Chapter 1

Introduction to Cloud Computing and AWS

The cloud is where much of the serious technology innovation and growth happens these days, and Amazon Web Services (AWS), more than any other, is the platform of choice for business and institutional workloads. If you want to be successful as an AWS solutions architect, you'll first need to understand what the cloud really is and how Amazon's end of it works.

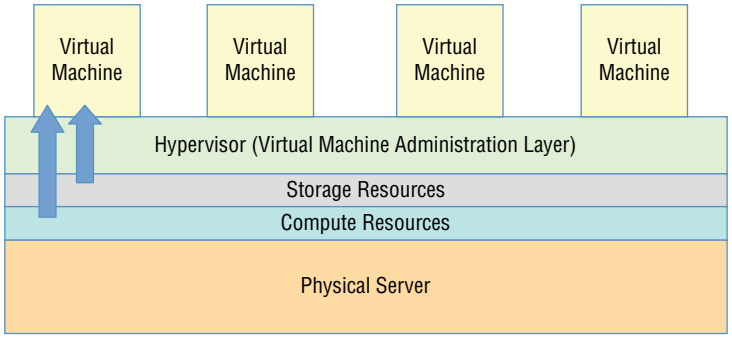
TO MAKE SURE YOU'VE GOT THE BIG PICTURE, THIS CHAPTER WILL EXPLORE THE BASICS:

- ✓ **What makes cloud computing different from other applications and client-server models**
- ✓ **How the AWS platform provides secure and flexible virtual networked environments for your resources**
- ✓ **How AWS provides such a high level of service reliability**
- ✓ **How to access and manage your AWS-based resources**
- ✓ **Where you can go for documentation and help with your AWS deployments**

Cloud Computing and Virtualization

The technology that lies at the core of all cloud operations is virtualization. As illustrated in Figure 1.1, virtualization lets you divide the hardware resources of a single physical server into smaller units. That physical server could therefore host multiple virtual machines (VMs) running their own complete operating systems, each with its own memory, storage, and network access.

FIGURE 1.1 A virtual machine host



Virtualization's flexibility makes it possible to provision a virtual server in a matter of seconds, run it for exactly the time your project requires, and then shut it down. The resources released will become instantly available to other workloads. The usage density you can achieve lets you squeeze the greatest value from your hardware and makes it easy to generate experimental and sandboxed environments.

Cloud Computing Architecture

Major cloud providers like AWS have enormous server farms where hundreds of thousands of servers and disk drives are maintained along with the network cabling necessary to connect them. A well-built virtualized environment could provide a virtual server using storage, memory, compute cycles, and network bandwidth collected from the most efficient mix of available sources it can find.

A cloud computing platform offers on-demand, self-service access to pooled compute resources where your usage is metered and billed according to the volume you consume. Cloud computing systems allow for precise billing models, sometimes involving fractions of a penny for an hour of consumption.

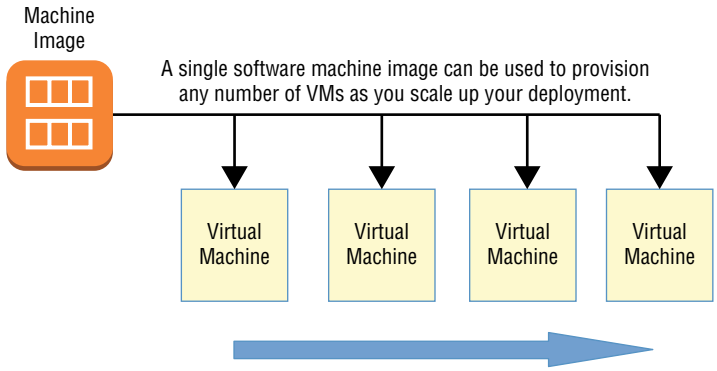
Cloud Computing Optimization

The cloud is a great choice for so many serious workloads because it's scalable, elastic, and, often, a lot cheaper than traditional alternatives. Effective deployment provisioning will require some insight into those three features.

Scalability

A scalable infrastructure can efficiently meet unexpected increases in demand for your application by automatically adding resources. As Figure 1.2 shows, this most often means dynamically increasing the number of virtual machines (or *instances* as AWS calls them) you've got running.

FIGURE 1.2 Copies of a machine image are added to new VMs as they're launched.



AWS offers its autoscaling service through which you define a machine image that can be instantly and automatically replicated and launched into multiple instances to meet demand.

Elasticity

The principle of elasticity covers some of the same ground as scalability—both address how the system manages changing demand. However, though the images used in a scalable environment let you ramp up capacity to meet rising demand, an elastic infrastructure will automatically *reduce* capacity when demand drops. This makes it possible to control costs, since you'll run resources only when they're needed.

Cost Management

Besides the ability to control expenses by closely managing the resources you use, cloud computing transitions your IT spending from a capital expenditure (capex) framework into something closer to operational expenditure (opex).

In practical terms, this means you no longer have to spend $10,000 up front for every new server you deploy—along with associated electricity, cooling, security, and rack space costs. Instead, you're billed much smaller incremental amounts for as long as your application runs.

That doesn't necessarily mean your long-term cloud-based opex costs will always be less than you'd pay over the lifetime of a comparable data center deployment. But it does mean you won't have to expose yourself to risky speculation about your long-term needs. If, sometime in the future, changing demand calls for new hardware, AWS will be able to deliver it within a minute or two.

To help you understand the full implications of cloud compute spending, AWS provides a free Total Cost of Ownership (TCO) Calculator at `aws.amazon.com/tco-calculator`. This calculator helps you perform proper "apples-to-apples" comparisons between your current data center costs and what an identical operation would cost you on AWS.

The AWS Cloud

Keeping up with the steady stream of new services showing up on the AWS Console can be frustrating. But as a solutions architect, your main focus should be on the core service categories. This section briefly summarizes each of the core categories (as shown in Table 1.1) and then does the same for key individual services. You'll learn much more about all of these (and more) services through the rest of the book, but it's worth focusing on these short definitions, because they lie at the foundation of everything else you're going to learn.

TABLE 1.1 AWS service categories

Category	Function
Compute	Services replicating the traditional role of local physical servers for the cloud, offering advanced configurations including autoscaling, load balancing, and even serverless architectures (a method for delivering server functionality with a very small footprint)
Networking	Application connectivity, access control, and enhanced remote connections
Storage	Various kinds of storage platforms designed to fit a range of both immediate accessibility and long-term backup needs

Category	Function
Database	Managed data solutions for use cases requiring multiple data formats: relational, NoSQL, or caching
Application management	Monitoring, auditing, and configuring AWS account services and running resources
Security and identity	Services for managing authentication and authorization, data and connection encryption, and integration with third-party authentication management systems

Table 1.2 describes the functions of some core AWS services, organized by category.

TABLE 1.2 Core AWS services (by category)

Category	Service	Function
Compute	Elastic Compute Cloud (EC2)	EC2 server instances provide virtual versions of the servers you would run in your local data center. EC2 instances can be provisioned with the CPU, memory, storage, and network interface profile to meet any application need, from a simple web server to one part of a cluster of instances providing an integrated multi-tiered fleet architecture. Since EC2 instances are virtual, they're resource-efficient and deploy nearly instantly.
	Lambda	Serverless application architectures like the one provided by Amazon's Lambda service allow you to provide responsive public-facing services without the need for a server that's actually running 24/7. Instead, network events (like consumer requests) can trigger the execution of a predefined code-based operation. When the operation (which can currently run for as long as 15 minutes) is complete, the Lambda event ends, and all resources automatically shut down.
	Auto Scaling	Copies of running EC2 instances can be defined as image templates and automatically launched (or *scaled up*) when client demand can't be met by existing instances. As demand drops, unused instances can be terminated (or *scaled down*).
	Elastic Load Balancing	Incoming network traffic can be directed between multiple web servers to ensure that a single web server isn't overwhelmed while other servers are underused or that traffic isn't directed to failed servers.

TABLE 1.2 Core AWS services (by category) *(continued)*

Category	Service	Function
	Elastic Beanstalk	Beanstalk is a managed service that abstracts the provisioning of AWS compute and networking infrastructure. You are required to do nothing more than push your application code, and Beanstalk automatically launches and manages all the necessary services in the background.
Networking	Virtual Private Cloud (VPC)	VPCs are highly configurable networking environments designed to host your EC2 (and RDS) instances. You use VPC-based tools to secure and, if desired, isolate your instances by closely controlling inbound and outbound network access.
	Direct Connect	By purchasing fast and secure network connections to AWS through a third-party provider, you can use Direct Connect to establish an enhanced direct tunnel between your local data center or office and your AWS-based VPCs.
	Route 53	Route 53 is the AWS DNS service that lets you manage domain registration, record administration, routing protocols, and health checks, which are all fully integrated with the rest of your AWS resources
	CloudFront	CloudFront is Amazon's distributed global content delivery network (CDN). When properly configured, a CloudFront distribution can store cached versions of your site's content at edge locations around the world so that they can be delivered to customers on request with the greatest efficiency and lowest latency.
Storage	Simple Storage Service (S3)	S3 offers highly versatile, reliable, and inexpensive object storage that's great for data storage and backups. It's also commonly used as part of larger AWS production processes, including through the storage of script, template, and log files.
	S3 Glacier	A good choice for when you need large data archives stored cheaply over the long term and can live with retrieval delays measuring in the hours. Glacier's lifecycle management is closely integrated with S3.
	Elastic Block Store (EBS)	EBS provides the persistent virtual storage drives that host the operating systems and working data of an EC2 instance. They're meant to mimic the function of the storage drives and partitions attached to physical servers.

