# Chapter 2. First Steps with EC2 and CloudFormation

Launched in 2006, Elastic Compute Cloud (or EC2, as it is universally known) is a core part of AWS, and probably one of the better-known components of the service. It allows customers to rent computing resources by the hour in the form of virtual machines (known asinstances) that run a wide range of operating systems. These instances can be customized by the user to run any software applications supported by their operating system of choice.

The idea of renting computing resources by the hour goes back to the 1960s, when it was simply not financially feasible for a company or university department to own a dedicated computer (the idea of an individual owning a computer seeming, at this point, to be the stuff of science fiction). This changed as computers became cheaper and more popular, and dedicated computing resources became the norm.

The explosive growth of the consumer Internet, and thus of the services and applications that make up the motivation for its ever-increasing use, has helped the pendulum swing back the other way, to the point where being able to elastically increase or decrease your computing resources (and therefore costs) has become a key financial advantage.

In the pre-cloud days, capacity planning required a large amount of time and forward thinking. Bringing new servers online was a multistep process with the potential for delays at every step: ordering hardware from the supplier, waiting for its arrival, visiting the datacenter to unpack and rack the server, and installing and configuring the operating system and software. Renting a virtual private server, while usually quicker than provisioning physical hardware, also had its own set of challenges and potential delays. With the launch of EC2, all of this was replaced by a single API call.

Particularly in the consumer-driven web application market, it is possible for new companies to experience month after month of exponential growth. This can lead to service interruption as systems administrators struggle valiantly to ensure that the demands of their users do not surpass their supply of computing power. This process is often one of the key factors in the success of young companies and also presents one of their most acute challenges—if you do not have enough computing capacity, your users will quickly tire of seeing error pages and move on to a competitor. But oversupply can be equally terminal, as you will be paying for unused computing capacity. This contributed to the failure of many companies in the 2000 dot-com bubble: they spent a huge amount of money in capital expenses building datacenter capacity to support users who never materialized.

EC2 provides a particularly interesting approach to solving this problem. As instances can be launched and terminated automatically based on your current traffic levels, it is possible to dynamically design your infrastructure to operate at (for example) 80% utilization. Large upfront hardware purchases are then replaced by a much smaller, ongoing operational expense exactly matching your capacity needs.

Flexibility is at the heart of the AWS product offering, and this flexibility also extends to the way one interacts with AWS. For most people, the first steps with EC2 are taken via the Management Console, which is the public face of EC2. This web application lets you control most aspects of your infrastructure, although some features (such as Auto Scaling groups, discussed later in the book) require the use of API calls or command-line tools. Historically, Amazon has usually provided command-line tools and API access to new features before they appear in the Management Console.

At the lowest level, AWS is “simply” an HTTP-based API. You can submit a request asking for 10 t2.micro instances, the API request is processed, and 10 instances are launched. The Management Console is merely another way of interacting with this API.

This book uses all the available methods provided by AWS. In nearly all cases, the methods are interchangeable. If a feature specifically requires you to use the command-line tools, we will indicate this. So, if you are familiar with AWS, you should feel free to ignore our recommendations and use whichever method you feel most comfortable with.

# What Is an Instance?

At the simplest level, an instance can be thought of as a virtual server, the same as you might rent on a monthly basis from a virtual private server (VPS) provider. Indeed, some people are using EC2 in exactly the same way as they would a VPS. While perfectly serviceable in this respect, to use it in this way ignores several interesting features and technologies that can make your job a lot more convenient.

Amazon Machine Images (AMIs) are the main building blocks of EC2. They allow you to configure an instance once (say, installing Apache or Nginx) and then create an image of that instance. The image can be used to launch more instances, all of which are functionally identical to the original. Of course, some attributes—such as the IP address or instance ID—must be unique, so there will be some differences.

##### AWS REGIONS AND AVAILABILITY ZONES

AWS services operate in multiple geographic regions around the world. At the time of this writing, there are fifteen public AWS regions, each of which is further divided into multiple availability zones. This geographic disparity has two main benefits: you can place your application resources close to your end users for performance reasons, and you can design your application so that it is resilient to loss of service in one particular region or availability zone. AWS provides the tools to build automatic damage control into your infrastructure, so if an availability zone fails, more resources can be provisioned in the other availability zones to handle the additional load.

Each availability zone (AZ) is located in a physically separate datacenter within its region. There are three datacenters in or around Dublin, Ireland that make up the three availability zones in the EU West 1 region—each with separate power and network connections. In theory, this means that an outage in one AZ will not have any effect on the other AZs in the region. In practice, however, an outage in one AZ can trigger a domino effect on its neighboring AZs, and not necessarily due to any failing on Amazon’s part.

Consider a well-architected application that, in the event of an AZ failure, will distribute traffic to the remaining AZs. This will result in new instances being launched in the AZs that are still available. Now consider what happens when hundreds of well-architected applications all fail-over at the same time—the rush for new instances could outstrip the capability of AWS to provide them, leaving some applications with too few instances.

This is an unlikely event—although AWS has service outages like any other cloud provider, deploying your application to multiple AZs will usually be sufficient for most use cases. To sustain the loss of a significant number of AZs within a region, applications must be deployed to multiple regions. This is considerably more challenging than running an application in multiple AZs.

[Chapter 6](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch06.html#aws_sysadmin_autoscaling) demonstrates an example application that can survive the loss of one or more AZs, while [reserved and spot instances](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch02.html#reserved_and_spot) provide a way around capacity shortages in a failover.

## Instance Types

EC2 instances come in a range of sizes, referred to as instance types, to suit various use cases. The instance types differ wildly in the amount of resources allocated to them. The m3.medium instance type has 3.75 GB of memory and 1 virtual CPU core, whereas its significantly bigger brother c3.8xlarge has 60 GB of memory and 32 virtual CPU cores. Each virtual CPU is a hyperthread of an Intel Xeon core in the m3 and c3 instance classes.

For most of the examples in the book, we will use a t2.micro instance, among the smaller and one of the cheapest instance types suitable for any operating system choice, which makes it ideal for our tests.

In production, picking the right instance type for each component in your stack is important to minimize costs and maximize performance, and benchmarking can be the key when making this decision.

## Processing Power

EC2, along with the rest of AWS, is built using commodity hardware running Amazon’s software to provide the services and APIs. Because Amazon adds this hardware incrementally, several hardware generations are in service at any one time.

###### WARNING

When it comes to discussing the underlying hardware that makes up the EC2 cloud, Amazon used to play the cards close to its chest and reveal relatively little information about the exact hardware specifications. This led to the creation of a dedicated compute unit:

*One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.*

It is easy to encounter this metric in older AWS benchmarks. Amazon now openly identifies what hardware underlies the EC2 compute layer, and these abstract units are obsolete and no longer in use.

