have reported energy savings of around 20 percent on some of their tests.

In 2006, Google built two cloud computing data centers in Dalles, Oregon, each of which has the acreage of a football field with four floors and two four-story cooling

[1] http://perspectives.mvdirona.com/2008/11/28/CostOfPowerInLargeScaleDataCenters.aspx

[2] http://www.datacenterknowledge.com/archives/2007/11/05/microsoft-plans-500m-illinois-data-center

[3] http://www.theregister.co.uk/2009/09/25/microsoft_chillerless_data_center

[4] http://www.datacenterknowledge.com/archives/2009/06/23/amazon-adds-cloud-data-center-in-virginia

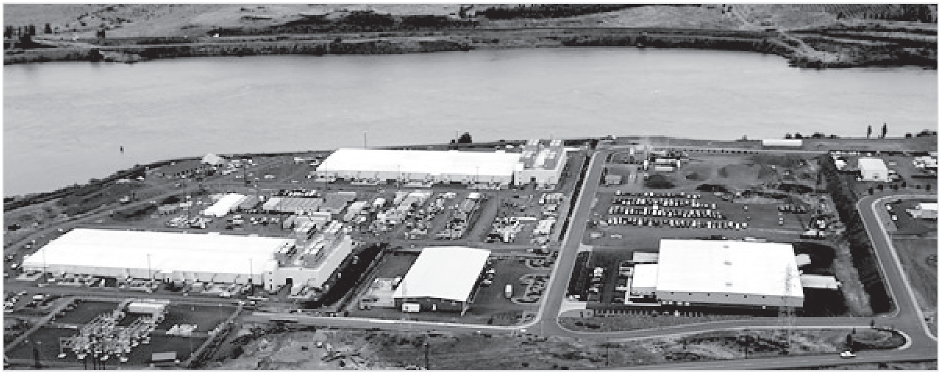


Figure 2.1 Photograph of Google's top-secret Dalles, OR data center, built near the Dalles Dam for access to cheap power. Note the large cooling towers on the end of each football-sized building on the left. These towers cool through evaporation rather than using more power-hungry chillers. Source: Melanie Conner, *New York Times*.

plants (see figure 2.1). The Dalles Dam is strategic for the significant energy and cooling needs of these data centers. (Some new cloud data centers rely on cooling towers, which use evaporation to remove heat from the cooling water, instead of traditional energy-intensive chillers.)

The Dalles data center also benefits from good fiber connectivity to various locations in the U.S., Asia, and Europe, thanks to a large surplus of fiber optic networking, a legacy of the dot-com boom.

In 2007, Google built at least four new data centers at an average cost of $600 million, each adding to its Googleplex: a massive global computer network estimated to span 25 locations and 450,000 servers. Amazon also chose a Dalles location down the river for its largest data center.

Yahoo! and Microsoft chose Quincy, Washington. Microsoft's new facility there has more than 477,000 square feet of space, nearly the area of 10 football fields. The company is tight-lipped about the number of servers at the site, but it does say the facility uses 3 miles of chiller piping, 600 miles of electrical wire, 1 million square feet of drywall, and 1.6 tons of batteries for backup power. And the data center consumes 48 megawatts—enough power for 40,000 homes.

World's servers surpassing Holland's emissions

The management consulting firm McKinsey & Co. reports that the world's 44 million servers consume 0.5 percent of all electricity and produce 0.2 percent of all carbon dioxide emissions, or 80 megatons a year, approaching the emissions of entire countries such as Argentina or the Netherlands.

CLOUD DATA CENTERS: BECOMING MORE EFFICIENT AND MORE FLEXIBLE THROUGH MODULARITY

Already, through volume purchasing, custom server construction, and careful geographic locality, the world's largest data-center owners can build data centers at a fraction of the cost per CPU operation of private corporations. They relentlessly work to widen that gap. The economies-of-scale trend will continue in the cloud providers' favor as they become dramatically more efficient through modular data centers. These highly modular, scalable, efficient, just-in-time data centers can provide capacity that can be delivered anywhere in the world quickly and cheaply.

Figure 2.2 is an artist's rendering of a modular data center (because photographs of such facilities are highly guarded). Corporate data centers can't compete with the myriad economic efficiencies that these mega data centers can achieve today and will fall further and further behind as time goes by.

The goal behind modular data centers is to standardize them and move away from custom designs, enabling a commoditized manufacturing approach. The most striking feature is that such data centers are roofless.

Like Google, Microsoft is driven by energy costs and environmental pressures to reduce emissions and increase efficiency. The company's goal is a power usage effectiveness (PUE) at or below 1.125 by 2012 across all its data centers.

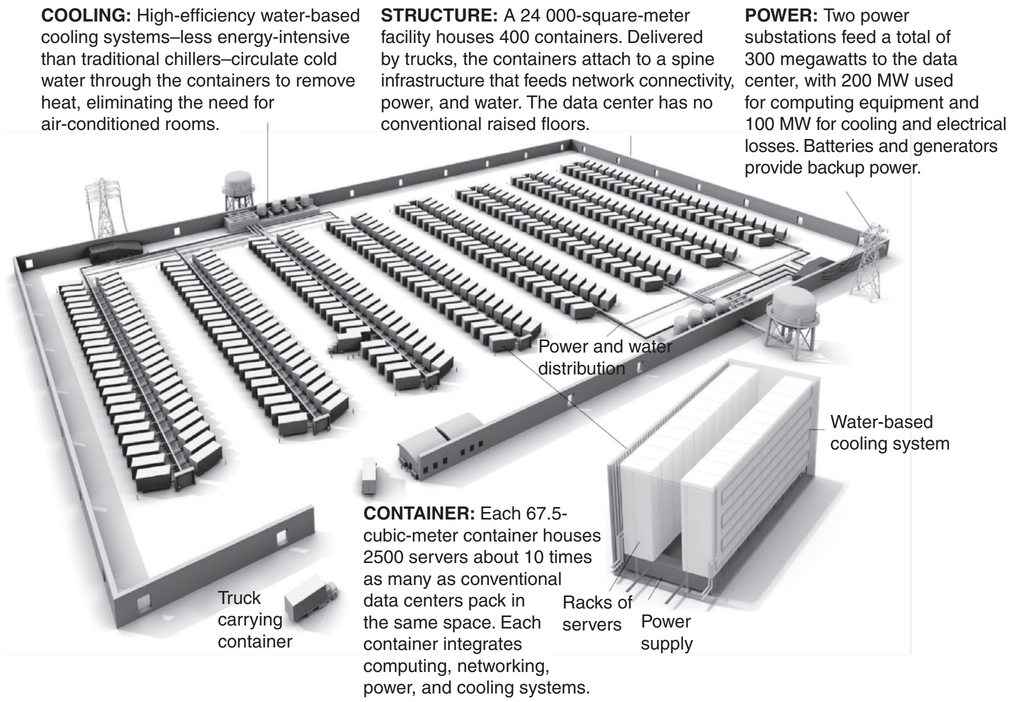

COOLING: High-efficiency water-based cooling systems–less energy-intensive than traditional chillers–circulate cold water through the containers to remove heat, eliminating the need for air-conditioned rooms.
STRUCTURE: A 24 000-square-meter facility houses 400 containers. Delivered by trucks, the containers attach to a spine infrastructure that feeds network connectivity, power, and water. The data center has no conventional raised floors.
POWER: Two power substations feed a total of 300 megawatts to the data center, with 200 MW used for computing equipment and 100 MW for cooling and electrical losses. Batteries and generators provide backup power.
Power and water distribution
Water-based cooling system
CONTAINER: Each 67.5-cubic-meter container houses 2500 servers about 10 times as many as conventional data centers pack in the same space. Each container integrates computing, networking, power, and cooling systems.
Truck carrying container
Racks of servers
Power supply

Figure 2.2 Expandable, modular cloud data center. Notice there is no roof. New containers with servers, power, cooling and network taps can be swapped in and out as needed. Source: *IEEE Spectrum* magazine.

Power usage effectiveness (PUE)

Power usage effectiveness (PUE) is a metric used to determine the energy efficiency of a data center. PUE is determined by dividing the amount of power entering a data center by the power used to run the computer infrastructure within it. PUE is therefore expressed as a ratio, with overall efficiency improving as the quotient decreases toward 1.

According to the Uptime Institute, the typical data center has an average PUE of 2.5. This means that for every 2.5 watts in at the utility meter, only 1 watt is delivered out to the IT load. Uptime estimates that most facilities could achieve 1.6 PUE using the most efficient equipment and best practices. Google and Microsoft are both approaching 1.125, far exceeding what any corporate or cohost data center can achieve.