Category	Service	Function
	Storage Gateway	Storage Gateway is a hybrid storage system that exposes AWS cloud storage as a local, on-premises appliance. Storage Gateway can be a great tool for migration and data backup and as part of disaster recovery operations.
Database	Relational Database Service (RDS)	RDS is a managed service that builds you a stable, secure, and reliable database instance. You can run a variety of SQL database engines on RDS, including MySQL, Microsoft SQL Server, Oracle, and Amazon's own Aurora.
	DynamoDB	DynamoDB can be used for fast, flexible, highly scalable, and managed nonrelational (NoSQL) database workloads.
Application management	CloudWatch	CloudWatch can be set to monitor process performance and resource utilization and, when preset thresholds are met, either send you a message or trigger an automated response.
	CloudFormation	This service enables you to use template files to define full and complex AWS deployments. The ability to script your use of any AWS resources makes it easier to automate, standardizing and speeding up the application launch process.
	CloudTrail	CloudTrail collects records of all your account's API events. This history is useful for account auditing and troubleshooting purposes.
	Config	The Config service is designed to help you with change management and compliance for your AWS account. You first define a desired configuration state, and Config evaluates any future states against that ideal. When a configuration change pushes too far from the ideal baseline, you'll be notified.
Security and identity	Identity and Access Management (IAM)	You use IAM to administrate user and programmatic access and authentication to your AWS account. Through the use of users, groups, roles, and policies, you can control exactly who and what can access and/or work with any of your AWS resources.
	Key Management Service (KMS)	KMS is a managed service that allows you to administrate the creation and use of encryption keys to secure data used by and for any of your AWS resources.

TABLE 1.2 Core AWS services (by category) *(continued)*

Category	Service	Function
	Directory Service	For AWS environments that need to manage identities and relationships, Directory Service can integrate AWS resources with identity providers like Amazon Cognito and Microsoft AD domains.
Application integration	Simple Notification Service (SNS)	SNS is a notification tool that can automate the publishing of alert *topics* to other services (to an SQS Queue or to trigger a Lambda function, for instance), to mobile devices, or to recipients using email or SMS.
	Simple Workflow (SWF)	SWF lets you coordinate a series of tasks that must be performed using a range of AWS services or even non-digital (meaning, human) events.
	Simple Queue Service (SQS)	SQS allows for event-driven messaging within distributed systems that can decouple while coordinating the discrete steps of a larger process. The data contained in your SQS messages will be reliably delivered, adding to the fault-tolerant qualities of an application.
	API Gateway	This service enables you to create and manage secure and reliable APIs for your AWS-based applications.

AWS Platform Architecture

AWS maintains data centers for its physical servers around the world. Because the centers are so widely distributed, you can reduce your own services' network transfer latency by hosting your workloads geographically close to your users. It can also help you manage compliance with regulations requiring you to keep data within a particular legal jurisdiction.

Data centers exist within AWS regions, of which there are currently 21—not including private U.S. government AWS GovCloud regions—although this number is constantly growing. It's important to always be conscious of the region you have selected when you launch new AWS resources; pricing and service availability can vary from one to the next. Table 1.3 shows a list of all 21 (nongovernment) regions along with each region's name and core endpoint addresses. Note that accessing and authenticating to the two Chinese regions requires unique protocols.

TABLE 1.3 A list of publicly accessible AWS regions

Region name	Region	Endpoint
US East (Ohio)	us-east-2	us-east-2.amazonaws.com
US West (N. California)	us-west-1	us-west-1.amazonaws.com
US West (Oregon)	us-west-2	us-west-2.amazonaws.com
Asia Pacific (Hong Kong)	ap-east-1	ap-east-1.amazonaws.com
Asia Pacific (Mumbai)	ap-south-1	ap-south-1.amazonaws.com
Asia Pacific (Seoul)	ap-northeast-2	ap-northeast-2.amazonaws.com
Asia Pacific (Osaka-Local)	ap-northeast-3	ap-northeast-3.amazonaws.com
Asia Pacific (Singapore)	ap-southeast-1	ap-southeast-1.amazonaws.com
Asia Pacific (Sydney)	ap-southeast-2	ap-southeast-2.amazonaws.com
Asia Pacific (Tokyo)	ap-northeast-1	ap-northeast-1.amazonaws.com
Canada (Central)	ca-central-1	ca-central-1.amazonaws.com
China (Beijing)	cn-north-1	cn-north-1.amazonaws.com.cn
China (Ningxia)	cn-northwest-1	cn-northwest-1.amazonaws.com.cn
EU (Frankfurt)	eu-central-1	eu-central-1.amazonaws.com
EU (Ireland)	eu-west-1	eu-west-1.amazonaws.com
EU (London)	eu-west-2	eu-west-2.amazonaws.com
EU (Paris)	eu-west-3	eu-west-3.amazonaws.com
EU (Stockholm)	eu-north-1	eu-north-1.amazonaws.com
Middle East (Bahrain)	me-south-1	me-south-1.amazon.aws.com

Endpoint addresses are used to access your AWS resources remotely from within application code or scripts. Prefixes like `ec2`, `apigateway`, or `cloudformation` are often added to the endpoints to specify a particular AWS service. Such an address might look like this: `cloudformation.us-east-2.amazonaws.com`. You can see a complete list of endpoint addresses and their prefixes at `docs.aws.amazon.com/general/latest/gr/rande.html`.

Because low-latency access is so important, certain AWS services are offered from designated edge network locations. These services include Amazon CloudFront, Amazon Route 53, AWS Firewall Manager, AWS Shield, and AWS WAF. For a complete and up-to-date list of available locations, see `aws.amazon.com/about-aws/global-infrastructure/regional-product-services`.

Physical AWS data centers are exposed within your AWS account as availability zones. There might be half a dozen availability zones within a region, like `us-east-1a` and `us-east-1b`, each consisting of one or more data centers.

You organize your resources from a region within one or more virtual private clouds (VPCs). A VPC is effectively a network address space within which you can create network subnets and associate them with availability zones. When configured properly, this architecture can provide effective resource isolation and durable replication.

AWS Reliability and Compliance

AWS has a lot of the basic regulatory, legal, and security groundwork covered before you even launch your first service.

AWS has invested significant planning and funds into resources and expertise relating to infrastructure administration. Its heavily protected and secretive data centers, layers of redundancy, and carefully developed best-practice protocols would be difficult or even impossible for a regular enterprise to replicate.

Where applicable, resources on the AWS platform are compliant with dozens of national and international standards, frameworks, and certifications, including ISO 9001, FedRAMP, NIST, and GDPR. (See `aws.amazon.com/compliance/programs` for more information.)

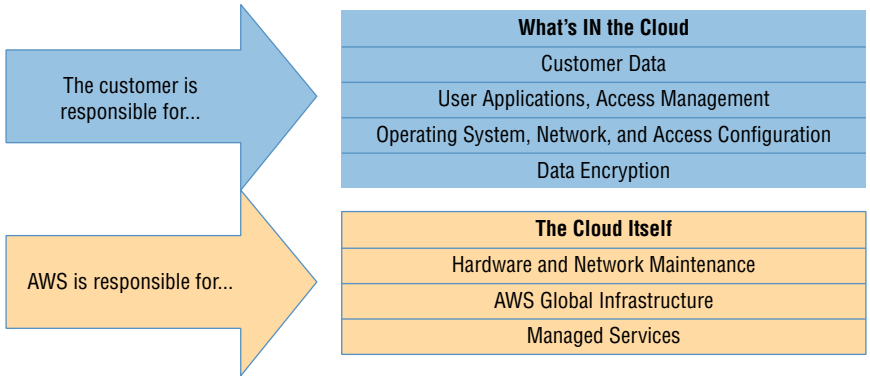
The AWS Shared Responsibility Model

Of course, those guarantees cover only the underlying AWS platform. The way you decide to use AWS resources is your business—and therefore your responsibility. So, it's important to be familiar with the AWS Shared Responsibility Model.

AWS guarantees the secure and uninterrupted operation of its "cloud." That means its physical servers, storage devices, networking infrastructure, and managed services. AWS customers, as illustrated in Figure 1.3, are responsible for whatever happens *within* that

cloud. This covers the security and operation of installed operating systems, client-side data, the movement of data across networks, end-user authentication and access, and customer data.

FIGURE 1.3 The AWS Shared Responsibility Model



The AWS Service Level Agreement

By "guarantee," AWS doesn't mean that service disruptions or security breaches will *never* occur. Drives may stop spinning, major electricity systems may fail, and natural disasters may happen. But when something does go wrong, AWS will provide service credits to reimburse customers for their *direct* losses whenever uptimes fall below a defined threshold. Of course, that won't help you recover customer confidence or lost business.

The exact percentage of the guarantee will differ according to service. The service level agreement (SLA) rate for AWS EC2, for instance, is set to 99.99 percent—meaning that you can expect your EC2 instances, ECS containers, and EBS storage devices to be available for all but around four minutes of each month.

The important thing to remember is that it's not *if* things will fail but *when*. Build your applications to be geographically dispersed and fault tolerant so that when things do break, your users will barely notice.

Working with AWS

Whatever AWS services you choose to run, you'll need a way to manage them all. The browser-based management console is an excellent way to introduce yourself to a service's features and learn how it will perform in the real world. There are few AWS administration tasks that you can't do from the console, which provides plenty of useful visualizations and

helpful documentation. But as you become more familiar with the way things work, and especially as your AWS deployments become more complex, you'll probably find yourself getting more of your serious work done away from the console.

The AWS CLI

The AWS Command Line Interface (CLI) lets you run complex AWS operations from your local command line. Once you get the hang of how it works, you'll discover that it can make things much simpler and more efficient.

As an example, suppose you need to launch a half-dozen EC2 instances to make up a microservices environment. Each instance is meant to play a separate role and therefore will require a subtly different provisioning process. Clicking through window after window to launch the instances from the console can quickly become tedious and time-consuming, especially if you find yourself repeating the task every few days. But the whole process can alternatively be incorporated into a simple script that you can run from your local terminal shell or PowerShell interface using the AWS CLI.

Installing and configuring the AWS CLI on Linux, Windows, or macOS machines is not hard at all, but the details might change depending on your platform. For the most up-to-date instructions, see `docs.aws.amazon.com/cli/latest/userguide/installing.html`.

AWS SDKs

If you want to incorporate access to your AWS resources into your application code, you'll need to use an AWS software development kit (SDK) for the language you're working with. AWS currently offers SDKs for nine languages, including Java, .NET, and Python, and a number of mobile SDKs that include Android and iOS. There are also toolkits available for IntelliJ, Visual Studio, and Visual Studio Team Services (VSTS).

You can see a full overview of AWS developer tools at `aws.amazon.com/tools`.

Technical Support and Online Resources

Things won't always go smoothly for you—on AWS just as everywhere else in your life. Sooner or later, you'll need some kind of technical or account support. There's a variety of types of support, and you should understand what's available.