Amazon provides a rather vast selection of instance types, the current generation of which is described at the [EC2 Instance Types](http://aws.amazon.com/ec2/instance-types/) page. The previously mentioned t2.micro instance type therefore refers to a second generation general purpose burstable performance instance. An immediate update of already running applications is generally not required as [older generations](http://aws.amazon.com/ec2/previous-generation/) remain available for provisioning, with their original functionality intact. It remains advisable to adopt the latest instance type generation when designing a new (or revised) application, so as to benefit from the capabilities of the newer hosting hardware.

###### TIP

No EC2 instance type has ever been discontinued in almost 10 years. This record is made possible by market forces: as newer instance types become available, their significantly better price/performance ratio induces a user migration away from the previous generation. A reduced demand base in turn allows Amazon to continue to supply those deprecated instance types without having to add capacity with old hardware that may be unavailable.

Older instance types are however not available in the newer AWS regions they pre-date-for example, the first generation to be deprecated, cc1, is not found in the newest region ap-northeast-2 hosted in Seoul, Korea. If our spirited advice and the cost savings produced by migrating to newer instance generations are not sufficient to entice you to regularly update your instance selection, perhaps your global expansion plans will.

AWS machine images may make use of either of the two virtualization types supported by the Xen hypervisor: paravirtualized or hardware virtual machine (HVM). It is not necessary to be conversant in the finer differences of the two technologies to make effective use of AWS, but the two approaches present boot-time differences to the guest OS environment. A given Linux machine image will only support booting one virtualization type as a result, a requirement easily met filtering any image search by the appropriate virtualization type.

Amazon recommends using HVM virtualization on current-generation AMIs. Where that approach is not suitable, it becomes necessary to determine what virtualization type is supported by the older generation of a specific instance type. This is quickly accomplished by launching a test HVM instance from the AWS CLI and watching for a helpful error message. The AWS documentation also provides insight into what [virtualization type](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/virtualization_types.html) is supported by what older instance type.

Different combinations of CPU, memory, network bandwidth and even custom hardware differentiate AWS instance types. There are 9 instance type classes in the current generation at the time of writing, including general purpose (M4, M3), burstable performance (T2), compute optimized (C4, C3), memory intensive (R3), storage optimized (I2 for performance, or D2 for cost), and GPU enabled (G2). These in turn include multiple types with resource allottments of increasing size, bringing the total number of choices we choose from above forty.

###### NOTE

Jeff Barr of Amazon has published an interesting [timeline](https://aws.amazon.com/blogs/aws/ec2-instance-history/) of EC2’s instance generations.

Taking a scientific approach to benchmarking is the only way to really be sure you are using the right instance type. AWS makes it really simple to run the very same instance with a succession of different instance types, considerably simplifying this task. The most common approach in the AWS user community is to start with an instance type considered high-CPU for the workload under consideration. While running top, drive the CPU to 100% using your application’s load generator of choice. Now examine memory use: if you observe the instance running out of memory before the CPU is at full throttle, switch to a higher-memory instance type. Continue this process until you achieve a reasonable balance.

Alongside fixed-performance instances, including the C4, C3 and R3 types, EC2 offers burstable performance instances like the T2 type. Burstable performance instances generally operate at a CPU performance baseline but can “burst” above this limit for a time. Bursting is governed by CPU credits that are accumulated when the instance runs without its full allottment of CPU. A CPU Credit represents use of a full CPU core for one minute.

A practical example will illustrate the accounting mechanism EC2 employs: a t2.microinstance type allocates one virtual CPU to your cloud instance, with 6 CPU credits earned each hour, representing a 10% share of a real CPU core. Let’s assume our workload is a web server, often idling while waiting for requests. If the CPU load falls below 10%, CPU credits are added to that instance’s credit for up to 24 hours. Burstable performance is particularly useful for workloads that do not consistently use their full share of the CPU, but benefit from having access to additional, fast CPUs when the occasion arises - applications include small databases, web servers and development systems.

##### STOLEN CPU TIME

Alongside the traditional CPU shares of us(user), sy(system), id(idle), and wa(IO wait), the EC2 hypervisor exposes the additional metric st, meaning stolen:

%Cpu(s): 0.1 us, 0.1 sy, 0.1 ni, 98.2 id, 1.0 wa, 0.0 hi, 0.0 si, 0.5 st

Stolen CPU time represents the share of time the instance’s virtual CPU has been waiting for a real CPU while the hypervisor is using it to service another virtual processor. Stolen CPU has gained prominence as a metric that Netflix Inc., possibly the most prominent AWS tenant, tracks closely. Despite its present fame, Stolen CPU is not as significant for workloads that are not sensitive to network jitter or real-time in nature.

The Noisy Neighbor is a related compute Cause célèbre: in any virtual environment, the noisy neighbor effect occurs when an instance starves other instances for a shared resource, causing performance issues to others on the same infrastructure. You will not observe memory or CPU contention as EC2 instances are generally not overprovisioned, any potential noisy neighbor problems will be limited to network or disk IO.

One simple approach countering this issue is to automatically allocate a new instance, replacing the one where the performance problem was encountered. Larger instance types are less likely to present this problem on account of sharing a host with fewer neighbors. SR-IOV support (Enhanced Networking) increases storage and network IO bandwidth, helping to minimize any noise. The most reliable approach is to use [Dedicated hosts](https://aws.amazon.com/ec2/dedicated-hosts/), a facility providing complete control of your instance placement for an additional fee.

Specific instance types may provide the latest advanced features found in Intel hardware, including on-chip support for AES encryption and the Advanced Vector Extensions instruction set. The G2 instance type is currently the most promiment example of enhanced compute support, featuring more than 1,500 NVIDIA GPU cores. Advanced compute options are rapidly evolving, their most recent iteration is documented in the [instance types](http://aws.amazon.com/ec2/instance-types/) page, which we recommend you review often.

EC2 instances can be purchased in three ways. Allocated by the hour and requiring no upfront committment, on-demand instances are the default and are used exclusively throughout this book. Reserved instances represent a pre-paid committment on the part of a customer which is usually rewarded by AWS with very steep discounts, up to 75% of on-demand pricing. Spot instance pricing requires no upfront committment, and their pricing fluctuates according to the supply and demand of compute capacity. The customer may define a maximum hourly price not to be exceeded, and EC2 will automatically shut those instances down if their spot pricing tops the set threshold.

## Storage

There are two options when it comes to virtual disk storage for your instances: instance storage (also known as ephemeral storage) and Elastic Block Store (or EBS). Both are simply block storage devices that can be attached to instances. Once attached, they can be formatted with your operating system’s tools and will act like a standard disk. AWS storage comes in two flavors: magnetic disks and solid-state drives (SSDs). SSDs provide higher read and write performance when compared to magnetic disks, but the cost is slightly higher.