2.1.2 *Ensuring high server utilization in the cloud with virtualization*

Virtualization, following the car analogy, is the suspension. It provides the high server utilization you need. It smoothes out the variations between applications that need barely any CPU time (they can share a CPU with other applications) and those that are compute intensive and need every CPU cycle they can get. Virtualization is the single-most revolutionary cloud technology whose broad acceptance and deployment truly enabled the cloud computing trend to begin. Without virtualization, and the 60-plus percent server utilization it allows, the economics of the cloud would not work.

VIRTUALIZATION For this book, we're interested primarily in *platform* virtualization. Platform virtualization is a technique to abstract computer resources such that it separates the operating system from the underlying physical server resources. Instead of the OS running on (that is, directly using) hardware resources. The OS interacts instead with a new software layer called a *virtual machine monitor* that accesses the hardware and presents the OS with a virtual set of hardware resources. This means multiple virtual machine images or instances can run on a single physical server, and new instances can be generated and run on demand, creating the basis for elastic computing resources.

As we discussed earlier, virtualization isn't new at all. IBM mainframes used time-sharing virtualization in the '60s to enable many people to share a large computer without interacting or interfering with each other. Previously, constraints of scheduling dedicated time on these machines required you to get all your work for the day done in that scheduled time slot. The concept of virtual memory, introduced around 1962, although considered pretty radical, ultimately freed programmers from having to constantly worry about how close they were to the limits of physical memory. Today, server virtualization is proving equally dramatic for application deployment and scaling. And it's the key enabler for the cloud. How did this happen?

The average server in a corporate data center has typical utilization of only 6 percent.[5] Even at peak load, utilization is no better than 20 percent. In the best-run data centers, servers only run on average at 15 percent or less of their maximum capacity. But when these same data centers fully adopt server virtualization, their CPU utilization increases to 65 percent or higher. For this reason, in a few short years, most corporate data centers have deployed hundreds or thousands of virtual servers in place of their previous model of one server on one hardware computer box. Let's see how server virtualization works to make utilization jump this dramatically.

HOW IT WORKS

Server virtualization transforms or *virtualizes* the hardware resources of a computer—including the CPU, RAM, hard disk, and network controller—to create a fully functional virtual machine that can run its own operating system and applications like a physical computer. This is accomplished by inserting a thin layer of software directly on the computer hardware that contains a virtual machine monitor (VMM)—also called a *hypervisor*—that allocates hardware resources dynamically and transparently. Multiple guest operating systems run concurrently on a single physical computer and share hardware resources with each other. By encapsulating an entire machine, including CPU, memory, operating system, and network devices, a virtual machine becomes completely compatible with all standard operating systems, applications, and device drivers. You can see the virtual machine architecture for VMware on the x86 in figure 2.3.

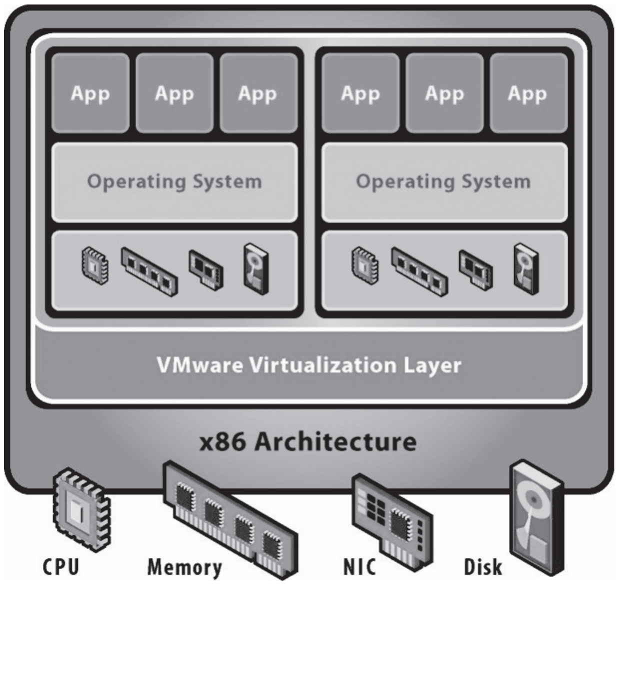


Figure 2.3 Virtual machine architecture using VMware as an example. The virtualization layer is what interfaces directly with all hardware components, including the CPU. That layer then presents each guest operating system with its own array of *virtual* hardware resources. The guest OS doesn't operate differently than it would if installed on the bare hardware, but now several instances of guest OSs with all their applications can share a single physical device and have higher effective utilization. Source: VMWare.

[5] McKinsey & Company, 2008 Data Center Efficiency report.

VIRTUALIZATION AS APPLIED TO THE CLOUD

When virtualization passed muster with enterprise architects and CIOs, it had arrived. It was all about saving money. Enterprises began seeing utilization of their hardware assets increase dramatically. It was easy to go from the typical 5 or 6 percent to 20 percent. They could get 65 percent utilization or better with good planning.

In addition to increased utilization and the associated cost savings, virtualization in corporate data centers set the stage for cloud computing in several interesting ways. It decoupled users from implementation; it brought speed, flexibility, and agility never before seen in corporate data centers; and it broke the old model of software pricing and licensing. Let's look at table 2.1 for more clarity.

Table 2.1 Impact of virtualization on corporate data centers

Benefit	Explanation
Decouples users from implementation	The concept of a virtual server forces users to not worry about the physical servers or their location. Instead, they focus on service-level agreements and their applications.
Decreases server provisioning from months to minutes	Getting a (physical) server requisitioned, installed, configured, and deployed takes larger organizations 60–90 days and some 120 days. In the virtual server model, it's literally minutes or hours from request to fully ready for application deployment, depending on how much automation has been put in place.
Breaks software pricing and licensing	No longer can the data center charge for an entire server or every server the software runs on. Instead, they have to charge for actual usage—a whole new model for IT.

Table 2.1 illustrates the services the cloud providers offer. We also see a growing recognition of and readiness for the cloud within the enterprise. This is because the model change that virtualization has already brought to enterprise IT has prepared companies to adapt more easily to the cloud computing model.

Let's look at a scenario that uses thousands of physical servers. Each one is virtualized and can run any number of guest OSs, can be configured and deployed in minutes, and is set up to bill by the CPU hour. The combination of cheap, abundant hardware and virtualization capability, coupled with automated provisioning and billing allows the huge economies of scale now achievable in the mega data centers to be harnessed through cloud computing. This is possible because of virtualization, much as car suspension systems enable vehicles to speed up without killing the occupants at every bump in the road.

But a powerful engine (data center) and a smooth suspension (virtualization) aren't enough. Following the vehicle analogy, you need a set of controls to start, stop, and steer the car; you need an API to control your cloud.

2.1.3 Controlling remote servers with a cloud API

The API is to a cloud what the dashboard and controls are to a car. You have tremendous power under that hood, but you need the dials and readouts to know what the vehicle is doing. You need the steering wheel, accelerator, and brake to control it. Remember, you'd never drive fast if you didn't have good brakes.

When you have a cloud, you need a way to access it. The highest-level clouds—those offering Software as a Service (SaaS) applications—offer a browser-based web interface. Lower-level clouds—those offering Infrastructure as a Service (IaaS)—also need a way to access applications. Each type of cloud must provide some kind of API that can be used to provision resources, configure and control them, and release them when they're no longer needed.

An API is necessary to engage the service of a cloud provider. It's a way for the vendor to expose service features and potentially enable competitive differentiation. For example, Amazon's EC2 API is a SOAP- and HTTP Query-based API used to send proprietary commands to create, store, provision, and manage Amazon Machine Images (AMIs). Sun's Project Kenai Cloud API specification is a Representational State Transfer (REST)-ful API for creating and managing cloud resources, including compute, storage, and networking components.

REST ARCHITECTURE AND RESTFUL APIS Representational State Transfer (REST) is a style of software architecture for distributed hypermedia systems, such as the World Wide Web. The REST architectural style was developed in parallel with the HTTP protocol. The largest-known implementation of a system conforming to the REST architectural style is the World Wide Web. In fact, REST can be considered a post hoc description of the features of the web that made the web successful. REST-style architectures consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of *representations* of *resources.* A resource can be any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of a resource. Conforming to the REST constraints is referred to as being *RESTful.*

Because your cloud applications will be the lifeblood of your company, you'll want to ensure that only authorized parties can access your applications. If an application was running in your company's secure data center protected by layers of physical and logical security you'd be certain that no unauthorized person could access it. Here, because everything having to do with your application and the server it runs on is by definition accessible over the internet, the approach Amazon and others take to security is to issue X.509 public key pairs initially and then require a key on every API call. This ensures that the caller has the credentials to access the infrastructure.