One of the first things you'll be asked to decide when you create a new AWS account is which support plan you'd like. Your business needs and budget will determine the way you answer this question.

Support Plans

The Basic plan is free with every account and gives you access to customer service, along with documentation, white papers, and the support forum. Customer service covers billing and account support issues.

The Developer plan starts at $29/month and adds access for one account holder to a Cloud Support associate along with limited general guidance and "system impaired" response.

For $100/month (and up), the Business plan will deliver faster guaranteed response times to unlimited users for help with "impaired" systems, personal guidance and troubleshooting, and a support API.

Finally, Enterprise support plans cover all of the other features, plus direct access to AWS solutions architects for operational and design reviews, your own technical account manager, and something called a support concierge. For complex, mission-critical deployments, those benefits can make a big difference. But they'll cost you at least $15,000 each month.

You can read more about AWS support plans at `aws.amazon.com/premiumsupport/compare-plans`.

Other Support Resources

There's plenty of self-serve support available outside of the official support plans:

- AWS community help forums are open to anyone with a valid AWS account (`forums.aws.amazon.com`).
- Extensive and well-maintained AWS documentation is available at `aws.amazon.com/documentation`.
- The AWS Well-Architected page (`aws.amazon.com/architecture/well-architected`) is a hub that links to some valuable white papers and documentation addressing best practices for cloud deployment design.
- There are also plenty of third-party companies offering commercial support for your AWS deployments.

Summary

Cloud computing is built on the ability to efficiently divide physical resources into smaller but flexible virtual units. Those units can be "rented" by businesses on a pay-as-you-go basis and used to satisfy just about any networked application and/or workflow's needs in an affordable, scalable, and elastic way.

Amazon Web Services provides reliable and secure resources that are replicated and globally distributed across a growing number of regions and availability zones. AWS infrastructure is designed to be compliant with best-practice and regulatory standards—although the Shared Responsibility Model leaves you in charge of what you place *within* the cloud.

The growing family of AWS services covers just about any digital needs you can imagine, with core services addressing compute, networking, database, storage, security, and application management and integration needs.

Chapter 2

Amazon Elastic Compute Cloud and Amazon Elastic Block Store

THE AWS CERTIFIED SOLUTIONS ARCHITECT ASSOCIATE EXAM OBJECTIVES COVERED IN THIS CHAPTER MAY INCLUDE, BUT ARE NOT LIMITED TO, THE FOLLOWING:

✓ **Domain 1: Design Resilient Architectures**

- 1.1 Design a multi-tier architecture solution
- 1.2 Design highly available and/or fault-tolerant architectures
- 1.4 Choose appropriate resilient storage

✓ **Domain 2: Design High-Performing Architectures**

- 2.1 Identify elastic and scalable compute solutions for a workload
- 2.2 Select high-performing and scalable storage solutions for a workload

✓ **Domain 3: Design Secure Applications and Architectures**

- 3.1 Design secure access to AWS resources

✓ **Domain 4: Design Cost-Optimized Architectures**

- 4.1 Identify cost-effective storage solutions
- 4.2 Identify cost-effective compute and database services
- 4.3 Design cost-optimized network architectures

Introduction

The ultimate focus of a traditional data center/server room is its precious servers. But, to make those servers useful, you'll need to add racks, power supplies, cabling, switches, firewalls, and cooling.

AWS's Elastic Compute Cloud (EC2) is designed to replicate the data center/server room experience as closely as possible. At the center of it all is the EC2 virtual server, known as an *instance*. But, like the local server room I just described, EC2 provides a range of tools meant to support and enhance your instance's operations.

This chapter will explore the tools and practices used to fully leverage the power of the EC2 ecosystem, including the following:

- Provisioning an EC2 instance with the right hardware resources for your project
- Configuring the right base operating system for your application needs
- Building a secure and effective network environment for your instance
- Adding scripts to run as the instance boots to support (or start) your application
- Choosing the best EC2 pricing model for your needs
- Understanding how to manage and leverage the EC2 instance lifecycle
- Choosing the right storage drive type for your needs
- Securing your EC2 resources using key pairs, security groups, network access lists, and Identity and Access Management (IAM) roles
- Scaling the number of instances up and down to meet changing demand using Auto Scaling
- Accessing your instance as an administrator or end-user client

EC2 Instances

An EC2 instance may only be a virtualized and abstracted subset of a physical server, but it behaves just like the real thing. It will have access to storage, memory, and a network interface, and its primary storage drive will come with a fresh and clean operating system running.

It's up to you to decide what kind of hardware resources you want your instance to have, what operating system and software stack you'd like it to run, and, ultimately, how much you'll pay for it. Let's see how all that works.

Provisioning Your Instance

You configure your instance's operating system and software stack, hardware specs (the CPU power, memory, primary storage, and network performance), and environment before launching it. The OS is defined by the Amazon Machine Image (AMI) you choose, and the hardware follows the instance type.

EC2 Amazon Machine Images

An AMI is really just a template document that contains information telling EC2 what OS and application software to include on the root data volume of the instance it's about to launch. There are four kinds of AMIs:

Amazon Quick Start AMIs Amazon Quick Start images appear at the top of the list in the console when you start the process of launching a new instance. The Quick Start AMIs are popular choices and include various releases of Linux or Windows Server OSs and some specialty images for performing common operations (like deep learning and database). These AMIs are up-to-date and officially supported.

AWS Marketplace AMIs AMIs from the AWS Marketplace are official, production-ready images provided and supported by industry vendors like SAP and Cisco.

Community AMIs More than 100,000 images are available as community AMIs. Many of these images are AMIs created and maintained by independent vendors and are usually built to meet a specific need. This is a good catalog to search if you're planning an application built on a custom combination of software resources.

Private AMIs You can also store images created from your own instance deployments as private AMIs. Why would you want to do that? You might, for instance, want the ability to scale up the number of instances you've got running to meet growing demand. Having a reliable, tested, and patched instance image as an AMI makes incorporating autoscaling easy. You can also share images as AMIs or import VMs from your local infrastructure (by way of AWS S3) using the AWS VM Import/Export tool.

A particular AMI will be available in only a single region—although there will often be images with identical functionality in all regions. Keep this in mind as you plan your deployments: invoking the ID of an AMI in one region while working from within a different region will fail.

An Important Note About Billing

Besides the normal charges for running an EC2 instance, your AWS account might also be billed hourly amounts or license fees for the use of the AMI software itself. Although vendors make every effort to clearly display the charges for their AMIs, it's your responsibility to accept and honor those charges.

Instance Types

AWS allocates hardware resources to your instances according to the instance type—or hardware profile—you select. The particular workload you're planning for your instance will determine the type you choose. The idea is to balance cost against your need for compute power, memory, and storage space. Ideally, you'll find a type that offers exactly the amount of each to satisfy both your application and budget.

Should your needs change over time, you can easily move to a different instance type by stopping your instance, editing its instance type, and starting it back up again.

As listed in Table 2.1, there are currently more than 75 instance types organized into five instance families, although AWS frequently updates their selection. You can view the most recent collection at `aws.amazon.com/ec2/instance-types`.

TABLE 2.1 EC2 instance type family and their top-level designations

Instance type family	Types
General purpose	A1, T3, T3a, T2, M6g, M5, M5a, M5n, M4
Compute optimized	C5, C5n, C4
Memory optimized	R5, R5a, R5n, X1e, X1, High Memory, z1d
Accelerated computing	P3, P2, Inf1, G4, G3, F1
Storage optimized	I3, I3en, D2, H1

General Purpose The General Purpose family includes T3, T2, M5, and M4 types, which all aim to provide a balance of compute, memory, and network resources. T2 types, for instance, range from the t2.nano with one virtual CPU (vCPU0) and half a gigabyte of memory all the way up to the t2.2xlarge with its eight vCPUs and 32 GB of memory. Because it's eligible as part of the Free Tier, the t2.micro is often a good choice for experimenting. But there's nothing stopping you from using it for light-use websites and various development-related services.

T2s are burstable, which means you can accumulate CPU credits when your instance is underutilized that can be applied during high-demand periods in the form of higher CPU performance.

M5 and M4 instances are recommended for many small and midsized data-centric operations. Unlike T2, which requires EBS virtual volumes for storage, some M* instances come with their own instance storage drives that are actually physically attached to the host server. M5 types range from m5.large (2 vCPUs and 8 GB of memory) to the monstrous m5d.metal (96 vCPUs and 384 GB of memory).

Compute Optimized For more demanding web servers and high-end machine learning workloads, you'll choose from the Compute Optimized family that includes C5 and C4 types. C5 machines—currently available from the c5.large to the c5d.24xlarge—give you as much as 3.5 GHz of processor speed and strong network bandwidth.

Memory Optimized Memory Optimized instances work well for intensive database, data analysis, and caching operations. The X1e, X1, and R4 types are available with as much as 3.9 terabytes of dynamic random-access memory (DRAM)-based memory and low-latency solid-state drive (SSD) storage volumes attached.

Accelerated Computing You can achieve higher-performing general-purpose graphics processing unit (GPGPU) performance from the P3, P2, G3, and F1 types within the Accelerated Computing group. These instances make use of various generations of high-end NVIDIA GPUs or, in the case of the F1 instances, an Xilinx Virtex UltraScale+ field-programmable gate array (FPGA—if you don't know what that is, then you probably don't need it). Accelerated Computing instances are recommended for demanding workloads such as 3D visualizations and rendering, financial analysis, and computational fluid dynamics.

Storage Optimized The H1, I3, and D2 types currently make up the Storage Optimized family that have large, low-latency instance storage volumes (in the case of I3en, up to 60 TB of slower **hard disk drive** [HDD] storage). These instances work well with distributed filesystems and heavyweight data processing applications.

The specification details and instance type names will frequently change as AWS continues to leverage new technologies to support its customers' growing computing demands. But it's important to be at least familiar with the instance type families and the naming conventions AWS uses to identify them.

Configuring an Environment for Your Instance

Deciding where your EC2 instance will live is as important as choosing a performance configuration. Here, there are three primary details to get right: geographic region, virtual private cloud (VPC), and tenancy model.