There are some key differences between instance storage and EBS. Instance storage is directly attached to the physical host that runs your instance, whereas EBS is attached over the network. This has implications in terms of disk latency and throughput, so we recommend performing another series of benchmarks to see which is best if your application is sensitive to latency or IO jitter.

IO speeds are not the only difference—EBS has features that make it preferable to instance storage in nearly all usage scenarios. One of the most useful is the ability to create a snapshot from an EBS. A snapshot is a copy of an EBS volume at a particular point in time. Once you have created a snapshot, you can then create additional EBS volumes that will be identical copies of the source snapshot. You could, for example, create a snapshot containing your database backups. Every time a new instance is launched, it will have a copy of the data ready for use. EBS snapshots form the backbone of many AWS backup strategies.

When an instance is terminated, any data stored on instance storage volumes is lost permanently. EBS volumes can persist after the instance has been terminated. Given all of the additional features, using EBS volumes is clearly preferable except in a few cases, such as when you need fast temporary storage for data that can be safely discarded.

Multiple volumes (of either type) can be attached to an instance, leading to pretty flexible storage configurations. The [Block Device Mapping](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/block-device-mapping-concepts.html) facility allows multiple volumes to be associated with an instance at boot time. It is even possible to attach multiple volumes to an instance and build a software RAID array on them — an advantage of volumes appearing as block storage devices to the operating system.

In June 2012, AWS began offering SSDs as a higher-performance alternative to magnetic storage, and over time introduced multiple options with different performance levels and cost. Some instance types now include an SSD-backed instance store to deliver very-high random IO performance, with types I2 and R3 being the first to support TRIM extensions. Instance types themselves have evolved to include high-IO instances (type I2), aimed at delivering high IOPS from up to 8 local SSD drives, while dense storage instances (type D2) offer the lowest price per disk throughput in EC2 and balance cost and performance, using 24 local magnetic drives.

EBS Magnetic volumes are currently limited to 1TB in size, with SSD volumes topping out at 16 TB, limits easily exceeded by instance storage’s local disks. Whereas EBS volumes can be provisioned at any time and in arbitrary configurations, the number and size of available instance store volumes varies with instance type, and can only be attached to an instance at boot time. EBS volumes can also be dynamically resized.

EBS SSD options include a number of performance flavors. General purpose SSD volumes are provisioned with 3 IOPS per GB, with burst performance reaching 3,000 IOPS for extended periods. Provisioned IOPS SSD volumes allow the user to define the desired level of performance, up to 20,000 IOPS and 320 MB/s of throughput. A less costly option is offered by the [EBS-optimized](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSOptimized.html) M4 type instances, which include dedicated EBS bandwidth between 450 and 4,000 Mbps depending on the specific instance type. EBS–optimized instances use an optimized configuration stack requiring corresponding support on the machine image’s part for optimal performance (see [Finding Ubuntu Images](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch02.html#finding_ubuntu) for details on locating optimized images).

Long term storage options are best supported by the S3 service, but a block storage option is available through Cold HDD EBS volumes. Backed by magnetic drives, Cold HDD volumes offer the lowest cost per GB of all EBS volume types, and still provide enough performance to support a full-volume scan at burst speeds. EBS also supports [native at-rest encryption](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSEncryption.html)that is transparently available to EC2 instances and requires very little effort on the administrator’s part to deploy and maintain. EBS encryption has no IOPS performance impact and shows very limited impact on latency, making it a general-purpose architectural option even when high-security is not strictly required.

## Networking

At its simplest, networking in AWS is straightforward—launching an instance with the default networking configuration will give you an instance with a public IP address. Many applications will require nothing more complicated than enabling SSH or HTTP access. At the other end of the scale, Amazon offers more-advanced solutions that can, for example, give you a secure VPN connection from your datacenter to a Virtual Private Cloud (VPC) within EC2.

At a minimum, an AWS instance has one network device attached. The maximum number of network devices that can be attached depends on the instance type. Running ip addr show on the instance will show that it has a private IP address in the 172.31.0.0/16range. Every instance has a private IP and a public IP; the private IP can be used only within the EC2 network.

###### WARNING

AWS accounts created after December 2013 no longer have access to the legacy EC2-classic networking model. This book covers the current EC2-VPC networking model exclusively.

Amazon Virtual Private Cloud enables you to provision EC2 instances in a virtual network of your own design. A VPC is a network dedicated to your account, isolated from other networks in AWS, and completely under your control. You can create subnets and gateways, configure routing, select IP address ranges and define its security perimeter - a series of complex tasks that are bypassed by the existance of the default VPC. The [default VPC](http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/default-vpc.html) includes a default subnet in each availability zone, along with routing rules, a DHCP setup, and an Internet gateway. The default VPC enables new accounts to immediately start launching instances without having to first master advanced VPC configuration, but its security configuration will not allow instances to accept connections from the Internet until we expressly give our permission, by assigning our own security group settings.

The default security group allows all outbound traffic from instances to reach the Internet, and also permits instances in the same security group to receive inbound traffic from one another, but not from the outside world. Instances launched in the default VPC receive both a public and a private IP address. Behind the scenes, AWS will also create two DNS entries for convenience.

For example, if an instance has a private IP of 172.31.16.166 and a public IP of 54.152.163.171, their respective DNS entries will be ip-172-31-16-166.ec2.internal and ec2-54-152-163-171.compute-1.amazonaws.com. These DNS entries are known as the private hostname and public hostname.

It is interesting to note that Amazon operates a split-view DNS system, which means it is able to provide different responses depending on the source of the request. If you query the public DNS name from outside EC2 (not from an EC2 instance), you will receive the public IP in response. However, if you query the public DNS name from an EC2 instance in the same region, the response will contain the private IP:

# From an EC2 instance

$ dig ec2-54-152-163-171.compute-1.amazonaws.com +short

172.31.16.166

# From Digital Ocean

$ dig ec2-54-152-163-171.compute-1.amazonaws.com +short

54.152.163.171

The purpose of this is to ensure that traffic does not leave the internal EC2 network needlessly. This is important as AWS has a highly granular pricing structure when it comes to networking, and Amazon makes a distinction between traffic destined for the public Internet and traffic that will remain on the internal EC2 network. The full breakdown of costs is available on the [EC2 Pricing](https://aws.amazon.com/ec2/pricing/#Data_Transfer) page.

If two instances in the same availability zone communicate using their private IPs, the data transfer is free of charge. However, using their public IPs will incur Internet transfer charges on both sides of the connection. Although both instances are in EC2, using the public IPs means the traffic will need to leave the internal EC2 network, which will result in higher data transfer costs.