To understand a cloud API—for which there isn't yet an accepted standard—it's best to look at Amazon's cloud API as the default standard as they're the leaders. Table 2.2 outlines some of the basic definitions and operations central to the Amazon cloud API.

Table 2.2 Basic terms and operations of the Amazon EC2 API

Term	Description
AMI	An Amazon Machine Image is an encrypted and signed machine image suitable to run in a virtual server environment. For example, it may contain Linux, Apache, MySQL, or PHP, as well as the application of the AMI's owner. AMIs can be public (provided by Amazon), private (custom designed by its creator), paid (purchased from a third party), or shared (created by the community for free). AMIs can be stored in Amazon's Simple Storage Service (S3).
Instance	The result of launching an AMI is a running system called an *instance*. When an instance terminates, the data on that instance vanishes. For all intents and purposes, an Instance is identical to a traditional host computer.
Standard flow	1. Use a standard AMI by customizing an existing one. 2. Bundle the AMI, and get an AMI ID to enable launching as many instances of the AMI as needed. 3. Launch one or more instances of this AMI. 4. Administer and use the running instance(s).
Connecting	From a web browser, go to http://<hostname>, where <hostname> is your instance's public hostname. If you want to connect to a just-launched public AMI that hasn't been modified, run the `ec2-get-console-output` command. The result in either case enables you to log in as root and exercise full control over this instance, just like any host computer you could walk up to in a data center.

We've barely scratched the surface of all the concepts and corresponding API calls that exist in Amazon's API. Documentation is available at http://docs.amazonwebservices.com. APIs also cover these areas:

- Using instance addressing
- Using network security
- Using regions and availability zones
- Using Amazon Elastic Block Store (EBS)
- Using auto scaling, elastic load balancing, and Amazon CloudWatch
- Using public data sets
- Using Amazon's Virtual Private Cloud

We'll revisit select aspects of the cloud API at various points throughout the book. Let's leave this now and talk about the next important layer in what it takes to set up and use a cloud: cloud storage.

2.1.4 Saving persistent data in cloud storage

You store your luggage in the trunk of your car. Similarly, the cloud provides a place to store your machine images, your applications, and any data your applications need.

Cloud storage has also increased in popularity recently for many of the same reasons as cloud computing. Cloud storage delivers *virtualized storage on demand* over a network based on a request for a given quality of service (QoS). Unlike the long provisioning lead times required in corporate data centers, there is no need to purchase storage or in some cases even provision it before storing data. Typically, you pay for transit of data into the cloud and, subsequently, a recurring fee based on the amount of storage consumption your data uses.

You can use cloud storage in many different ways. For example, you can back up local data (such as on a laptop) to cloud storage; a virtual disk can be synched to the cloud and distributed to other computers; and you can use it as an archive to retain (under some policy) data for regulatory or other purposes.

You can use cloud storage for applications that provide data directly to their clients via the network. The application redirects the client to a location at the cloud storage provider for the data. Media such as audio and video files are examples. The network requirements for streaming data files can be made to scale in order to meet the demand without affecting the application.

The type of interface used for this is HTTP. You can fetch the file from a browser without having to do any special coding, and the correct application is invoked automatically. But how do you get the file there in the first place, and how do you make sure the storage you use is of the right type and QoS? Again, many offerings expose an interface for these operations, and it's not surprising that many of these interfaces use REST principles. This is typically a data-object interface with operations for creating, reading, updating, and deleting the individual data objects via HTTP operations.

Keeping with Amazon's APIs as good examples to study, we've outlined a simple API dealing with Amazon's S3 in table 2.3.

A cloud storage standard

The Storage Networking Industry Association has created a technical work group to address the need for a cloud storage standard. The new Cloud Data Management Interface (CDMI) enables interoperable cloud storage and data management. In CDMI, the underlying storage space exposed by the interfaces is abstracted using the notion of a *container*. A container is not only a useful abstraction for storage space, but also serves as a grouping of the data stored in it and a point of control for applying data services in the aggregate.

Table 2.3 Basic terms and operations of Amazon S3

Terms	Description
Object	Fundamental entity stored in S3. Each object can range in size from 1 byte to 5 GB. Each object has object data and metadata. *Metadata* is a set of name-value pairs that describe the data.
Bucket	Fundamental container in S3 for data storage. Objects are uploaded into buckets. There is no limit to the number of objects you can store in a bucket. The bucket provides a unique namespace for the management of objects contained in the bucket. Bucket names are global, so each developer can own only up to 100 buckets at a time.
Key	A key is the unique identifier for an object within a bucket. A bucket name plus a key uniquely identifies an object within all of S3.
Usage	1. Create a bucket in which to store your data. 2. Upload (write) data (objects) into the bucket. 3. Download (read) the data stored in the bucket. 4. Delete some data stored in the bucket. 5. List the objects in the bucket.

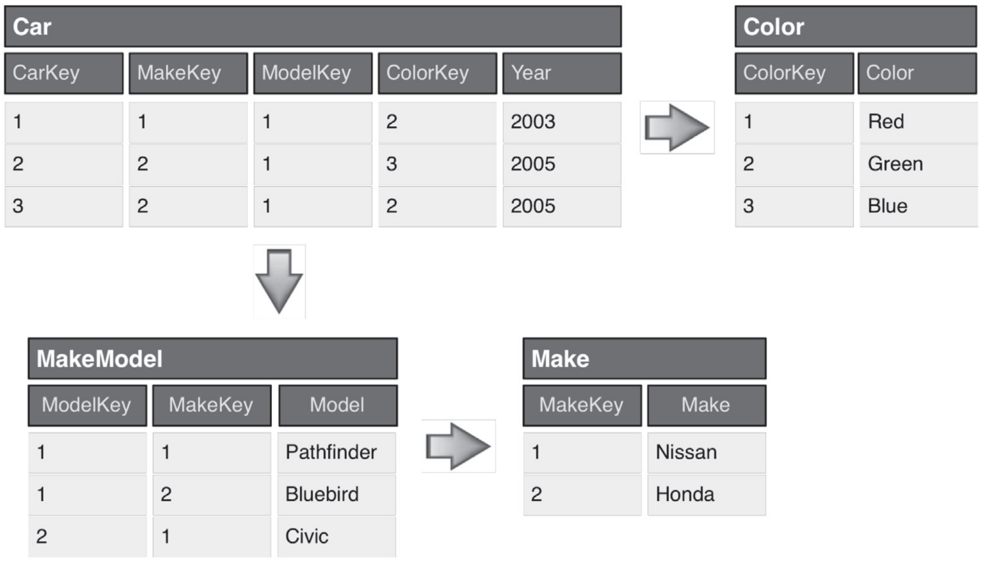
In many cases, the coarse granularity and unstructured nature of cloud storage services such as S3 aren't sufficient for the type of data access required. For many applications, an alternative structured data-storage method is required. Let's explore how databases in the cloud work (and don't).

2.1.5 *Storing your application's structured data in a cloud database*

Your car's navigation system provides constant updates about your current location and destination during your journey. It guides you through the route you need to take. This data, although critical for the trip, isn't useful afterward. The navigation system is to the car what a cloud database is to an application running in the cloud: it's transactional data created and used during the running of that application. When we think of transactional data stored in databases, we usually think of relational databases.

What is a Relational Database Management System (RDBMS)? Why do we frequently hear that they don't work in the cloud? An RDBMS is a database management system in which you store data in the form of tables; the relationship among the data is also stored in the form of tables. You can see this in the simple relation in figure 2.4.

RDBMS A database management system (DBMS) based on the relational model. *Relational* describes a broad class of database systems that at a minimum present the data to the user as relations (a presentation in tabular form—that is, as a collection of tables with each table consisting of a set of rows and columns—can satisfy this property) and provide relational operators to manipulate the data in tabular form. All modern commercial relational databases employ SQL as their query language, leading to a shorthand for RDBMSs as *SQL databases*.