AWS Regions

As you learned earlier, AWS servers are housed in data centers around the world and organized by geographical region. You'll generally want to launch an EC2 instance in the region that's physically closest to the majority of your customers or, if you're working with data that's subject to legal restrictions, within a jurisdiction that meets your compliance needs.

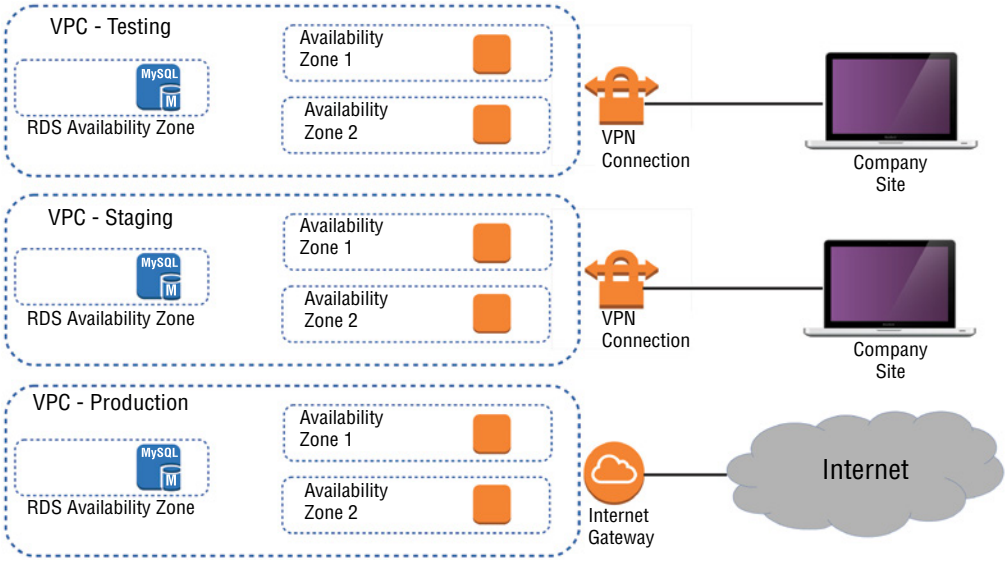
EC2 resources can be managed only when you're "located within" their region. You set the active region in the console through the drop-down menu at the top of the page and through default configuration values in the AWS CLI or your SDK. You can update your CLI configuration by running `aws configure`.

Bear in mind that the costs and even functionality of services and features might vary between regions. It's always a good idea to consult the most up-to-date official documentation.

VPCs

Virtual private clouds (VPCs) are easy-to-use AWS network organizers and great tools for organizing your infrastructure. Because it's so easy to isolate the instances in one VPC from whatever else you have running, you might want to create a new VPC for each one of your projects or project stages. For example, you might have one VPC for early application development, another for beta testing, and a third for production (see Figure 2.1).

FIGURE 2.1 A multi-VPC infrastructure for a development environment



Adding a simple VPC that doesn't incorporate a network address translation (NAT) gateway (`docs.aws.amazon.com/AmazonVPC/latest/UserGuide/vpc-nat-gateway.html`) or VPN access (`docs.aws.amazon.com/vpn/latest/s2svpn/VPC_VPN.html`) won't cost you anything. You'll learn much more about all this in Chapter 4, "Amazon Virtual Private Cloud."

Tenancy

When launching an EC2 instance, you'll have the opportunity to choose a tenancy model. The default setting is *shared tenancy*, where your instance will run as a virtual machine on a physical server that's concurrently hosting other instances. Those other instances might well be owned and operated by other AWS customers, although the possibility of any kind of insecure interaction between instances is remote.

To meet special regulatory requirements, your organization's instances might need an extra level of isolation. The Dedicated Instance option ensures that your instance will run on a dedicated physical server. This means that it won't be sharing the server with resources

owned by a different customer account. The Dedicated Host option allows you to actually identify and control the physical server you've been assigned to meet more restrictive licensing or regulatory requirements.

Naturally, dedicated instances and dedicated hosts will cost you more than instances using shared tenancy.

Exercise 2.1 will guide you through the launch of a simple EC2 Linux instance.

EXERCISE 2.1

Launch an EC2 Linux Instance and Log in Using SSH

1. From the EC2 Dashboard, click to launch a new instance and select a Linux AMI and instance type. Remember, the t2.micro is Free Tier–eligible if your AWS account is still within its first year.
2. Explore the Configure Instance Details, Add Storage, and Add Tags pages—although the default settings should work fine.
3. On the Configure Security Group page, make sure there's a rule permitting incoming SSH (port 22) traffic. It should be there by default.
4. Before letting you launch the instance, AWS will require you to select—or create—a key pair. Follow the instructions.
5. Once the instance is launched, you can return to the Instances Dashboard to wait a minute or two until everything is running properly.
6. Click the Actions pull-down and then the Connect item for instructions on how to connect to the instance from your local machine. Then connect and take a look at your virtual cloud server.

In Exercise 2.2, you'll see how changing an instance's type works.

EXERCISE 2.2

Assess the Free Capacity of a Running Instance and Change Its Instance Type

1. With an instance running, open the Instances Dashboard in the EC2 console. Select the instance you're interested in and click the Monitoring tab in the bottom half of the screen. That's a good place to see what percentage of compute and network resources you've been using over the past hours or weeks.

   Now pretend that your instance is nearly maxed out and change the instance type as follows.

EXERCISE 2.2 *(continued)*

2. Stop the instance. (Remember, your public IP address might be different when you start up again.)
3. From the Actions drop-down, click Instance Settings and then Change Instance Type. Select a new type.
4. Restart the instance and confirm that it's running properly.

Configuring Instance Behavior

You can optionally tell EC2 to execute commands on your instance as it boots by pointing to user data in your instance configuration (this is sometimes known as *bootstrapping*). Whether you specify the data during the console configuration process or by using the `--user-data` value with the AWS CLI, you can have script files bring your instance to any desired state.

User data can consist of a few simple commands to install a web server and populate its web root, or it can be a sophisticated script setting up the instance as a working node within a Puppet Enterprise–driven platform.

Placement Groups

By default AWS will attempt to spread your instances across their infrastructure to create a profile that will be optimal for most use cases. But the specific demands of your operation might require a different setup. EC2 placement groups give you the power to define nonstandard profiles to better meet your needs. There are, at this time, three kinds of placement groups:

- *Cluster* groups launch each associated instance into a single availability zone within close physical proximity to each other. This provides low-latency network interconnectivity and can be useful for high-performance computing (HPC) applications, for instance.
- *Spread* groups separate instances physically across distinct hardware racks and even availability zones to reduce the risk of failure-related data or service loss. Such a setup can be valuable when you're running hosts that can't tolerate multiple concurrent failures. If you're familiar with VMware's Distributed Resource Scheduler (DRS), this is similar to that.
- *Partition* groups let you associate some instances with each other, placing them in a single "partition." But the instances within that single partition can be kept physically separated from instances within other partitions. This differs from spread groups where no two instances will ever share physical hardware.

Instance Pricing

You can purchase the use of EC2 instances through one of three models.

For always-on deployments that you expect to run for less than 12 months, you'll normally pay for each hour your instance is running through the *on-demand model*. On-demand is the most flexible way to consume EC2 resources since you're able to closely control how much you pay by stopping and starting your instances according to your need. But, per hour, it's also the most expensive.

If you're planning to keep the lights burning 24/7 for more than a year, then you'll enjoy a significant discount by purchasing a *reserve instance*—generally over a term commitment of between one and three years. You can pay up front for the entire term of a reserve instance or, for incrementally higher rates, either partially up front and the rest in monthly charges or entirely through monthly charges. Table 2.2 gives you a sense of how costs can change between models. These estimates assume a Linux platform, all up-front payments, and default tenancy. Actual costs may vary over time and between regions.

TABLE 2.2 Pricing estimates comparing on-demand with reserve costs

Instance type	Pricing model	Cost/hour	Cost/year
t2.micro	On-demand	$0.0116	$102.00
t2.micro	Reserve (three-year term)		$38.33
g3.4xlarge	On-demand	$1.14	$9986.40
g3.4xlarge	Reserve (three-year term)		$4429.66

For workloads that can withstand unexpected disruption (like computation-intensive genome research applications), purchasing instances on Amazon's spot market can save you a lot of money. The idea is that you enter a maximum dollar-value bid for an instance type running in a particular region. The next time an instance in that region becomes available at a per-hour rate that's equal to or below your bid, it'll be launched using the AMI and launch template you specified. Once up, the instance will keep running either until you stop it—when your workload completes, for example—or until the instance's per-hour rate rises above your maximum bid. You'll learn more about the spot market and reserve instances in Chapter 13, "The Cost Optimization Pillar."

It will often make sense to combine multiple models within a single application infrastructure. An online store might, for instance, purchase one or two reserve instances to cover its normal customer demand but also allow autoscaling to automatically launch on-demand instances during periods of unusually high demand.

Use Exercise 2.3 to dive deeper into EC2 pricing.

EXERCISE 2.3

Assess Which Pricing Model Will Best Meet the Needs of a Deployment

Imagine that your application will need to run two always-on f1.2xlarge instances (which come with instance storage and won't require any EBS volumes). To meet seasonal demand, you can expect to require as many as four more instances for a total of 100 hours through the course of a single year. How should you pay for this deployment?

Bonus: Calculate your total estimated monthly and annual costs.

Instance Lifecycle

The state of a running EC2 instance can be managed in a number of ways. Terminating the instance will shut it down and cause its resources to be reallocated to the general AWS pool.

Terminating an instance will, in most cases, destroy all data kept on the primary storage. The exception to this would be an Elastic Block Store (EBS) volume that has been set to persist after its instance is terminated.

If your instance won't be needed for some time but you don't want to terminate it, you can save money by simply stopping it and then restarting it when it's needed again. The data on an EBS volume will in this case not be lost, although that would not be true for an instance volume.

Later in this chapter, you'll learn about both EBS and instance store volumes and the ways they work with EC2 instances.

You should be aware that a stopped instance that had been using a nonpersistent public IP address will most likely be assigned a different address when it's restarted. If you need a predictable IP address that can survive restarts, allocate an elastic IP address and associate it with your instance.