By using the private IP of your instances when possible, you can reduce your data transfer costs. AWS makes this easy with their split-horizon DNS system: as long as you always reference the public hostname of the instance (rather than the public IP), AWS will pick the cheapest option.

Most of the early examples in the book use a single interface, and we will look at more exotic topologies in later chapters.

# Launching Instances

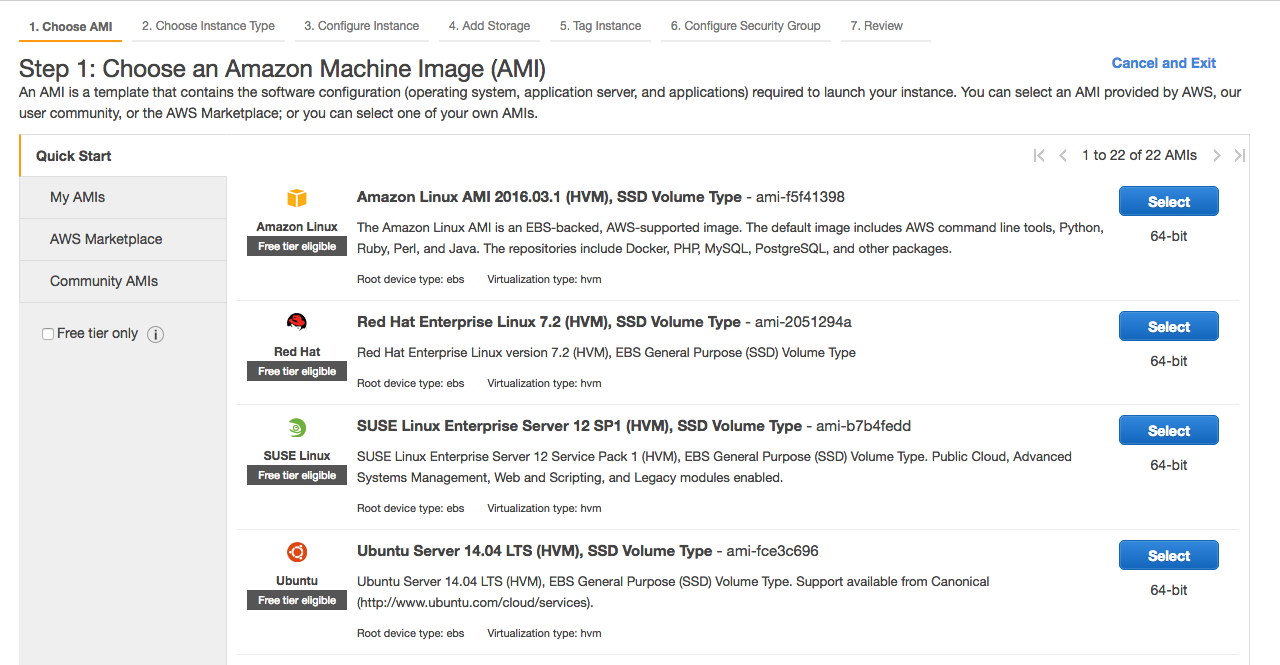
The most useful thing one can do with an instance is launch it, which is a good place for us to start. As an automation-loving sysadmin, you will no doubt quickly automate this process and rarely spend much time manually launching instances. Like any task, though, it is worth stepping slowly through it the first time to familiarize yourself with the process.

## Launching from the Management Console

Most people take their first steps with EC2 via the Management Console, which is the public face of EC2. Our first journey through the Launch Instance Wizard will introduce a number of new concepts, so we will go through each page in the wizard and take a moment to look at each of these in turn. Although there are faster methods of launching an instance, the wizard is certainly the best way to familiarize yourself with related concepts.

### LAUNCHING A NEW INSTANCE OF AN AMI

To launch a new instance, first log in to [Amazon’s web console](http://aws.amazon.com/), open the EC2 section, and click Launch Instance. This shows the first in a series of pages that will let us configure the instance options. The first of these pages is shown in [Figure 2-1](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch02.html#launch_instance_ami).