Car

CarKey	MakeKey	ModelKey	ColorKey	Year
1	1	1	2	2003
2	2	1	3	2005
3	2	1	2	2005

Color

ColorKey	Color
1	Red
2	Green
3	Blue

MakeModel

ModelKey	MakeKey	Model
1	1	Pathfinder
1	2	Bluebird
2	1	Civic

Make

MakeKey	Make
1	Nissan
2	Honda

Figure 2.4 A simple example of how a relational database works. Four tables map out relationships among the data. Because a separate table lists the car manufacturers and colors, there is no need to separately list a red Nissan and a blue Nissan. But to fully understand what the car with CarKey 1 is, you must do a join of the Car, Color, MakeModel, and Make tables.

The challenge for an RDBMS in the cloud is scaling. Applications having a fixed number of users and workload requiring an RDBMS won't have any problems. Most cloud providers have an RDBMS offering for these cases. But when applications are launched in environments that have massive workloads, such as web services, their scalability requirements can change quickly and grow large. The first scenario can be difficult to manage if you have a relational database sitting on a single in-house server. For example, if your load triples overnight, how quickly can you upgrade your hardware? The second scenario can be too difficult to manage with a relational database, because it becomes a bottleneck choking the application's ability to scale. We'll cover solutions to this in depth in chapter 5.

As you've already learned, one of the core benefits of the cloud is the ability to quickly (or automatically, as we'll show) add more servers to an application as its load increases, thereby scaling it to heavier workloads. But it's hard to expand an RDBMS this way. You have to either replicate data across the new servers or partition between them. In either case, adding a machine requires data to be copied or moved to the new server. This data shipping is a time-consuming and expensive process, so databases are unable to be dynamically and efficiently provisioned on demand.

A big challenge with RDBMS partitioning or replicating is maintaining *referential integrity*. Referential integrity requires that every value of one attribute (column) of a relation (table) exist as a value of another attribute in a different (or the same) relation (table). A little less formally, any field in a table that's declared a foreign key can contain only values from a parent table's primary key or a candidate key. In practice, this means that deleting a record that contains a value referred to by a foreign key in another table break's referential integrity. When you partition or replicate a database, it becomes nearly impossible to guarantee maintenance of referential integrity across all databases. This extremely useful property of RDBMS—its ability to construct a relation out of lots of small index tables that are referred to by values in records—becomes unworkable when these databases have to scale to deal with huge workloads, but cloud applications are otherwise ideally suited for this purpose.

THE NOSQL MOVEMENT

Since 1998, there has been a small but rapidly growing movement away from SQL databases. Instead, participants in this movement promote a class of nonrelational data stores that break some of the fundamental guarantees of SQL in favor of being able to reach massive scale. This is obviously important for *some* cloud applications. These non-SQL data stores may not require fixed table schemas and usually avoid join operations. They're described as scaling *horizontally*. Some categorize them as *structured storage*.

A new non-SQL type of database, generically a key-value database, does scale well. Consequently, it's started to be used in the cloud. Key-value databases are *item-oriented*, meaning all relevant data relating to an item are stored in that item. A table can

NoSQL architecture

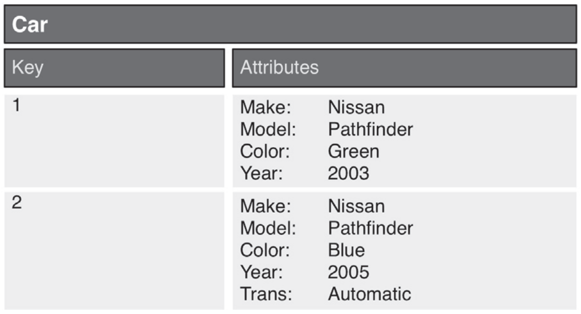
Relational databases have a limitation on handling big data volumes and typical modern workloads. Today's scale is unprecedented and can't be handled with relational SQL databases. Examples of enormous scale are synonymous with the most popular sites: Digg's 3 TB for green badges, Facebook's 50 TB for inbox search, and eBay's 2 PB overall data.

NoSQL systems often provide weak consistency guarantees, such as *eventual* consistency and transactions restricted to single data items; in most cases, you can impose full ACID (atomicity, consistency, isolation, durability) guarantees by adding a supplementary middleware layer.

Several NoSQL systems employ a distributed architecture, with the data being held in a redundant manner on several servers, often using a distributed hash table. In this way, the system can be scaled up easily by adding more servers, and failure of a server can be tolerated.

Some NoSQL advocates promote simple interfaces, such as associative arrays or key-value pairs. Other systems, such as native XML databases, promote support of the XQuery standard.

Clearly, we're in the early days of cloud evolution, with a lot of development yet to come.



Car	
Key	Attributes
1	Make: Nissan Model: Pathfinder Color: Green Year: 2003
2	Make: Nissan Model: Pathfinder Color: Blue Year: 2005 Trans: Automatic

Figure 2.5 The same data as in figure 2.4, shown for a key-value type of database. Because all data for an item (row) is contained in that item, this type of database is trivial to scale because a data store can be split (by copying some of the items) or replicated (by copying all the items to an additional data store), and referential integrity is maintained.

contain vastly different items. For example, a table may contain car makes, car models, and car color items. This means data are commonly duplicated between items in a table (another item also contains Color: Green). You can see this in figure 2.5. In an RDBMS, this is anathema; here, this is accepted practice because disk space is relatively cheap. But this model allows a single item to contain all relevant data, which improves scalability by eliminating the need to join data from multiple tables. With a relational database, such data needs to be joined to be able to regroup relevant attributes. This is the key issue for scaling—if a join is needed that depends on shared tables, then replicating the data is hard and blocks easy scaling.

When companies set out to create a public computing cloud (such as Amazon) or build massively parallel, redundant, and economical data-driven applications (such as Google), relational databases became untenable. Both companies needed a way of managing data that was almost infinitely scalable, inherently reliable, and cost effective. Consequently, both came up with nonrelational database systems based on this key-value concept that can handle massive scale. Amazon calls its cloud database offering SimpleDB, and Google calls its BigTable. (Both were developed long before either company launched a cloud. They created these structures to solve their own problems. When they launched a cloud, the same structures became part of their cloud offerings.)

Google's BigTable solution was to develop a relatively simple storage management system that could provide fast access to petabytes of data, potentially redundantly distributed across thousands of machines. Physically, BigTable resembles a B-tree index-organized table in which branch and leaf nodes are distributed across multiple machines. Like a B-tree, nodes split as they grow, and—because nodes are distributed—this allows for high scalability across large numbers of machines. Data elements in BigTable are identified by a primary key, column name, and, optionally, a timestamp. Lookups via primary key are predictable and relatively fast. BigTable provides the data storage mechanism for Google App Engine. You'll learn about this PaaS cloud-based application environment in detail later in this chapter.

Google charges $180 per terabyte per month for BigTable storage. Here are some examples of BigTable usage (in Python):

This code declares a data store class:

```
class Patient(db.Modal);
    firstName = db.UserProperty()
    lastName = db.UserProperty()
    dateOfBirth = db.DateTimeProperty()
    sex = db.UserProperty()
```

This code creates and stores an object:

```
patient = Patient()

patient.firstName = "George"
patient.lastName = "James"
dateOfBirth = "2008-01-01"
sex = "M"

patient.put()
```

This code queries a class:

```
patients = Patient.all()

for patient in patients:
    self.response.out.write('Name %s %s. ',
            patient.firstName, patient.lastName)
```

And this code selects the 100 youngest male patients:

```
allPatients = Patient.all()
allPatients.filter('sex=', 'Male')
allPatients.order('dateOfBirth')
patients = allPatients.fetch(100)
```

Amazon's SimpleDB is conceptually similar to BigTable and forms a key part of the Amazon Web Services (AWS) cloud computing environment. (Microsoft's SQL Server Data Services [SSDS] provides a similar capability in their Azure cloud.) Like BigTable, this is a key-value type of database. The basic organizing entity is a *domain*. Domains are collections of items that are described by attribute-value pairs. You can see an abbreviated list of the SimpleDB API calls with their functional description in table 2.4.

Table 2.4 Amazon's SimpleDB API summary

API call	API functional description
`CreateDomain`	Creates a domain that contains your dataset.
`DeleteDomain`	Deletes a domain.
`ListDomains`	Lists all domains.
`DomainMetadata`	Retrieves information about creation time for the domain, storage information both as counts of item names and attributes, and total size in bytes.
`PutAttributes`	Adds or updates an item and its attributes, or adds attribute-value pairs to items that exist already. Items are automatically indexed as they're received.
`BatchPutAttributes`	For greater overall throughput of bulk writes, performs up to 25 `PutAttribute` operations in a single call.