You can edit or change an instance's security group (which we'll discuss a bit later in this chapter) to update access policies at any time—even while an instance is running. You can also change its instance type to increase or decrease its compute, memory, and storage capacity (just try doing *that* on a physical server). You will need to stop the instance, change the type, and then restart it.

Resource Tags

The more resources you deploy on your AWS account, the harder it can be to properly keep track of things. Having constantly changing numbers of EC2 instances—along with accompanying storage volumes, security groups, and elastic IP addresses—all spread across two or three VPCs can get complicated.

The best way to keep a lid on the chaos is to find a way to quickly identify each resource you've got running by its purpose and its relationships to other resources. The best way to do that is by establishing a consistent naming convention and applying it to tags.

AWS resource tags can be used to label everything you'll ever touch across your AWS account—they're certainly not restricted to just EC2. Tags have a key and, optionally, an associated value. So, for example, you could assign a tag with the key `production-server` to each element of a production deployment. Server instances could, in addition, have a value of `server1`, `server2`, and so on. A related security group could have the same `production-server` key but `security-group1` for its value. Table 2.3 illustrates how that convention might play out over a larger deployment group.

TABLE 2.3 A sample key/value tagging convention

Key	Value
`production-server`	`server1`
`production-server`	`server2`
`production-server`	`security-group1`
`staging-server`	`server1`
`staging-server`	`server2`
`staging-server`	`security-group1`
`test-server`	`server1`
`test-server`	`security-group1`

Applied properly, tags can improve the visibility of your resources, making it much easier to manage them effectively, audit and control costs and billing trends, and avoid costly errors.

Service Limits

By default, each AWS account has limits to the number of instances of a particular service you're able to launch. Sometimes those limits apply to a single region within an account, and others are global. As examples, you're allowed only five VPCs per region and 5,000 Secure Shell (SSH) key pairs across your account. If necessary, you can ask AWS to raise your ceiling for a particular service.

You can find up-to-date details regarding the limits of all AWS services at `docs.aws.amazon.com/general/latest/gr/aws_service_limits.html`.

EC2 Storage Volumes

Storage drives (or *volumes* as they're described in AWS documentation) are for the most part virtualized spaces carved out of larger physical drives. To the OS running on your instance, though, all AWS volumes will present themselves exactly as though they were normal physical drives. But there's actually more than one kind of AWS volume, and it's important to understand how each type works.

Elastic Block Store Volumes

You can attach as many Elastic Block Store (EBS) volumes to your instance as you like (although one volume can be attached to no more than a single instance at a time) and use them just as you would hard drives, flash drives, or USB drives with your physical server. And as with physical drives, the type of EBS volume you choose will have an impact on both performance and cost.

The AWS SLA guarantees the reliability of the data you store on its EBS volumes (promising at least 99.99 percent availability), so you don't have to worry about failure. When an EBS drive does fail, its data has already been duplicated and will probably be brought back online before anyone notices a problem. So, practically, the only thing that should concern you is how quickly and efficiently you can access your data.

There are currently four EBS volume types, two using SSD technologies and two using the older spinning hard drives. The performance of each volume type is measured in maximum IOPS/volume (where IOPS means input/output operations per second).

EBS-Provisioned IOPS SSD

If your applications will require intense rates of I/O operations, then you should consider provisioned IOPS, which provides a maximum IOPS/volume of 64,000 and a maximum throughput/volume of 1,000 MB/s. Provisioned IOPS—which in some contexts is referred to as EBS Optimized—can cost $0.125/GB/month in addition to $0.065/provisioned IOPS.

EBS General-Purpose SSD

For most regular server workloads that, ideally, deliver low-latency performance, general-purpose SSDs will work well. You'll get a maximum of 16,000 IOPS/volume, and it will cost you $0.10/GB/month. For reference, a general-purpose SSD used as a typical 8 GB boot drive for a Linux instance would, at current rates, cost you $9.60/year.

Throughput-Optimized HDD

Throughput-optimized HDD volumes can provide reduced costs with acceptable performance where you're looking for throughput-intensive workloads, including log processing and big data operations. These volumes can deliver only 500 IOPS/volume but with a 500 MB/s maximum throughput/volume, and they'll cost you only $0.045/GB/month.

Cold HDD

When you're working with larger volumes of data that require only infrequent access, a 250 IOPS/volume type might meet your needs for only $0.025/GB/month.

Table 2.4 lets you compare the basic specifications and estimated costs of those types.

TABLE 2.4 Sample costs for each of the four EBS storage volume types

	EBS-provisioned IOPS SSD	EBS general-purpose SSD	Throughput-optimized HDD	Cold HDD
Volume size	4 GB–16 TB	1 GB–16 TB	500 GB–16 TB	500 GB–16 TB
Max IOPS/volume	64,000	16,000	500	250
Max throughput/volume (MB/s)	1,000	250	500	250
Price (/month)	$0.125/GB + $0.065/prov IOPS	$0.10/GB	$0.045/GB	$0.025/GB

EBS Volume Features

All EBS volumes can be copied by creating a snapshot. Existing snapshots can be used to generate other volumes that can be shared and/or attached to other instances or converted to images from which AMIs can be made. You can also generate an AMI image directly from a running instance-attached EBS volume—although, to be sure no data is lost, you should shut down the instance first.

EBS volumes can be encrypted to protect their data while at rest or as it's sent back and forth to the EC2 host instance. EBS can manage the encryption keys automatically behind the scenes or use keys that you provide through the AWS Key Management Service (KMS).

Exercise 2.4 will walk you through launching a new instance based on an existing snapshot image.

EXERCISE 2.4

Create and Launch an AMI Based on an Existing Instance Storage Volume

1. If necessary, launch an instance and make at least some token change to the root volume. This could be something as simple as typing `touch test.txt` on a Linux instance to create an empty file.
2. Create an image from the instance's volume (you'll access the dialog through the Actions pull-down menu in the Instance's Dashboard).
3. Launch an instance from the console and select the new AMI from the My AMIs tab.
4. Log into the instance and confirm that your previous change has persisted.

Instance Store Volumes

Unlike EBS volumes, instance store volumes are ephemeral. This means that when the instances they're attached to are shut down, their data is permanently lost. So, why would you want to keep your data on an instance store volume more than on EBS?

- Instance store volumes are SSDs that are physically attached to the server hosting your instance and are connected via a fast NVMe (Non-Volatile Memory Express) interface.
- The use of instance store volumes is included in the price of the instance itself.
- Instance store volumes work especially well for deployment models where instances are launched to fill short-term roles (as part of autoscaling groups, for instance), import data from external sources, and are, effectively, disposable.

Whether one or more instance store volumes are available for your instance will depend on the instance type you choose. This is an important consideration to take into account when planning your deployment.

Even with all the benefits of EBS and instance storage, it's worth noting that there will be cases where you're much better off keeping large data sets outside of EC2 altogether. For many use cases, Amazon's S3 service can be a dramatically less expensive way to store files or even databases that are nevertheless instantly available for compute operations.

You'll learn more about this in Chapter 3, "AWS Storage."

The bottom line is that EBS volumes are likely to be the right choice for instances whose data needs to persist beyond a reboot and for working with custom or off-the-shelf AMIs. Instance store volumes are, where available, useful for operations requiring low-latency access to large amounts of data that needn't survive a system failure or reboot. And non-EC2 storage can work well when you don't need fantastic read/write speeds, but you wish to enjoy the flexibility and cost savings of S3.

Accessing Your EC2 Instance

Like all networked devices, EC2 instances are identified by unique IP addresses. All instances are assigned at least one private IPv4 address that, by default, will fall within one of the blocks shown in Table 2.5.

TABLE 2.5 The three IP address ranges used by private networks

From	To
10.0.0.0	10.255.255.255
172.16.0.0	172.31.255.255
192.168.0.0	192.168.255.255

Out of the box, you'll only be able to connect to your instance from within its subnet, and the instance will have no direct connection to the Internet.

If your instance configuration calls for multiple network interfaces (to connect to otherwise unreachable resources), you can create and then attach one or more virtual elastic network interfaces to your instance. Each of these interfaces must be connected to an existing subnet and security group. You can optionally assign a static IP address within the subnet range.

Of course, an instance can also be assigned a public IP through which full Internet access is possible. As noted in the instance lifecycle discussion, the default public IP assigned to your instance is ephemeral and probably won't survive a reboot. Therefore, you'll usually want to allocate a permanent elastic IP for long-term deployments. As long as it's attached to a running instance, there's no charge for elastic IPs.

I'll talk about accessing an instance as an administrator a bit later within the context of security. But there's a lot you can learn about a running EC2 instance—including the IP addresses you'll need to connect—through the instance metadata system. Running the following `curl` command from the command line while logged into the instance will return a list of the kinds of data that are available:

```
$ curl http://169.254.169.254/latest/meta-data/
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
```

```
hostname
instance-action
instance-id
instance-type
local-hostname
local-ipv4
mac
metrics/
network/
placement/
profile
public-hostname
public-ipv4
public-keys/
reservation-id
security-groups
```

You'll use the 169.254.169.254 IP for the command no matter what your instance public or private IPs happen to be.

Entries ending with a trailing slash (/) contain further sublevels of information that can also be displayed by `curl`. Adding a data type to that `curl` command will then return the information you're after. This example displays the name of the security groups used by the instance:

```
$ curl http://169.254.169.254/latest/meta-data/security-groups launch-wizard-1
```

Securing Your EC2 Instance

You are responsible for configuring appropriate and effective access controls to protect your EC2 instances from unauthorized use. Broadly speaking, AWS provides four tools to help you with this task: security groups, Identity and Access Management (IAM) roles, network address translation (NAT) instances, and key pairs.

Security Groups

An EC2 security group plays the role of a firewall. By default, a security group will deny all incoming traffic while permitting all outgoing traffic. You define group behavior by setting policy rules that will either block or allow specified traffic types. From that point on, any

data packet coming into or leaving the perimeter will be measured against those rules and processed accordingly.

Traffic is assessed by examining its source and destination, the network port it's targeting, and the protocol it's set to use. A TCP packet sent to the SSH port 22 could, for example, only be allowed access to a particular instance if its source IP address matches the local public IP used by computers in your office. This lets you open up SSH access on your instance without having to worry about anyone from outside your company getting in.