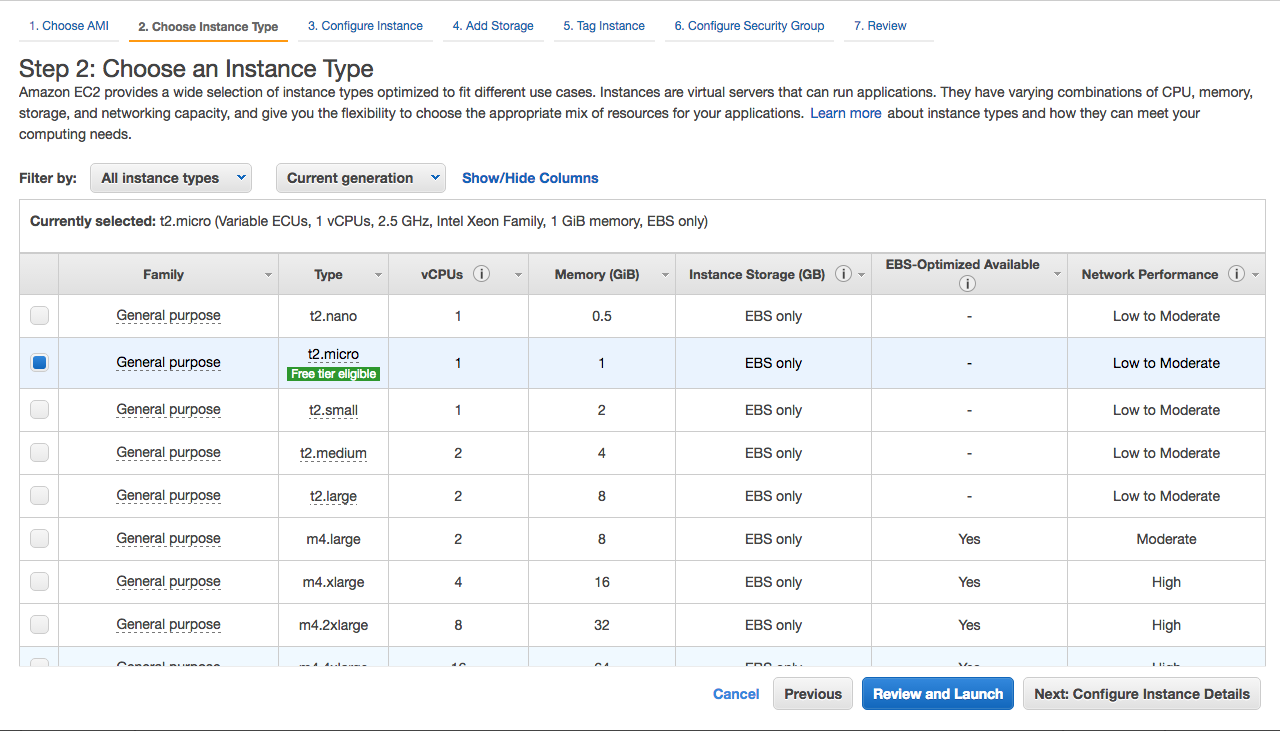


###### *Figure 2-1. AMI selection*

As described earlier, Amazon Machine Images (AMIs) are used to launch instances that already have the required software packages installed, configured, and ready to run. Amazon provides AMIs for a variety of operating systems, and the Community and Marketplace AMIs provide additional choices. For example, Canonical provides officially supported AMIs for various versions of its Ubuntu operating system. Other open source and commercial operating systems are also available, both with and without support. The AWS Marketplace lets you use virtual appliances created by Amazon or third-party developers. These are Amazon Machine Images already configured to run a particular set of software; for example, many variations of AMIs running the popular WordPress blogging software exist. While some of these appliances are free to use (i.e., you only pay for the underlying AWS resources you use), others require you to pay an additional fee on top of the basic cost of the Amazon resources.

If this is your first time launching an instance, the My AMIs tab will be empty. Later in this chapter, we will create our own custom AMIs, which will subsequently be available via this tab. The Quick Start tab lists several popular AMIs that are available for public use.

Click the Select button next to the Amazon Linux AMI. This gives you instance types to choose from ([Figure 2-2](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch02.html#launch_instance_type)).



###### *Figure 2-2. Selecting the instance type*

EC2 instances come in a range of shapes and sizes to suit many use cases. In addition to offering increasing amounts of memory and CPU power, instance types also offer differing ratios of memory to CPU. Different components in your infrastructure will vary in their resource requirements, so it can pay to benchmark each part of your application to see which instance type is best for your particular needs. You can also find useful Community-developed resources to quickly compare instance types at [EC2instances.info](http://www.ec2instances.info/).

The Micro instance class is part of Amazon’s free usage tier. New customers can use 750 instance-hours free of charge each month with the Linux and Windows microinstance types. After exceeding these limits, normal on-demand prices apply.

Select the checkbox next to t2.micro and click Review and Launch. Now you are presented with the review screen, which gives you a chance to confirm your options before launching the instance.

##### EC2 INSTANCE DETAILS AND USER DATA

So far, we have been using only the most common options when launching our instance. As you will see on the review screen, there are a number of options that we have not changed from the defaults. Some of these will be covered in great detail later in the book, whereas others will rarely be used in the most common use cases. It is worth looking through the advanced options pages to familiarize yourself with the possibilities.

User data is an incredibly powerful feature of EC2, and one that will be used a lot later in the book to demonstrate some of the more interesting things you can do with EC2 instances. Any data entered in this box will be available to the instance once it has launched, which is a useful thing to have in your sysadmin toolbox. Among other things, user data lets you create a single AMI that can fulfill multiple roles depending on the user data it receives, which can be a huge time-saver when it comes to maintaining and updating AMIs. Ubuntu and Amazon Linux support using shell scripts as user data, so you can provide a custom script that will be executed when the instance is launched.

Furthermore, user data is accessible to configuration management tools such as Puppet or Chef, allowing dynamic configuration of the instance based on user data supplied at launch time. This is covered in further detail in [Chapter 4](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch04.html#aws_config_management).

The Kernel ID and RAM Disk ID options will rarely need to be changed if you are using AMIs provided by Amazon or other developers.

Termination protection provides a small level of protection against operator error in the Management Console. When running a large number of instances, it can be easy to accidentally select the wrong instance for termination. If termination protection is enabled for a particular instance, you will not be able to terminate it via the Management Console or API calls. This protection can be toggled on or off while the instance is running, so there is no need to worry that you will be stuck with an immortal instance. Mike can personally attest to its usefulness—it once stopped him from erroneously terminating a production instance running a master database.

IAM roles are covered in [Chapter 3](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch03.html#chap_aws_access_security). Briefly, they allow you to assign a security role to the instance. Access keys are made available to the instance so it can access other AWS APIs with a restricted set of permissions specific to its role.

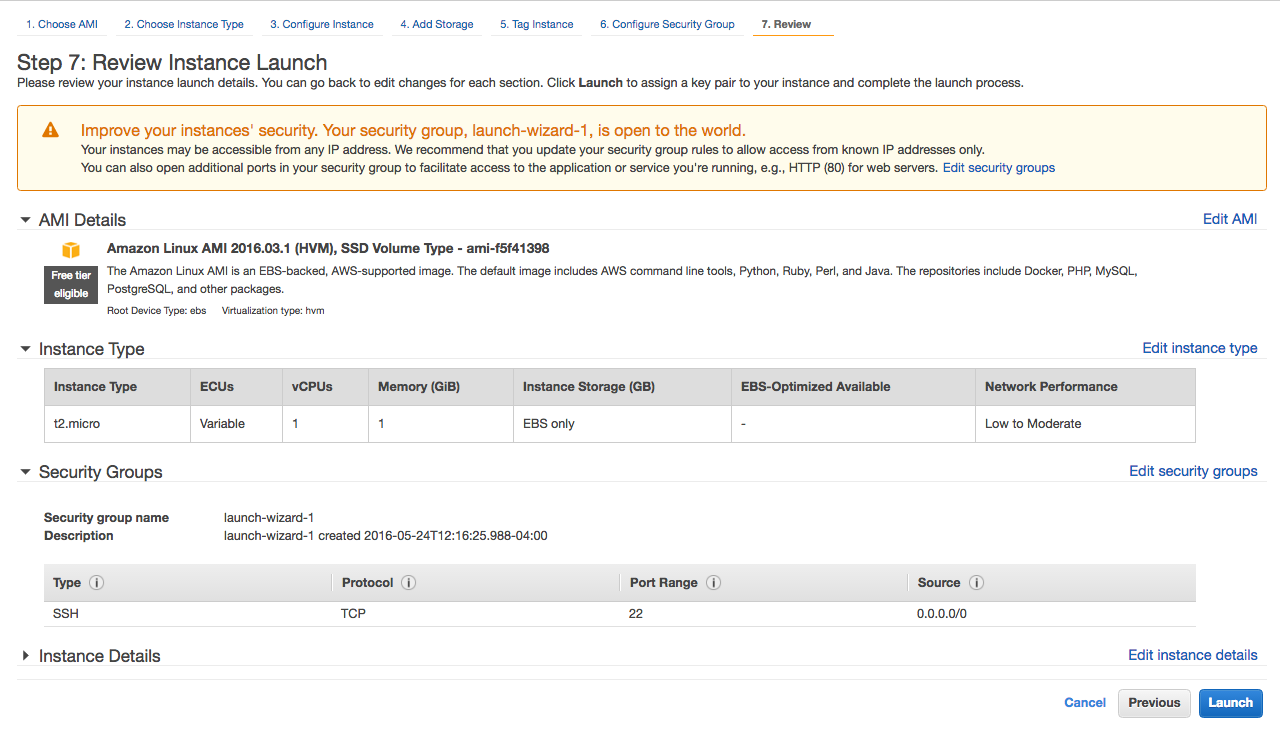
Most of the time your instances will be terminated through the Management Console or API calls. Shutdown Behavior controls what happens when the instance itself initiates the shutdown, for example, after running shutdown -h now on a Linux machine. The available options are to stop the machine so it can be restarted later, or to terminate it, in which case it is gone forever.