Table 2.4 Amazon's SimpleDB API summary (*continued*)

API call	API functional description
`DeleteAttributes`	Deletes an item, an attribute, or an attribute value.
`GetAttributes`	Retrieves an item and all or a subset of its attributes and values.
`Select`	Queries the data set in the familiar "Select target from *domain_name* where *query_expression*" syntax. Supported value tests are `=`, `!=`, `<`, `>`, `<=`, `>=`, `like`, `not like`, `between`, `is null`, `isn't null`, and `every()`. Example: `select * from mydomain where every(keyword) = "Book"`. Orders results using the `SORT` operator, and counts items that meet the condition(s) specified by the predicate(s) in a query using the `Count` operator.

Converting an existing application to use one of these cloud-based databases is somewhere between difficult and not worth the trouble; but for applications already using the Object-Relational Mapping (ORM)-based frameworks, these cloud databases can easily provide core data-management functionality. They can do it with compelling scalability and the same economic benefits of cloud computing in general. But as table 2.5 illustrates, there are definite drawbacks to these new types of cloud databases that you must take into account when contemplating a shift to the cloud.

Table 2.5 Cloud database drawbacks

Database use	Challenges faced with a cloud database
Transactional support and referential integrity	Applications using cloud databases are largely responsible for maintaining the integrity of transactions and relationships between tables.
Complex data access	Cloud databases (and ORM in general) excel at single-row transactions: get a row, save a row, and so on. But most nontrivial applications have to perform joins and other operations that cloud databases can't.
Business Intelligence	Application data has value not only in terms of powering applications but also as information that drives business intelligence. The dilemma of the pre-relational database, in which valuable business data was locked inside impenetrable application data stores, isn't something to which business will willingly return.

Cloud databases could displace the relational database for a significant segment of next-generation, cloud-enabled applications. But business is unlikely to be enthusiastic about an architecture that prevents application data from being used for business intelligence and decision-support purposes, which fundamentally require a relational database. An architecture that delivered the scalability and other advantages of cloud databases without sacrificing information management would fit the bill. We can expect a lot of innovation and advancements in these database models over the next few years.

The last technological underpinning you need to learn about is elasticity, the transmission in the ongoing vehicle analogy.

2.1.6 Elasticity: scaling your application as demand rises and falls

The transmission smoothly adapts the speed of a car's wheels to the engine speed as you vary the accelerator position. Similarly, *elasticity* enables an application running in a cloud to smoothly expand and contract according to demand. More precisely, elasticity is the ability to have capacity as demand increases and to release that capacity when you're done with it. Many big organizations have been close to disaster or faced it because of scalability failures in times of need.

Elasticity and celebrity deaths

In July 2009, two celebrity deaths occurred on the same day. First, *Charlie's Angels* star Farrah Fawcett died, which resulted in a minor news flurry. Then, later in the afternoon, a major web storm erupted when news of Michael Jackson's death hit the social web. Unexpectedly, Twitter had major scaling issues dealing with the sudden influx of hundreds of thousands of tweets as news of Jackson's death spread. But Twitter wasn't alone.

According to TechCrunch, "Various reports had the AOL-owned TMZ, which broke the story, being down at multiple points throughout the ordeal. As a result, Perez Hilton's hugely popular blog may have failed as people rushed there to try and confirm the news. Then the *LA Times* had a report saying Jackson was only in a coma rather than dead, so people rushed there, and that site went down. (The *LA Times* eventually confirmed his passing.)"

Numerous examples exist of a news story, a product announcement, or even the infamous Victoria's Secret Super Bowl commercial, sending people directly to a web site that then crashes. Too much traffic meets with insufficient capacity and results in catastrophe. When people are directed to a site and it then breaks down, their reaction is to not try that again. These issues severely hurt a company's business. This illustrates the criticality of being able to scale as capacity dynamically grows.

Scalability is about the cloud platform being able to handle an increased load of users working on a cloud application. *Elasticity* is the ability of the cloud platform to scale up or down based on need without disrupting the business. Without this, the economies of moving a business/application to the cloud don't make sense.

The example code snippets that follow set up an EC2 application to be load balanced and auto-scaled (that is, elastic) with a minimum of 2 instances and a maximum of 20 instances. Auto-scaling in this example is configured to scale out by 1 instance when the application's average CPU utilization exceeds a threshold of 80 percent and scale in by 1 instance when it drops below 40 percent for 10 minutes.

Call `CreateLoadBalancer` with the following parameters:

```
    AvailabilityZones = us-east-1a
```

```
LoadBalancerName = MyLoadBalancer
Listeners = lb-port=80,instance-port=8080,protocol=HTTP
```

Call `CreateLaunchConfiguration` with the following parameters:

```
ImageId = myAMI
LaunchConfigurationName = MyLaunchConfiguration
InstanceType = m1.small
```

Call `CreateAutoScalingGroup` with the following parameters:

```
AutoScalingGroupName = MyAutoScalingGroup
AvailabilityZones = us-east-1a
LaunchConfigurationName = MyLaunchConfiguration
LoadBalancerNames = MyLoadBalancer
MaxSize = 20
MinSize = 2
```

Call `CreateOrUpdateScalingTrigger` with the following parameters:

```
AutoScalingGroupName = MyAutoScalingGroup
MeasureName = CPUUtilization
Statistic = Average
TriggerName = MyTrigger1a
Namespace = AWS/EC2
Period = 60
LowerThreshold = 40
LowerBreachScaleIncrement = -1
UpperThreshold = 80
UpperBreachScaleIncrement = 1
BreachDuration = 600
```

You learned in chapter 1 that there is more than one flavor of cloud computing. Let's combine what you learned in chapter 1 about the different types of clouds with what you now know about the six critical enabling technologies in clouds to better understand how these different flavors of clouds work, what they offer, and how they differ. The next section will help you better understand which is best for you.

2.2 Understanding the different classifications of clouds

Now that you've learned about the technological underpinnings of cloud computing, such as virtualization, elasticity, storage, and databases, it's useful to understand how those concepts are employed in the different types (classifications) of cloud computing services being offered. Let's go back to the taxonomy of cloud types from chapter 1—IaaS, PaaS, and DaaS—to classify the clouds from the most prominent providers in the industry.

2.2.1 Amazon EC2: Infrastructure as a Service

Amazon EC2 is categorized as IaaS (some cloud observers call it HaaS, but Amazon has added so many additional services that Hardware as a Service would now be a misnomer). It was the first and is by far the biggest in this category. Amazon opened its service in 2006 after initially using excess capacity from its retail operation. The company claimed to have over 500,000 users by the end of 2008.

Amazon EC2 is the most general purpose of the major clouds but has the least support for automatic scaling or failover, both of which have to be programmed into the application. This is in contrast to the automatic and invisible scaling that occurs in the PaaS types of clouds, such as Google's AppEngine, which we'll discuss in section 2.2.3. In IaaS-type clouds, such as EC2, elasticity requires careful programming using their APIs. On the other hand, you can use any programming language, and you have complete control over your application in an IaaS cloud. Sure, it requires more manual work, but you get something that has the appearance of being physical hardware that you have control over from the operating system outward. The LAMP stack is the easiest and most common EC2 configuration (see table 2.6).

Table 2.6 The components of the LAMP stack in an IaaS cloud

L	Linux	Operating system
A	Apache	Web server
M	MySQL	Relational database
P	PHP	Server side of website

Amazon EC2 and Xen paravirtualization

Amazon EC2 utilizes a customized version of the open source Xen hypervisor, taking advantage of paravirtualization. Because paravirtualized guest OSs rely on the hypervisor to provide support for operations that normally require privileged access, the guest OS runs with no elevated access to the CPU.

Paravirtualization is more efficient than a virtualized environment where the guest OS runs unmodified. But the OS must be ported to the paravirtualized environment so that certain OS tasks that would have to be performed by the VMM and run more slowly can be directly executed by the guest OS. This is why Amazon doesn't run any OS you may desire—it runs only OSs that it or the original vendor has ported and fully tested.

Amazon has an extensive API for all its services, some of which are described in table 2.7. It has a SOAP as well as a simple HTML (`GET`, `POST`) form for its APIs. The company needs only a dozen and a half calls to request and configure virtualized hardware in the cloud.