Using security groups, you can easily create sophisticated rule sets to finely manage access to your services. You could, say, open up a website to the whole world while blocking access to your backend servers for everyone besides members of your team.

If necessary, you can update your security group rules and/or apply them to multiple instances.

Security groups control traffic at the instance level. However, AWS also provides you with network access control lists (NACLs) that are associated with entire subnets rather than individual instances. Chapter 4, "Amazon Virtual Private Cloud," discusses both security groups and NACLs.

IAM Roles

You can also control access to AWS resources—including EC2 instances—through the use of IAM roles. You define an IAM role by giving it permissions to perform actions on specified services or resources within your AWS account. When a particular role is assigned to a user or resource, they'll gain access to whichever resources were included in the role policies.

Using roles, you can give a limited number of entities (other resources or users) exclusive access to resources like your EC2 instances. But you can also assign an IAM role *to* an EC2 instance so that processes running within it can access the external tools—like an RDS database instance—it needs to do its work.

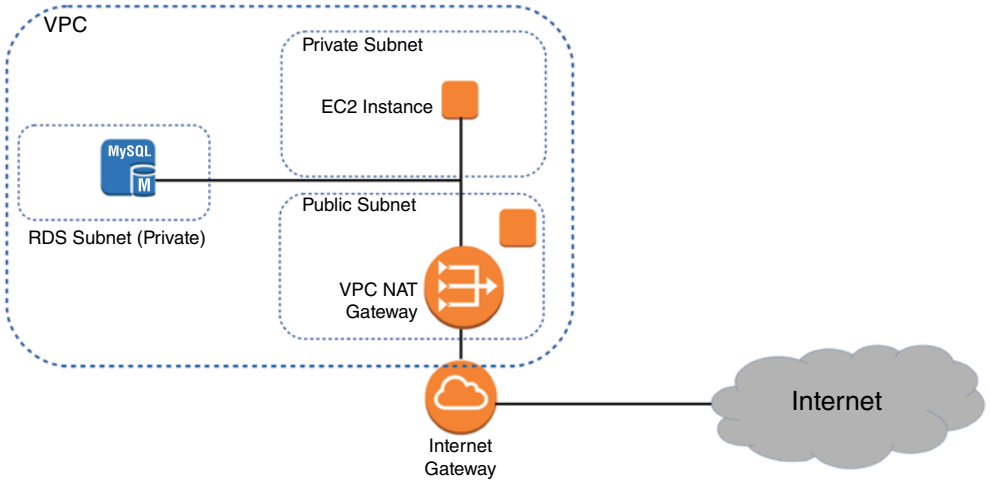
You'll learn more about IAM in Chapter 6, "Authentication and Authorization—AWS Identity and Access Management."

NAT Devices

Sometimes you'll need to configure an EC2 instance without a public IP address to limit its exposure to the network. Naturally, that means it won't have any Internet connectivity. But that can present a problem because you'll probably still need to give it Internet access so that it can receive security patches and software updates.

One solution is to use network address translation (NAT) to give your private instance access *to* the Internet without allowing access to it *from* the Internet. AWS gives you two ways to do that: a NAT instance and a NAT gateway (see Figure 2.2). They'll both do the job, but since a NAT gateway is a managed service, it doesn't require that you manually launch and maintain an instance. Both approaches will incur monthly charges.

FIGURE 2.2 A NAT gateway providing network access to resources in private subnets



NAT will be discussed at greater length in Chapter 4.

Key Pairs

As any professional administrator will know, remote login sessions on your running instances should never be initiated over unencrypted plain-text connections. To ensure properly secured sessions, you'll need to generate a key pair, save the public key to your EC2 server, and save its private half to your local machine. If you're working with a Windows AMI, you'll use the private key file to retrieve the password you'll need to authenticate into your instance. For a Linux AMI, the private key will allow you to open an SSH session.

Each key pair that AWS generates for you will remain installed within its original region and available for use with newly launched instances until you delete it. You *should* delete the AWS copy in the event your public key is lost or exposed. Just be careful before you mess with your keys—your access to an instance might depend on it.

EC2 Auto Scaling

The *EC2 Auto Scaling* service offers a way to both avoid application failure and recover from it when it happens. Auto Scaling works by provisioning and starting on your behalf a specified number of EC2 instances. It can dynamically add more instances to keep up with increased demand. And when an instance fails or gets terminated, Auto Scaling will automatically replace it.

EC2 Auto Scaling uses either a *launch configuration* or a *launch template* to automatically configure the instances that it launches. Both perform the same basic function of defining the basic configuration parameters of the instance as well as what scripts (if any) run on it at launch time. Launch configurations have been around longer and are more familiar to you if you've been using AWS for a while. You're also more likely to encounter them if you're going into an existing AWS environment. Launch templates are newer and are what AWS now recommends. You'll learn about both, but which you use is up to you.

Launch Configurations

When you create an instance manually, you have to specify many configuration parameters, including an AMI, instance type, SSH key pair, security group, instance profile, block device mapping, whether it's EBS optimized, placement tenancy, and user data, such as custom scripts to install and configure your application. A launch configuration is essentially a named document that contains the same information you'd provide when manually provisioning an instance.

You can create a launch configuration from an existing EC2 instance. Auto Scaling will copy the settings from the instance for you, but you can customize them as needed. You can also create a launch configuration from scratch.

Launch configurations are for use only with EC2 Auto Scaling, meaning you can't manually launch an instance using a launch configuration. Also, once you create a launch configuration, you can't modify it. If you want to change any of the settings, you have to create an entirely new launch configuration.

Launch Templates

Launch templates are similar to launch configurations in that you can specify the same settings. But the uses for launch templates are more versatile. You can use a launch template with Auto Scaling, of course, but you can also use it for spinning up one-off EC2 instances or even creating a spot fleet.

Launch templates are also versioned, allowing you to change them after creation. Any time you need to make changes to a launch template, you create a new version of it. AWS keeps all versions, and you can then flip back and forth between versions as needed. This makes it easier to track your launch template changes over time. Complete Exercise 2.5 to create your own launch template.

If you have an existing launch configuration, you can copy it to a launch template using the AWS web console. There's no need to create launch templates from scratch!

EXERCISE 2.5

Create a Launch Template

In this exercise, you'll create a launch template that installs and configures a simple web server. You'll then use the launch template to manually create an instance.

1. In the EC2 Dashboard, click Launch Templates.
2. Click the Create Launch Template button.
3. Give the launch template a name such as **MyTemplate**.
4. Click the Search For AMI link to locate one of the Ubuntu Server LTS AMIs (make sure the AMI you choose uses the 64-bit x86 architecture and not 64-bit ARM).
5. For Instance Type, select t2.micro.
6. Under Security Groups, select a security group that allows inbound HTTP access. Create a new security group if necessary.
7. Expand the Advanced Details section and enter the following in the User Data field:

```
#!/bin/bash
apt-get update
apt-get install -y apache2
echo "Welcome to my website" > index.html
cp index.html /var/www/html
```

8. Click the Create Launch Template button.
9. Click the Launch Instance From This Template link.
10. Under Source Template Version, select 1 (Default).
11. Click the Launch Instance From Template button.
12. After the instance boots, browse to its public IP address. You should see a web page that says "Welcome to my website."
13. Terminate the instance when you're done with it.

Auto Scaling Groups

An *Auto Scaling group* is a group of EC2 instances that Auto Scaling manages. When creating an Auto Scaling group, you must first specify either the launch configuration or launch template you created. When you create an Auto Scaling group, you must specify how many running instances you want Auto Scaling to provision and maintain using the launch configuration or template you created. You must specify the minimum and maximum size of the Auto Scaling group. You may also optionally set the desired number of instances you want Auto Scaling to provision and maintain.

Minimum Auto Scaling will ensure the number of healthy instances never goes below the minimum. If you set this to 0, Auto Scaling will not spawn any instances and will terminate any running instances in the group.

Maximum Auto Scaling will make sure the number of healthy instances never exceeds this amount. This might seem strange but remember that you might have budget limitations and need to be protected from unexpected (and unaffordable) usage demands.

Desired Capacity The desired capacity is an optional setting that must lie within the minimum and maximum values. If you don't specify a desired capacity, Auto Scaling will launch the number of instances as the minimum value. If you specify a desired capacity, Auto Scaling will add or terminate instances to stay at the desired capacity. For example, if you set the minimum to 1, the maximum to 10, and the desired capacity to 4, then Auto Scaling will create four instances. If one of those instances gets terminated—for example, because of human action or a host crash—Auto Scaling will replace it to maintain the desired capacity setting of 4. In the web console, desired capacity is also called the *group size*.

Specifying an Application Load Balancer Target Group

If you want to use an application load balancer (ALB) to distribute traffic to instances in your Auto Scaling group, just plug in the name of the ALB target group when creating the Auto Scaling group. Whenever Auto Scaling creates a new instance, it will automatically add it to the ALB target group.

Health Checks Against Application Instances

When you create an Auto Scaling group, Auto Scaling will strive to maintain the minimum number of instances, or the desired number if you've specified it. If an instance becomes unhealthy, Auto Scaling will terminate and replace it.

By default, Auto Scaling determines an instance's health based on EC2 health checks. Chapter 7, "CloudTrail, CloudWatch, and AWS Config," covers how EC2 automatically performs system and instance status checks. These checks monitor for instance problems such as memory exhaustion, filesystem corruption, or an incorrect network or startup configuration, as well as for system problems that require AWS involvement to repair. Although these checks can catch a variety of instance and host-related problems, they won't necessarily catch application-specific problems.

If you're using an application load balancer to route traffic to your instances, you can configure health checks for the load balancer's target group. Target group health checks can check for HTTP response codes from 200 to 499. You can then configure your Auto Scaling group to use the results of these health checks to determine whether an instance is healthy.

If an instance fails the ALB health check, it will route traffic away from the failed instance, ensuring that users don't reach it. At the same time, Auto Scaling will remove the

instance, create a replacement, and add the new instance to the load balancer's target group. The load balancer will then route traffic to the new instance.