Tags are a great way to keep track of your instances and other EC2 resources via the Management Console.

Tags perform a similar role to user data, with an important distinction: user data is for the instance’s internal use, whereas tags are primarily for external use. An instance does not have any built-in way to access tags, whereas user data, along with other metadata describing the instance, can be accessed by reading a URL from the instance. It is, of course, possible for the instance to access its own tags by querying the EC2 API, but that would require API access privileges to be granted to the instance itself in the form of a key, something less than desirable in a healthy security posture.

Using the API, you can perform queries to find instances that are tagged with a particular key/value combination. For example, two tags we always use in our EC2 infrastructures are environment (which can take values such as production or staging) and role(which, for instance, could be webserver or database). When scripting common tasks—deployments or software upgrades—it becomes a trivial task to perform a set of actions on all web servers in the staging environment. This makes tags an integral part of any well-managed AWS infrastructure.

If the Cost Allocation Reports feature ( on the billing options page of your account settings page) is enabled, your CSV-formatted bill will contain additional fields, allowing you to link line-item costs with resource tags. This information is invaluable when it comes to identifying areas for cost savings, and for larger companies where it is necessary to separate costs on a departmental basis for charge-back purposes. Even for small companies, it can be useful to know where your sources of cost are.

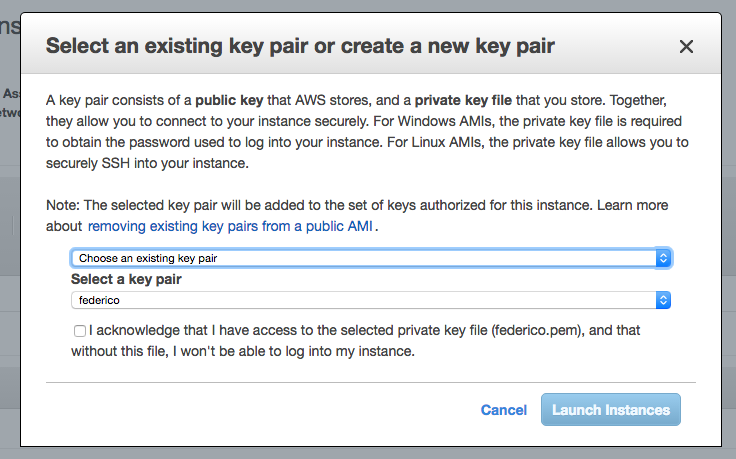


###### *Figure 2-3. The Review screen. The prominent security warning is alerting you that SSH access has been opened with a default security group.*

After reviewing the options, click Launch to move to the final screen. At the time of this writing, the wizard’s Quick Start process will automatically create a convenient launch-wizard-1 security group granting the instance SSH access from the Internet at large. This is not the default security group previously discussed, and this helpfulness is not present when using the AWS CLI or API interfaces to create instances ([Figure 2-3](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch02.html#aws_fig_review)).

### KEY PAIRS

The next screen presents the available Key Pairs options ([Figure 2-4](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch02.html#aws_fig_key_pairs)).



###### *Figure 2-4. Key pair selection*

Key pairs provide secure access to your instances. To understand the benefits of key pairs, consider how we could securely give someone access to an AMI that anyone in the world can launch an instance of. Using default passwords would be a security risk, as it is almost certain some people would forget to change the default password at some point. Amazon has implemented SSH key pairs to help avoid this eventuality. Of course, it is possible to create an AMI that uses normal usernames and passwords, but this is not the default for AWS-supplied AMIs.

All AMIs have a default user: when an instance is booted, the public part of your chosen key pair is copied to that user’s SSH authorized keys file. This ensures that you can securely log in to the instance without a password. In fact, the only thing you need to know about the instance is the default username and its IP address or hostname.

This also means that only people with access to the private part of the key pair will be able to log in to the instance. Sharing your private keys is against security best practices, so to allow others access to the instance, you will need to create additional system accounts and configure them with passwords or SSH authorized keys.

###### NOTE

The name of the default user varies between AMIs. For example, Amazon’s own AMIs use ec2-user, whereas Ubuntu’s official AMIs use ubuntu.

If you are unsure of the username, one trick you can use is to try to connect to the instance as root. The most friendly AMIs present an error message informing you that root login is disabled, and letting you know which username you should use to connect instead.

Changing the default user of an existing AMI is not recommended, but can be easily done. The details of how to accomplish this have been documented by Eric Hammond of Alestic <https://alestic.com/2014/01/ec2-change-username/>.

| **Distribution** | **Default User Name** |
| --- | --- |
| Amazon Linux | ec2-user |
| Ubuntu | ubuntu |
| Debian | admin |
| RHEL | ec2-user (since 6.4), root (before 6.4) |
| CentOS | root |
| Fedora | ec2-user |
| SUSE | root |
| FreeBSD | ec2-user |
| BitNami | bitnami |

You can create a new SSH key pair through the [EC2 Key Pairs page](https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#KeyPairs:sort=keyName) in the AWS console-note that key pairs are region-specific, and this URL refers to the US East 1 region. Keys you create in one EC2 region cannot be immediately used in another region, although you can, of course, upload the same key to each region instead of maintaining a specific key pair for each region. After creating a key, a .pem file will be automatically downloaded.

Alternatively, you can upload the public part of an existing SSH key pair to AWS. This can be of great help practically because it may eliminate the need to add the -i /path/to/keypair.pem option to each SSH command where multiple keys are in use (refer to ssh-agent’s man page if you need to manage multiple keys). It also means that the private part of the key pair remains entirely private—you never need to upload this to AWS, it is never transmitted over the Internet, and Amazon does not need to generate it on your behalf, all of which have security implications.

Alestic offers a handy [Bash script](http://alestic.com/2010/10/ec2-ssh-keys) to import an existing public SSH key into each region.

###### TIP

If you are a Windows user connecting with PuTTY, you will need to convert this to a PPK file using PuTTYgen before you can use it. To do this, launch PuTTYgen, select Conversions → Import Key, and follow the on-screen instructions to save a new key in the correct format. Once the key has been converted, it can be used with PuTTY and PuTTY Agent.