Table 2.7 Other Amazon cloud services (effectively providing some PaaS capabilities)

Service	Description
Simple Storage Service (S3)	Cloud storage used to store and retrieve large amounts of data from anywhere on the web through a simple API. Well integrated with EC2: AMIs are stored in S3, and data transferred from S3 to EC2 doesn't invoke separate charges.

Table 2.7 Other Amazon cloud services (effectively providing some PaaS capabilities) (*continued*)

Service	Description
SimpleDB	Provides the core database functions of indexing (special organizational entities for faster lookups) and querying. Avoids the big expense of relational database licensing, the requisite DBA, and the complex setup. But it isn't a relational database, has no schema, and doesn't work with SQL.
CloudFront	A web service for content delivery that competes with Akamai. Provides an easy way to distribute content to end users with low latency and high data-transfer speeds in a pay-as-you-go model.
Simple Queue Service (SQS)	A hosted queue for storing messages as they travel between computers. Useful for moving data between distributed components of applications that perform different tasks, without losing messages or requiring each component to be always available.

EC2 pricing starts at roughly a nickel per small Linux-based instance (CPU) hour, up to about half a dollar on a high-end Linux instance.[6] S3 pricing is about $0.15 per GB per month, scaling downward as more storage is used.

2.2.2 Microsoft Azure: Infrastructure as a Service

Microsoft's Azure is IaaS in the same way as Amazon EC2, but it also has other services that operate more at the PaaS level. Many of Microsoft's end-user applications are being recast to run in the cloud. As a result, increasingly, this overall platform is trying to reach the SaaS level to head off Google's thrust against Microsoft Office with the Google Docs and Google Apps SaaS offerings.

The box labeled Windows Azure in figure 2.6 is Windows Server 2008 modified to run in the cloud environment. This means it was paravirtualized to make it run efficiently in the virtualized environment created by running Microsoft's Hypervisor on bare hardware in Microsoft's cloud data centers.

Internally, the OS layer—derived from Windows Server 2008—consists of four pillars: storage (like a file system); the fabric controller, which is a management system for modeling/deploying and provisioning; virtualized computation/VM; and a development environment, which allows developers to emulate Windows Azure on their desktop and plug in Visual Studio, Eclipse, or other tools to write cloud applications against it. Because of this architecture, you merely have to deploy Azure on a single machine; then, you can duplicate multiple instances of it on the rest of the servers in the cloud using virtualization technology.

[6] http://aws.amazon.com/ec2/pricing/

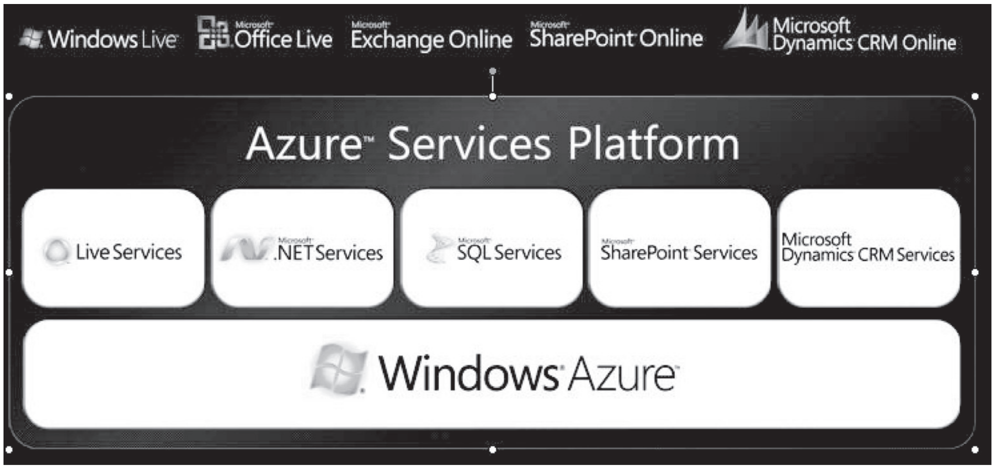

Windows Live
Microsoft Office Live
Microsoft Exchange Online
Microsoft SharePoint Online
Microsoft Dynamics CRM Online
Azure Services Platform
Live Services
Microsoft .NET Services
Microsoft SQL Services
Microsoft SharePoint Services
Microsoft Dynamics CRM Services
Windows Azure

Figure 2.6 The Windows Azure architecture and framework. At the bottom level is the Windows Azure operating system. This runs in the virtualization environment created by Microsoft's Hypervisor running on bare hardware. At the top layer are end-user applications, which Microsoft is recasting to be delivered as SaaS. Source: Microsoft.

Applications for Microsoft's Azure are written in proprietary programming environments like Visual Studio using the .NET libraries, and compiled to the Common Language Runtime, Microsoft's language-independent managed environment.

WINDOWS AZURE API

The Windows Azure API is a REST-based API that uses X.509 client certificates for authentication. Table 2.8 lists a portion of the API, giving you an idea of how applications running in the Azure cloud are manipulated and controlled. This is the set of calls related to performing operations on hosted services.

Similar to Amazon, a set of building-block services run on top of Azure creating Microsoft's PaaS capabilities. The initial set of services includes the following:

- Live Services
- SQL Services
- .Net Services
- SharePoint Services
- CRM Services

You can treat these lower-level services as APIs—they have no user interface elements—when constructing cloud applications.

Azure pricing is comparable to Amazon with computing time set at $0.12 per hour, storage at $0.15 per GB, and storage transactions at $0.01 per 10 KB. For the structured database, fees for the web edition are set at up to 1 GB relational database at $9.99 per month and for the business edition up to 10 GB relational database at $99.99 per month. A tiered, all-you-can-eat (within limits) model is said to be coming.

Table 2.8 A portion of the RESTful Windows Azure API

Service	Description
List Hosted Services	Lists the hosted services available under the current subscription.
`GET https://management.core.windows.net/<subscription-id>/services/hostedservices`	
Get Hosted Service Properties	Retrieves system properties for the specified hosted service. These properties include the service name and service type; the name of the affinity group to which the service belongs, or its location if it isn't part of an affinity group; and, optionally, information about the service's deployments.
`GET https://management.core.windows.net/<subscription-id>/services/hostedservices/<service-name>`	
Create Deployment	Uploads a new service package and creates a new deployment on staging or production.
`POST https://management.core.windows.net/<subscription-id>/services/hostedservices/<service-name>/deploymentslots/<deployment-slot-name>`	
Get Deployment	May be specified as follows. Note that you can delete a deployment either by specifying the deployment slot (staging or production) or by specifying the deployment's unique name.
`GET https://management.core.windows.net/<subscription-id>/services/hostedservices/<service-name>/deploymentslots/<deployment-slot>/` `GET https://management.core.windows.net/<subscription-id>/services/hostedservices/<service-name>/deployments/<deployment-name>/`	
Swap Deployment	Initiates a virtual IP swap between the staging and production deployment slots for a service. If the service is currently running in the staging environment, it's swapped to the production environment. If it's running in the production environment, it's swapped to staging. This is an asynchronous operation whose status must be checked using Get Operation Status.
`POST https://management.core.windows.net/<subscription-id>/hostedservices/<service-name>`	

Table 2.8 A portion of the RESTful Windows Azure API (*continued*)

Service	Description
Delete Deployment	Deletes the specified deployment. This is an asynchronous operation.

```
DELETE
https://management.core.windows.net/<subscription-id>/services/
   hostedservices/
     <service-name>/deploymentslots/<deployment-slot>
DELETE
https://management.core.windows.net/<subscription-id>/services/
   hostedservices/
     <service-name>/deployments/<deployment-name>
```

2.2.3 Google App Engine: Platform as a Service

App Engine is a pure PaaS cloud targeted exclusively at traditional web applications, enforcing an application structure of clean separation between a stateless computation tier and a stateful storage tier. The virtualization and the elasticity that are so visible in the IaaS model are almost completely invisible here. But they're a big part of the picture behind the scenes. One of the selling propositions of this model is its *automatic* elasticity in the face of capacity requirement changes.

The App Engine programming languages are Python and Java. App Engine isn't suitable for general-purpose computing. It works best for web applications and relies on the assumption of a request-reply structure, which assumes long periods of no CPU utilization (such as, human think time). Consequently, Google can and does severely ration CPU time for each request.

App Engine's automatic scaling and high-availability mechanisms, and the proprietary MegaStore data storage (built on BigTable) available to App Engine applications, all rely on these constraints. But if your application fits within those constraints, there is probably no faster and cheaper way to build an application that scales automatically and runs on the largest cloud on the planet.