A good design practice is to have a few recovery actions that work for a variety of circumstances. An instance may crash due to an out-of-memory condition, a bug, a deleted file, or an isolated network failure, but simply terminating and replacing the instance using Auto Scaling resolves all these cases. There's no need to come up with a separate recovery action for each cause when simply re-creating the instance solves them all.

Auto Scaling Options

Once you create an Auto Scaling group, you can leave it be and it will continue to maintain the minimum or desired number of instances indefinitely. However, maintaining the current number of instances is just one option. Auto Scaling provides several other options to scale out the number of instances to meet demand.

Manual Scaling

If you change the minimum, desired, or maximum values at any time after creating the group, Auto Scaling will immediately adjust. For example, if you have the desired capacity value set to 2 and change it to 4, Auto Scaling will launch two more instances. If you have four instances and set the desired capacity value to 2, Auto Scaling will terminate two instances. Think of the desired capacity as a thermostat.

Dynamic Scaling Policies

Most AWS-managed resources are elastic—that is, they automatically scale to accommodate increased load. Some examples include S3, load balancers, Internet gateways, and NAT gateways. Regardless of how much traffic you throw at them, AWS is responsible for ensuring that they remain available while continuing to perform well. But when it comes to your EC2 instances, you're responsible for ensuring that they're powerful and plentiful enough to meet demand.

Running out of instance resources—be it CPU utilization, memory, or disk space—will almost always result in the failure of whatever you're running on it. To ensure that your instances never become overburdened, dynamic scaling policies automatically provision more instances *before* they hit that point. Auto Scaling generates the following aggregate metrics for all instances within the group:

- Aggregate CPU utilization
- Average request count per target

- Average network bytes in
- Average network bytes out

You're not limited to using just these native metrics. You can also use metric filters to extract metrics from CloudWatch logs and use those. As an example, your application may generate logs that indicate how long it takes to complete a process. If the process takes too long, you could have Auto Scaling spin up new instances.

Dynamic scaling policies work by monitoring a CloudWatch alarm and scaling out—by increasing the desired capacity—when the alarm is breaching. You can choose from three dynamic scaling policies: simple, step, and target tracking.

Simple Scaling Policies

With a *simple scaling policy*, whenever the metric rises above the threshold, Auto Scaling simply increases the desired capacity. How much it increases the desired capacity, however, depends on which of the following *adjustment types* you choose:

ChangeInCapacity Increases the capacity by a specified amount. For instance, you could start with a desired capacity value of 4 and then have Auto Scaling increase the value by 2 when the load increases.

ExactCapacity Sets the capacity to a specific value, regardless of the current value. For example, suppose the desired capacity value is 4. You create a policy to change the value to 6 when the load increases.

PercentChangeInCapacity Increases the capacity by a percentage of the current amount. If the current desired capacity value is 4 and you specify the percent change in capacity as 50 percent, then Auto Scaling will bump the desired capacity value to 6.

For example, suppose you have four instances and create a simple scaling policy that specifies a PercentChangeInCapacity adjustment of 50 percent. When the monitored alarm triggers, Auto Scaling will increment the desired capacity by 2, which will in turn add two instances to the Auto Scaling group, for a total of six.

After Auto Scaling completes the adjustment, it waits a *cooldown period* before executing the policy again, even if the alarm is still breaching. The default cooldown period is 300 seconds, but you can set it as high as you want or as low as 0—effectively disabling it. Note that if an instance is unhealthy, Auto Scaling will not wait for the cooldown period before replacing the unhealthy instance.

Referring to the preceding example, suppose that after the scaling adjustment completes and the cooldown period expires, the monitored alarm drops below the threshold. At this point, the desired capacity value is 6. If the alarm triggers again, the simple scaling action will execute again and add three more instances. Keep in mind that Auto Scaling will never increase the desired capacity beyond the group's maximum setting.

Step Scaling Policies

If the demand on your application is rapidly increasing, a simple scaling policy may not add enough instances quickly enough. Using a *step scaling policy*, you can instead add instances based on how much the aggregate metric exceeds the threshold.

To illustrate, suppose your group starts out with four instances. You want to add more instances to the group as the average CPU utilization of the group increases. When the utilization hits 50 percent, you want to add two more instances. When it goes above 60 percent, you want to add four more instances.

You'd first create a CloudWatch Alarm to monitor the average CPU utilization and set the alarm threshold to 50 percent, since this is the utilization level at which you want to start increasing the desired capacity.

You must then specify at least one step adjustment. Each step adjustment consists of the following:

- A lower bound
- An upper bound
- The adjustment type
- The amount by which to increase the desired capacity

The upper and lower bounds define a range that the metric has to fall within for the step adjustment to execute. Suppose that for the first step you set a lower bound of 50 and an upper bound of 60, with a ChangeInCapacity adjustment of 2. When the alarm triggers, Auto Scaling will consider the metric value of the group's average CPU utilization. Suppose it's 55 percent. Because 55 is between 50 and 60, Auto Scaling will execute the action specified in this step, which is to add two instances to the desired capacity.

Suppose now that you create another step with a lower bound of 60 and an upper bound of infinity. You also set a ChangeInCapacity adjustment of 4. If the average CPU utilization increases to 62 percent, Auto Scaling will note that 60 <= 62 < infinity and will execute the action for this step, adding four instances to the desired capacity.

You might be wondering what would happen if the utilization were 60 percent. Step ranges can't overlap. A metric of 60 percent would fall within the lower bound of the second step.

With a step scaling policy, you can optionally specify a *warm-up time*, which is how long Auto Scaling will wait until considering the metrics of newly added instances. The default warm-up time is 300 seconds. Note that there are no cooldown periods in step scaling policies.

Target Tracking Policies

If step scaling policies are too involved for your taste, you can instead use *target tracking policies*. All you do is select a metric and target value, and Auto Scaling will create a CloudWatch Alarm and a scaling policy to adjust the number of instances to keep the metric near that target.

The metric you choose must change proportionally to the instance load. Metrics like this include average CPU utilization for the group and request count per target. Aggregate metrics like the total request count for the ALB don't change proportionally to the load on an individual instance and aren't appropriate for use in a target tracking policy.

In addition to scaling out, target tracking will scale in by deleting instances to maintain the target metric value. If you don't want this behavior, you can disable scaling in. Also, just as with a step scaling policy, you can optionally specify a warm-up time.

Scheduled Actions

Scheduled actions are useful if you have a predictable load pattern and want to adjust your capacity proactively, ensuring you have enough instances *before* demand hits.

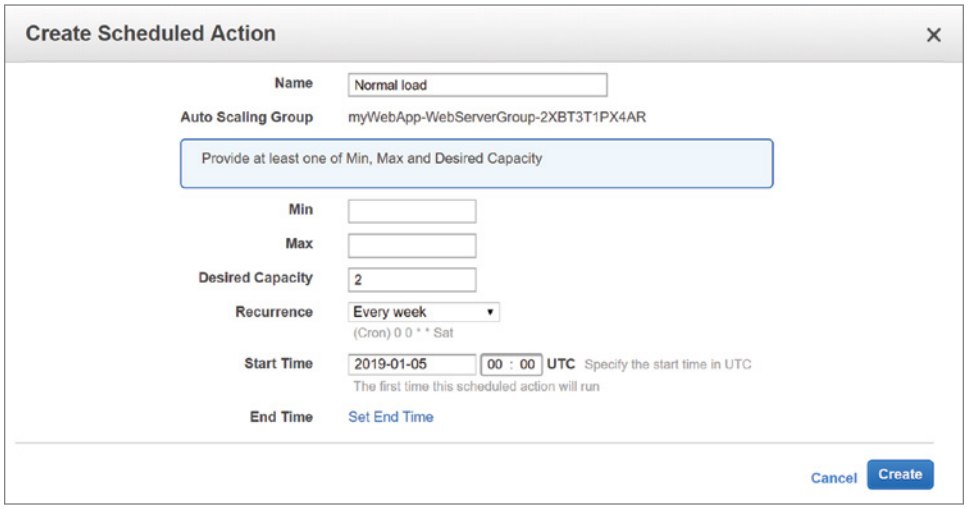
When you create a scheduled action, you must specify the following:

- A minimum, maximum, or desired capacity value
- A start date and time

You may optionally set the policy to recur at regular intervals, which is useful if you have a repeating load pattern. You can also set an end time, after which the scheduled policy gets deleted.

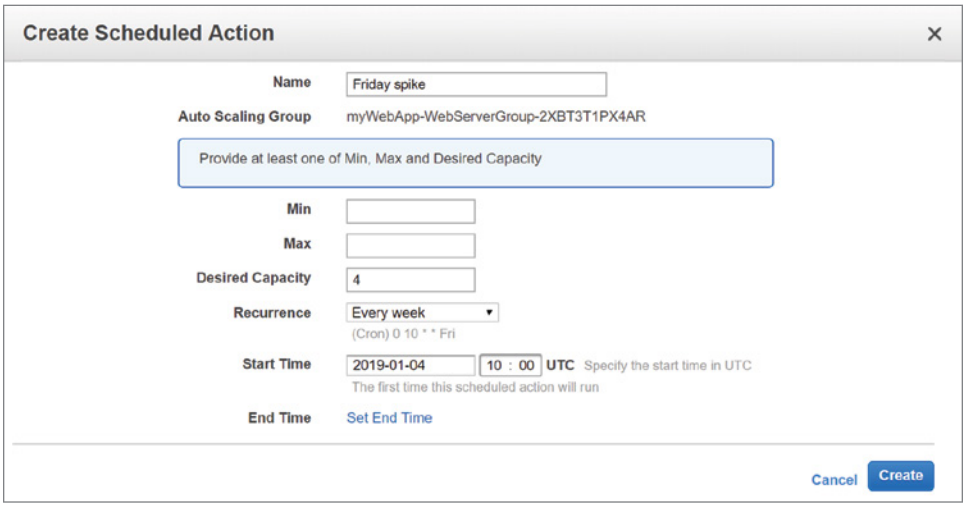
To illustrate how you might use a scheduled action, suppose you normally run only two instances in your Auto Scaling group during the week. But on Friday, things get busy, and you know you'll need four instances to keep up. You'd start by creating a scheduled action that sets the desired capacity to 2 and recurs every Saturday, as shown in Figure 2.3.