From the Key Pairs screen in the launch wizard, you can select which key pair will be used to access the instance, or to launch the instance without any key pair. You can select from your existing key pairs or choose to create a new key pair. It is not possible to import a new key pair at this point—if you would like to use an existing SSH key that you have not yet uploaded to AWS, you will need to upload it first by following the instructions on the [EC2 Key Pairs](https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#KeyPairs:sort=keyName) page.

Once you have created a new key pair or imported an existing one, click “Choose from your existing Key Pairs,” select your key pair from the drop-down menu, and continue to the next screen. You have now completed the last step of the wizard—click Launch Instances to create the instance.

### WAITING FOR THE INSTANCE

Phew, we made it. Launching an instance can take a few seconds, depending on the instance type, current traffic levels on AWS, and other factors. The Instances page of the Management Console will show you the status of your new instance. Initially, this will be pending, while the instance is being created on the underlying physical hardware. Once the instance has been created and has begun the boot process, the page will show the running state. This does not mean your instance is servicing requests or ready for you to log in to, merely that the instance has been created.

Selecting an instance in the Management Console will show you its public DNS name, as well as more detail about the settings and status of the instance. At this point, you can try to SSH to the public hostname. If the connection fails, it means SSH is not yet ready to accept connections, so wait a moment and try again. Once you manage to log in to the instance, you will see a welcome screen specific to the AMI you launched.

### QUERYING INFORMATION ABOUT THE INSTANCE

Now that you have an instance, what can you do with it? The answer is—anything you can do with an equivalent Linux server running on physical hardware. Later chapters demonstrate some of the more useful things you can do with EC2 instances. For now, let’s take a look at the ec2metadata tool, which is included on most well-designed AMIs.

###### WARNING

In the infancy of AWS, EC2 had no real style guide; the question of how to name something was up to the developer. A few different but equivalent tools parsing instance metadata appeared, ec2metadata in the case of Ubuntu’s, and ec2-metadata in the case of Amazon Linux’s variant.

The ec2metadata tool is useful for quickly accessing metadata attributes of your instance: for example, the instance ID, or the ID of the AMI from which this instance was created. Running ec2metadata without any arguments will display all of the available metadata.

If you are interested in only one or two specific metadata attributes, you can read the values one at a time by passing the name of the attribute as a command-line option, for example:

$ **ec2metadata --instance-id**

i-ba932720

$ **ec2metadata --ami-id**

ami-c80b0aa2

This is useful if you are writing shell scripts that need to access this information. Rather than getting all of the metadata and parsing it yourself, you can do something like this:

INSTANCE\_ID=$(ec2metadata --instance-id)

AMI\_ID=$(ec2metadata --ami-id)

echo "The instance $INSTANCE\_ID was created from AMI $AMI\_ID"

###### NOTE

Where does the metadata come from? Every instance downloads its metadata from the following URL:

http://169.254.169.254/latest/meta-data/*attribute\_name*

So to get the instance ID, you could request the URLhttp://169.254.169.254/latest/meta-data/instance-id.

This URL is accessible only from within the instance, while the IP address maps to the hostname http://instance-data which is easier for users to remember. See AWS’ Documentation for full details on [instance metadata](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html)

If you want to query the metadata from outside the instance, you will need to use the ec2-describe-instances command.

### TERMINATING THE INSTANCE

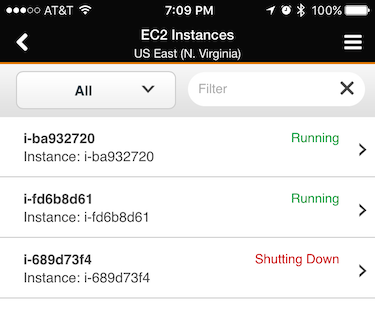
Once you have finished testing and exploring the instance, you can terminate it. In the Management Console, right-click the instance and select Terminate Instance.

Next, we will look at some of the other available methods of launching instances.

###### TIP

In early 2013, Amazon introduced a [mobile app](http://aws.amazon.com/console/mobile/) interface to the AWS Management Console with versions supporting both iOS and Android devices. After multiple updates and enhancements, the app has become an excellent tool for administrators who need a quick look at the state of their AWS deployment while on the move.

The app’s functionality is not as comprehensive as the web console’s, but it showcases remarkable usability in its streamlined workflow, and most users enjoy the quick access to select functionality it provides: some users now even pull up their mobile phone to execute certain tasks rather than resorting to their trusted terminal!



###### *Figure 2-5. The AWS Console mobile app*

## Launching with Command-Line Tools

If you followed the steps in the previous section, you probably noticed a few drawbacks to launching instances with the Management Console. The number of steps involved and the variety of available options engender complex documentation that takes a while to absorb. This is not meant as a criticism of the Management Console—EC2 is a complex beast, thus any interface to it requires a certain level of complexity.

Because AWS is a self-service system, it must support the use cases of many users, each with differing requirements and levels of familiarity with AWS itself. By necessity, the Management Console is equivalent to an enormous multipurpose device that can print, scan, fax, photocopy, shred, and collate.

This flexibility is great when it comes to discovering and learning the AWS ecosystem, but is less useful when you have a specific task on your to-do list that must be performed as quickly as possible. Interfaces for managing production systems should be streamlined for the task at hand, and not be conducive to making mistakes.

Documentation should also be easy to use, particularly in a crisis, and the Management Console does not lend itself well to this idea. Picture yourself in the midst of a downtime situation, where you need to quickly launch some instances, each with different AMIs and user data. Would you rather have to consult a 10-page document describing which options to choose in the Launch Instance Wizard, or copy and paste some commands into the terminal?

Fortunately, Amazon gives us precisely the tools required to do the latter. The EC2 command-line tools can be used to perform any action available from the Management Console, in a fashion that is much easier to document and much more amenable to automation.

If you have not already done so, you will need to set up the EC2 command-line tools according to the instructions in [“Preparing Your Tools”](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch01.html#preparing_your_tools) before continuing. Make sure you have set the AWS\_ACCESS\_KEY and AWS\_SECRET\_KEY environment variables or the equivalent values in the .aws/credentials file in your home directory.

##### ACCESS KEY IDS AND SECRETS

When you log in to the AWS Management Console, you will use your email address and password to authenticate yourself. Things work a little bit differently when it comes to the command-line tools. Instead of a username and password, you use an access key ID and secret access key. Together, these are often referred to as your access credentials.

Although access credentials consist of a pair of keys, they are not the same as an SSH key pair. The former is used to access AWS’ APIs, while the latter is used to SSH into an instance to perform work on the shell.

When you created your AWS account, you also generated a set of access credentials for your root account identity. These keys have full access to your AWS account—keep them safe! You are responsible for the cost of any resources created using these keys, so if a malicious person were to use these keys to launch some EC2 instances, you would be left with the bill.