APP ENGINE DEVELOPMENT ENVIRONMENT

The App Engine development environment consists of these elements:

- *Sandbox*—Applications run in a secure environment that provides limited access to the underlying operating system. These limitations allow App Engine to distribute web requests for the application across multiple servers and to start and stop servers to meet traffic demands. The sandbox isolates your application in its own secure, reliable environment independent of the hardware, operating system, and physical location of the web server.

- *Java runtime environment*—You can develop applications for the Java 6 runtime environment using common Java web development tools and API standards. An app interacts with the environment using the Java Servlet standard and can use common web application technologies, such as JavaServer Pages (JSPs). Apps access most App Engine services using Java standard APIs. The environment includes the Java SE Runtime Environment (JRE) 6 platform and libraries. The restrictions of the sandbox environment are implemented in the JVM. An app can use any JVM bytecode or library feature, as long as it doesn't exceed the sandbox restrictions.
- *Python runtime environment*—You can develop applications using the Python 2.5 programming language and run it on a Python interpreter. App Engine includes APIs and tools for Python web application development, including a data-modeling API, a web application framework, and tools for managing and accessing the App's data. The Python environment includes the Python standard library within the limitations of the sandbox environment. Application code written for the Python environment must be written exclusively in Python. The Python environment provides APIs for the datastore, Google Accounts, URL fetch, and email services.
- *Datastore*—App Engine provides a distributed data-storage service that features a query engine and transactions. This distributed datastore scales with the application's needs automatically. As we discussed previously regarding cloud databases, the App Engine datastore isn't like a traditional relational database. Data objects, or *entities*, have a kind and a set of properties. Queries can retrieve entities of a given kind filtered and sorted by the values of the properties. Property values can be of any of the supported property value types. Datastore entities are *schemaless*. The application code enforces and provides the structure of data entities. The Java JDO/JPA interfaces and the Python datastore interface include features for applying and enforcing this structure.

App Engine is free under these daily thresholds: 6.5 hours of CPU time, and 1 GB of data transferred in and out of the application. Beyond this, outgoing bandwidth costs $0.12 per GB, incoming bandwidth costs $0.10 per GB, CPU time is $0.10 per hour, stored data is $0.15 per GB per month, and recipients emailed are $0.0001 per recipient.

2.2.4 Ruby on Rails in a cloud: Platform as a Service

Ruby on Rails (RoR) is an open-source web application framework for the Ruby programming language. It's intended to be used with an agile development methodology, often used by web developers due to its suitability for short, client-driven projects. Similar to Google's App Engine, RoR applications are limited to request-response architecture web applications.

OPEN SOURCE SOFTWARE Computer software available in source code form for which the source code and certain other rights normally reserved for copyright holders are provided under a software license that permits users to study, change, and improve the software. Some consider open source a philosophy; others consider it a pragmatic methodology. Before the term *open source* became widely adopted, developers and producers used a variety of phrases to describe the concept; *open source* gained hold with the rise of the internet and the attendant need for massive retooling of the computing source code. Open-source software is most often developed in a public, collaborative manner.

The Ruby language was designed to combine Smalltalk's conceptual elegance, Python's ease of use and learning, and Perl's pragmatism. Many teams experience 10X faster development of web applications using RoR. But many have reported significant challenges getting RoR to scale massively, which probably has to do with architecture and design choices made in the application as opposed to something endemic to RoR itself.

Many small companies jumped in early to offer RoR stacks that run on top of Amazon's EC2, including Heroku, Aptana, EngineYard, and others.

2.2.5 Salesforce.com's Force.com: Platform as a Service

Salesforce.com is the most successful SaaS application used in the enterprise. It's a customer-relationship-management (CRM) application that has run strictly as a cloud application since 1999.

Force.com is the company's PaaS capability, where developers use the Apex programming language to create add-on applications that integrate into the main Salesforce application and are hosted on Salesforce.com's cloud infrastructure.

Google and Salesforce have created an integration between App Engine and Force.com such that applications can be built using either environment and still access the stored repository of corporate data on Salesforce's site.

Force.com also runs an exchange called AppExchange; it's a directory of applications built for Salesforce by third-party developers, which users can purchase and add to their Salesforce environment. More than 800 applications are available from over 450 ISVs.

The Force.com list price is $5.00 per login with a maximum of five logins per user per month. According to the company's website, "Force.com cloud pricing is for occasional-use, widely-deployed apps and is available for platform use only and not for CRM applications."

Our last classification is a strange one—strange because it's not about a different type of application environment but instead is about a different ownership structure. So-called Datacenter as a Service is about private companies creating a *private* cloud just for their use.

2.2.6 Private clouds: Datacenter as a Service (DaaS)

Private cloud (also *internal cloud* and *corporate cloud*) is a term for a computing architecture that provides hosted services to a limited number of people behind a firewall. The same advances in virtualization, automation, and distributed computing that enable the cloud for Amazon, Microsoft, and Google have allowed corporate network and data-center administrators to effectively become service providers that meet the needs of their customers within the corporation.

The concept of a private cloud is designed to appeal to an organization that needs or wants more control over its data than it can get by using a third-party hosted service, such as Amazon's EC2 or S3. Internal IT providers that build private clouds have to make fundamental changes in their operations so they behave and provide benefits (on a smaller scale) similar to those of cloud computing providers. In addition to economic gains through higher utilization and a pay-for-what-you-use model, an enterprise, to enable the private cloud model, implements changes in operations which, at the very least, makes an organization better equipped to shift to or overflow to a public cloud when appropriate.

The contrarian view

Here's the rub: some say private clouds are expensive data centers with a fancy name. Pundits predict that within the next year or so, we'll have seen the rise and fall of this concept. Whereas everyone agrees that virtualization, service-oriented architectures, and open standards are all great things for companies operating a data center to consider, critics argue that all this talk about private clouds is a distraction from the real news: the vast majority of companies shouldn't need to worry about operating any sort of data center anymore, cloud-like or not.

SOME CONCERNS FOR THOSE THINKING ABOUT PRIVATE CLOUDS

If you're considering implementing a private cloud, keep the following in mind:

- *Private clouds are small scale.* There's a reason why most innovative cloud computing providers have their roots in powering consumer web technology—that's where the numbers are. Few corporate data centers will see anything close to the type of volume seen by these vendors. And volume drives cost savings through the huge economies of scale we've discussed.
- *Legacy applications don't cloudify easily.* Legacy applications moved to a private cloud will see marginal improvements at best. You can achieve only so much without re-architecting these applications to a cloud infrastructure.
- *On-premises doesn't mean more secure.* The biggest drivers toward private clouds have been fear, uncertainty, and doubt about security. For many, it feels more secure to have your data behind your firewall in a data center that you control. But unless your company spends more money and energy thinking about security than Amazon, Google, and Salesforce, that is a fallacy.
- *Do what you do best.* There's no simple set of tricks that an operator of a data center can borrow from Amazon or Google. These companies make their living operating the world's largest data centers. They're constantly optimizing how

they operate based on real-time performance feedback from millions of transactions. You can try to learn from and emulate them, but your rate of innovation will never be the same—private clouds will always be many steps behind the public clouds.

AMAZON VIRTUAL PRIVATE CLOUD

Amazon Virtual Private Cloud (Amazon VPC) is a secure and seamless bridge between a company's existing IT infrastructure and the AWS cloud. Although it isn't a private cloud as we defined it, this approach offers corporations a hybrid model merging aspects of their data center with Amazon's cloud.

Amazon VPC enables an enterprise to connect its existing infrastructure to a set of isolated AWS compute resources via a Virtual Private Network (VPN) connection and to extend existing management capabilities, such as security services, firewalls, and intrusion-detection systems, to include AWS resources. You'll learn much more about cloud security, private clouds, and VPC in chapter 4.

Until now, we've explored the technological underpinnings of clouds to understand how they work, and we've applied that knowledge to a few of the most prominent clouds in a variety of categories to understand how they compare and contrast. You're now informed enough to ask this question: What type of cloud do I need?

2.3 Matching cloud providers to your needs

We've looked under the hood of a lot of different cloud types, their APIs, their other service offerings, and the technologies that underpin them. Which of them is appropriate for you? How can you prevent lock-in when you do make the choice? We'll try to answer those questions by going back through the major cloud providers and applying a framework of decision criteria by which you can evaluate each one for your projects.