FIGURE 2.3 Scheduled action setting the desired capacity to 2 every Saturday



The start date is January 5, 2019, which is a Saturday. To handle the expected Friday spike, you'd create another weekly recurring policy to set the desired capacity to 4, as shown in Figure 2.4.

FIGURE 2.4 Scheduled action setting the desired capacity to 4 every Friday



This action will run every Friday, setting the desired capacity to 4, prior to the anticipated increased load.

Note that you can combine scheduled actions with dynamic scaling policies. For example, if you're running an e-commerce site, you may use a scheduled action to increase the maximum group size during busy shopping seasons and then rely on dynamic scaling policies to increase the desired capacity as needed.

AWS Systems Manager

AWS Systems Manager, formerly known as EC2 Systems Manager and Simple Systems Manager (SSM), lets you automatically or manually perform *actions* against your AWS resources and on-premises servers.

From an operational perspective, Systems Manager can handle many of the maintenance tasks that often require manual intervention or writing scripts. For on-premises and EC2 instances, these tasks include upgrading installed packages, taking an inventory of installed software, and installing a new application. For your other AWS resources, such tasks may include creating an AMI golden image from an EBS snapshot, attaching IAM instance profiles, or disabling public read access to S3 buckets.

Systems Manager provides the following two capabilities:

- Actions
- Insights

Actions

Actions let you automatically or manually perform actions against your AWS resources, either individually or in bulk. These actions must be defined in *documents*, which are divided into three types:

- Automation—actions you can run against your AWS resources
- Command—actions you run against your Linux or Windows instances
- Policy—defined processes for collecting inventory data from managed instances

Automation

Automation enables you to perform actions against your AWS resources in bulk. For example, you can restart multiple EC2 instances, update CloudFormation stacks, and patch AMIs.

Automation provides granular control over how it carries out its individual actions. It can perform the entire automation task in one fell swoop, or it can perform one step at a time, enabling you to control precisely what happens and when. Automation also offers rate control so that you can specify as a number or a percentage how many resources to target at once.

Run Command

While automation lets you automate tasks against your AWS resources, Run commands let you execute tasks on your managed instances that would otherwise require logging in or using a third-party tool to execute a custom script.

Systems Manager accomplishes this via an agent installed on your EC2 and on-premises *managed instances*. The Systems Manager agent is installed by default on more recent Windows Server, Amazon Linux, and Ubuntu Server AMIs. You can manually install the agent on other AMIs and on-premises servers.

By default, Systems Manager doesn't have permissions to do anything on your instances. You first need to apply an instance profile role that contains the permissions in the AmazonEC2RoleforSSM policy.

AWS offers a variety of preconfigured command documents for Linux and Windows instances; for example, the AWS-InstallApplication document installs software on Windows, and the AWS-RunShellScript document allows you to execute arbitrary shell scripts against Linux instances. Other documents include tasks such as restarting a Windows service or installing the CodeDeploy agent.

You can target instances by tag or select them individually. As with automation, you may use rate limiting to control how many instances you target at once.

Session Manager

Session Manager lets you achieve interactive Bash and PowerShell access to your Linux and Windows instances, respectively, without having to open up inbound ports on a security group or network ACL or even having your instances in a public subnet. You don't need to set up a protective bastion host or worry about SSH keys. All Linux versions and Windows Server 2008 R2 through 2016 are supported.

You open a session using the web console or AWS CLI. You must first install the Session Manager plug-in on your local machine to use the AWS CLI to start a session. The Session Manager SDK has libraries for developers to create custom applications that connect to instances. This is useful if you want to integrate an existing configuration management system with your instances without opening ports in a security group or NACL.

Connections made via Session Manager are secured using TLS 1.2. Session Manager can keep a log of all logins in CloudTrail and store a record of commands run within a session in an S3 bucket.

Patch Manager

Patch Manager helps you automate the patching of your Linux and Windows instances. It will work for supporting versions of the following operating systems:

- Windows Server
- Ubuntu Server
- Red Hat Enterprise Linux (RHEL)
- SUSE Linux Enterprise Server (SLES)
- CentOS
- Amazon Linux
- Amazon Linux 2

You can individually choose instances to patch, patch according to tags, or create a *patch group*. A patch group is a collection of instances with the tag key Patch Group. For example, if you wanted to include some instances in the Webservers patch group, you'd assign tags to each instance with the tag key of Patch Group and the tag value of Webservers. Keep in mind that the tag key is case-sensitive.

Patch Manager uses *patch baselines* to define which available patches to install, as well as whether the patches will be installed automatically or require approval.

AWS offers default baselines that differ according to operating system but include patches that are classified as security related, critical, important, or required. The patch baselines for all operating systems except Ubuntu automatically approve these patches after seven days. This is called an *auto-approval delay*.

For more control over which patches get installed, you can create your own custom baselines. Each custom baseline contains one or more approval rules that define the operating system, the classification and severity level of patches to install, and an auto-approval delay.

You can also specify approved patches in a custom baseline configuration. For Windows baselines, you can specify knowledgebase and security bulletin IDs. For Linux baselines, you can specify Common Vulnerabilities and Exposures (CVE) IDs or full package names. If a patch is approved, it will be installed during a maintenance window that you specify. Alternatively, you can forego a maintenance window and patch your instances immediately. Patch Manager executes the AWS-RunPatchBaseline document to perform patching.

State Manager

While Patch Manager can help ensure your instances are all at the same patch level, State Manager is a configuration management tool that ensures your instances have the software you want them to have and are configured in the way you define. More generally, State Manager can automatically run command and policy documents against your instances, either one time only or on a schedule. For example, you may want to install antivirus software on your instances and then take a software inventory.

To use State Manager, you must create an *association* that defines the command document to run, any parameters you want to pass to it, the target instances, and the schedule. Once you create an association, State Manager will immediately execute it against the target instances that are online. Thereafter, it will follow the schedule.

There is currently only one policy document you can use with State Manager: AWS-GatherSoftwareInventory. This document defines what specific metadata to collect from your instances. Despite the name, in addition to collecting software inventory, you can have it collect network configurations, file information, CPU information, and for Windows, registry values.

Insights

Insights aggregate health, compliance, and operational details about your AWS resources into a single area of AWS Systems Manager. Some insights are categorized according to *AWS resource groups*, which are collections of resources in an AWS region. You define a resource group based on one or more tag keys and optionally tag values. For example, you can apply the same tag key to all resources related to a particular application—EC2 instances, S3 buckets, EBS volumes, security groups, and so on. Insight categories are covered next.

Built-in Insights

Built-in insights are monitoring views that Systems Manager makes available to you by default. Built-in insights include the following:

AWS Config Compliance This insight shows the total number of resources in a resource group that are compliant or noncompliant with AWS Config rules, as well as compliance by resource. It also shows a brief history of configuration changes tracked by AWS Config.

CloudTrail Events This insight displays each resource in the group, the resource type, and the last event that CloudTrail recorded against the resource.

Personal Health Dashboard The Personal Health Dashboard contains alerts when AWS experiences an issue that may impact your resources. For example, some service APIs occasionally experience increased latency. It also shows you the number of events that AWS resolved within the last 24 hours.

Trusted Advisor Recommendations The AWS Trusted Advisor tool can check your AWS environment for optimizations and recommendations related to cost optimization, performance, security, and fault tolerance. It will also show you when you've exceeded 80 percent of your limit for a service.

Business and Enterprise support customers get access to all Trusted Advisor checks. All AWS customers get the following security checks for free:

- Public access to an S3 bucket, particularly upload and delete access
- Security groups with unrestricted access to ports that normally should be restricted, such as TCP port 1433 (MySQL) and 3389 (Remote Desktop Protocol)
- Whether you've created an IAM user
- Whether multifactor authentication is enabled for the root user
- Public access to an EBS or RDS snapshot

Inventory Manager

The Inventory Manager collects data from your instances, including operating system and application versions. Inventory Manager can collect data for the following:

- Operating system name and version
- Applications and filenames, versions, and sizes
- Network configuration, including IP and media access control (MAC) addresses
- Windows updates, roles, services, and registry values
- CPU model, cores, and speed

You choose which instances to collect data from by creating a regionwide *inventory association* by executing the AWS-GatherSoftwareInventory policy document. You can choose all instances in your account or select instances manually or by tag. When you choose all instances in your account, it's called a *global inventory association*, and new instances you create in the region are automatically added to it. Inventory collection occurs at least every 30 minutes.

When you configure the Systems Manager agent on an on-premises server, you specify a region for inventory purposes. To aggregate metadata for instances from different regions and accounts, you may configure Resource Data Sync in each region to store all inventory data in a single S3 bucket.

Compliance

Compliance insights show how the patch and association status of your instances stacks up against the rules you've configured. Patch compliance shows the number of instances that have the patches in their configured baseline, as well as details of the specific patches installed. Association compliance shows the number of instances that have had an association successfully executed against them.

AWS CLI Example

The following example code shows how you can use an AWS CLI command to deploy an EC2 instance that includes many of the features you learned about in this chapter. Naturally, the `image-id`, `security-group-ids`, and `subnet-id` values are not real. Those you would replace with actual IDs that fit your account and region.

```
aws ec2 run-instances --image-id ami-xxxxxxxx --count 1 \
--instance-type t2.micro --key-name MyKeyPair \
--security-group-ids sg-xxxxxxxx --subnet-id subnet-xxxxxxxx \
--user-data file://my_script.sh \
--tag-specifications \
'ResourceType=instance,Tags=[{Key=webserver,Value=production}]' \
'ResourceType=volume,Tags=[{Key=cost-center,Value=cc123}]'
```

This example launches a single (`--count 1`) instance that's based on the specified AMI. The desired instance type, key name, security group, and subnet are all identified. A script file (that must exist locally so it can be read) is added using the `user-data` argument, and two tags are associated with the instance (`webserver:production` and `cost-center:cc123`).

If you need to install the AWS CLI, perform Exercise 2.6.

EXERCISE 2.6

Install the AWS CLI and Use It to Launch an EC2 Instance

Need help? Learn how to install the AWS CLI for your OS here:

`docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html`

Refer to the previous AWS CLI example for help launching your instance. (Hint: You will need to fill in some `xxxxx` placeholders with actual resource IDs.)

Never leave any resources running after you've finished using them. Exercise 2.7 can help.