2.3.1 Amazon web services IaaS cloud

Summarizing what we've explored so far, AWS is a flexible, lower-level offering (closer to hardware), which means you have more possibilities. And in general, it will be higher performing at the cost of "everything is left up to you," including how and when to scale, move or replicate your data, and more.

Amazon EC2 runs the platform you provide, supports all major programming languages, and offers a set of industry-standard services (getting more standard as standards groups and the open source Eucalyptus seeks to formalize theirs as the standard cloud API). But Amazon, being an IaaS, requires much more work, which means a longer time-to-market for your applications.

Use AWS if you

- Want to use third-party open-source software
- Have existing code
- Want to transfer a web app to your own machine/servers later
- Port code to another language

- Want complete control
- Need to stress/load test an app (for example, load up 1,000 instances)

And as for avoiding lock-in, Amazon EC2 is good because Amazon-compatible services can and will be easily provided by other companies as well as an open-source initiative. The leader always gets to set the standards. EC2 is practically the closest to zero lock-in of any choice you can make today.

2.3.2 Microsoft Windows Azure IaaS and PaaS cloud

Azure is intermediate between application frameworks, such as App Engine, and hardware virtual machines, such as EC2. Microsoft is trying to make the transition from desktop (data center) to its cloud as seamless as possible. The company suggests that you can build and test an application locally and then deploy to its cloud. But Microsoft does admit that all UI and any data-extraction logic must be rewritten to deal with low-bandwidth internet connections. Note that we said *its cloud*. In that sense, Microsoft is similar to App Engine and Force.com in terms of locking you in to its cloud, run by the company.

Use Windows Azure if you

- Already use the .NET and SQL Server portions of the Microsoft stack
- Have existing code developed to those Microsoft APIs
- Have teams that normally develop in Visual Studio using C#
- Want to blend development from desk top to cloud
- Have no issue with lock-in to Microsoft

As for lock-in, Windows Azure isn't looking as bad as Google App Engine. Although it will still be hosted exclusively by Microsoft, it may be possible for other companies to come up with (almost) compatible cloud service because core pieces of Windows Azure are based on the well-known SQL Server, IIS, and .NET framework stacks.

2.3.3 Google App Engine PaaS cloud

Google App Engine is a tightly controlled environment—a decision Google made to enable automatic scaling of application threads as well as the datastore. The environment supports only Python and Java, and no installation of any open source software is possible.

Use App Engine if you

- Have no preexisting code
- Are building request-response web apps or mashups
- Consider time-to-market the most important thing
- Aren't doing anything fancy (installing software)
- Aren't worried about lock-in to Google

App Engine is high on the lock-in scale. It's hard to imagine any compatible products from any other company for a long time, if ever. It's proprietary, and Google doesn't plan to release its technology. Automatic scale and time-to-market have many advantages, but almost complete lock-in will most likely be the price you pay for those benefits.

2.3.4 Ruby on Rails PaaS cloud

Ruby is slightly more computationally expensive than other languages, but having easy resource expansion available can cure a lot of the "what if I get mentioned on Oprah?" scares that business people experience. Rails is a particularly good match for cloud computing because of its shared-nothing architecture. This means you can generate new instances of an application, and they will begin to run. And developers love Ruby because of their much higher productivity. Many small companies are now providing RoR clouds (many layered on top of Amazon).

Use Ruby on Rails if you

- Are building request-response web apps with existing Ruby expertise
- Consider time-to-market critical
- Aren't doing anything fancy (installing software)
- Don't care about lock-in

Lock-in isn't a big concern with RoR because, as we've said, there are many choices of RoR vendors and probably more to come.

2.3.5 Force.com PaaS cloud

Force.com is an extension of the SaaS service Salesforce.com. Many companies have been using Salesforce for a long time. They have rich, sophisticated databases of sales contacts, history of sales cycles, information about their products, and a lot of other sales-process related information. This information forms the crown jewels of any company's sales team, and companies want many applications that aren't built into Salesforce.com. For this reason, Salesforce.com created a framework using many of the same back-end services used by the company's main SaaS application, operating on the same back-end data, and made it accessible and programmable to end users. Force.com is ideal for building applications that tie into your existing Salesforce.com databases, such as sales contacts, the internal sales team, your products, and so on.

Use Force.com if you

- Are already a customer of Salesforce.com's SaaS customer-resource-management product
- Have a requirement for a simple mashup style of web application
- Are willing to use Force.com's specialized programming language
- Don't care about lock-in

We didn't include a section about when to use private cloud because it's a much more complex discussion. We'll deal with the subject in chapter 4.

2.4 Summary

This chapter built on your understanding from chapter 1 of the types of clouds and the reasons—technical and economic—for this step in the evolution of computing. We've focused on how the cloud works by looking under the hood and examining the technological underpinnings. Cloud providers are making phenomenal economies of scale. Their costs keep getting lower while their specialized expertise in operating these massive data centers gets better.

This chapter examined some of the core enabling technologies of cloud computing. First and foremost is virtualization, which even most corporate data centers have embraced as a way to increase server utilization and thereby lower costs. Because a cloud is a virtualized server environment where you can quickly crate new instances of machines or applications and then control them over the network, both automation and network access are also vital in cloud computing. An API to create, operate, expand elastically, and destroy instances is also required. Trends seem to be leading in the direction of Amazon's API becoming an industry standard. We looked at cloud storage, focusing on Amazon's S3 API as an example.

You saw how relational databases don't scale because they have to be shared. This has led to the emergence of new key-value types of databases as the norm in the cloud. One of the biggest benefits of moving to the cloud is the ability to scale almost infinitely as application demand grows. You learned how this elasticity works with the example of the calls required in Amazon EC2 to create an automatically scaled, load-balanced EC2 application. We compared Amazon's SimpleDB to Google's BigTable.

This chapter also compared and contrasted the major flavors of cloud computing. Amazon EC2 is the most prominent example of IaaS. Microsoft Azure is mostly IaaS as well but has many PaaS offerings. Google is the most prominent of the PaaS with its App Engine. The plethora of Ruby on Rails offerings (such as Force.com from Salesforce) are highly specialized types of platforms.

Somewhat tongue in cheek, we expanded the taxonomy of cloud terms to include data center as a service and examined the concept of private clouds to see if this is something that will stick or is just a fad. This understanding of the cloud classifications should help you avoid the all-too-common "apples to oranges" comparisons between, say, an IaaS and a PaaS cloud. You're now in a position to distinguish between them. More important, you're in a better position to make an informed decision about what's best for you, depending on what you're trying to do.

You've learned that a major driver behind this IT evolution isn't technology but economics. Consequently, we'll spend the next chapter looking closely at the business case for cloud computing.

ENTERPRISE DEVELOPMENT

THE Cloud AT Your Service

Jothy Rosenberg • Arthur Mateos

Practically unlimited storage, instant scalability, zero-downtime upgrades, low start-up costs, plus pay-only-for-what-you-use without sacrificing security or performance are all benefits of cloud computing. But how do you make it work in your enterprise? What should you move to the cloud? How? And when?

The Cloud at Your Service answers these questions and more. Written for IT pros at all levels, this book finds the sweet spot between rapidly changing details and hand-waving hype. It shows you practical ways to work with current services like Amazon's EC2 and S3. You'll also learn the pros and cons of private clouds, the truth about cloud data security, and how to use the cloud for high scale applications.

What's Inside

- How to build scalable and reliable applications
- The state of the art in technology, vendors, practices
- What to keep in-house and what to offload
- How to migrate existing IT to the cloud
- How to build secure applications and data centers

A PhD in computer science, Jothy Rosenberg is a former Duke professor, author of three previous books, and serial entrepreneur involved in the cloud movement from its infancy. A technology entrepreneur with a PhD in nuclear physics from MIT, Arthur Mateos has brought to market pioneering SaaS products built on the cloud.

For online access to the authors and a free ebook for owners of this book, go to manning.com/CloudatYourService

Free ebook
SEE INSERT

"Cuts through the complexity to just what's needed."
—From the Foreword by Anne Thomas Manes

"A definitive source."
—Orhan Alkan
Sun Microsystems

"Approachable coverage of a key emerging technology."
—Chad Davis
Author of *Struts 2 in Action*

"Removes 'cloudiness' from the cloud."
—Shawn Henry
CloudSwitch, Inc.

"Refreshing... without fluff."
—Kunal Mittal
Sony Pictures Entertainment

ISBN 13: 978-1-935182-52-8
ISBN 10: 1-935182-52-8
52999
9 781935 182528

MANNING $29.99 / Can $34.99 [INCLUDING eBOOK]