[“IAM Users and Groups”](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch03.html#iam_users_and_groups) discusses how you can set up additional accounts and limit which actions they can perform, as defined by current security best practices. For the following examples, we will just use the access keys you have already created during CLI setup.

AWS lets you inspect all active access credentials for your account through the [Security Credentials page](https://portal.aws.amazon.com/gp/aws/securityCredentials) of the Management Console, but for increased security you will be unable to retrieve their secret access keys after creation. This stops any unuthorized access to your account from resulting in a compromise of your API credentials, but has the annoying side-effect of requiring you to replace your access keys if they ever were lost.

###### WARNING

As you start exploring dynamic infrastructure provisioning with AWS CLI, we recommend you [setup a billing alarm](http://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/free-tier-alarms.html). Leveraging the CloudWatch and Simple Notification services, billing alerts will notify you if you exceed pre-set spending thresholds.

While not ruinously expensive, forgetting to shut down a few of your test instances and letting them run for the rest of the month (until you notice as you are billed) will easily exceed your personal phone bill. It is a snap to inadvertedly make this mistake, we have slipped up ourselves and advise you let the system help keep track with these friendly notifications.

To launch an instance from the command line, you need to provide values that correspond to the options you can choose from when using the Management Console. Because all of this information must be entered in a single command, rather than gathered through a series of web pages, it is necessary to perform some preliminary steps so you know which values to choose. The Management Console can present you with a nice drop-down box containing all the valid AMIs for your chosen region, but to use the command line, you need to know the ID of the AMI before you can launch it.

The easiest way to get a list of available images is in the [Instances tab](https://console.aws.amazon.com/ec2/home?#s=Images)of the Management Console, which lets you search through all available AMIs. Keep in mind that AMIs exist independently in EC2 regions—the Amazon Linux AMI in the US East region is not the same image as the Amazon Linux AMI in Europe, although they are functionally identical. Amazon, Canonical, and other providers make copies of their AMIs available in each region as a convenience to their users, but the same AMI will show a different ID in different regions.

If you need to find an AMI using the command-line tools, you can do so with the aws ec2 describe-instances command. A few examples follow:

# Describe all of your own images in the US East region

aws ec2 describe-images --owners self --region us-east-1

# Find Amazon-owned images for Windows Server 2012, 64-bit version

aws ec2 describe-images --owners amazon --filters Name=architecture,Values=x86\_64 | grep Server-2012

# List the AMIs that have a specific set of key/value tags

aws ec2 describe-images --owners self --filters Name=tag:role,Values=webserver Name=tag:environment,Values=production

The first query should of course yield no results, unless you have already created some AMIs of your own. Later examples showcase combining the tool’s own filtering and grep to find the image you are really looking for. In our second example we are searching for a Windows Server image created by another party. Please note we explicitly searched for Amazon-owned images, as any AWS customer can decide to make her AMIs accessible to all other customers. Image names are freely chosen by their creator just like their contents, thus not only complicating our search with a very large number of results, but potentially posing a security problem if one carelessly selects an unknown parties’ bits.

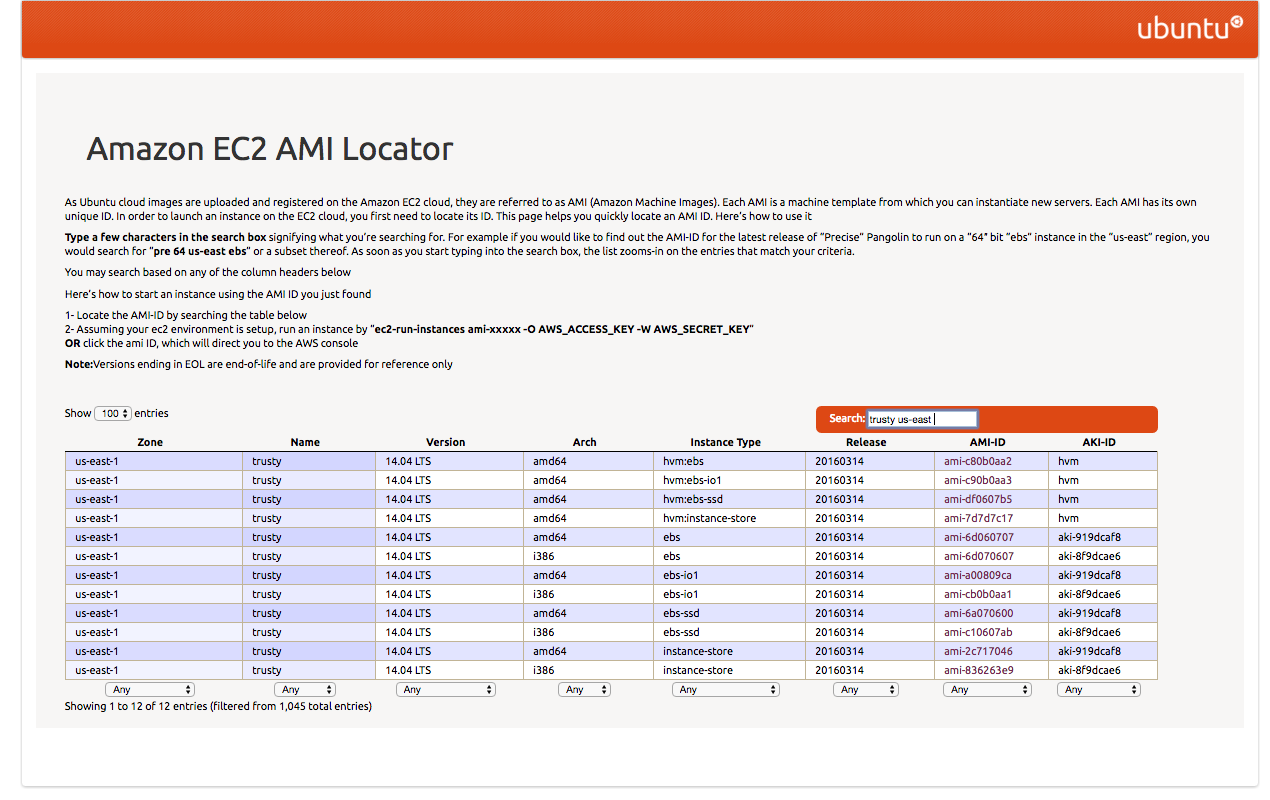
##### FINDING UBUNTU IMAGES

Searching for Ubuntu images yields 27,175 results in the us-east-1 region alone at the time of this writing. Filtering for official images released by Canonical (owner 099720109477) reduces the crop to only 6,074 images. These high numbers are due to Ubuntu’s high popularity in public cloud environments, and to Canonical’s committment to refreshing AMIs with the newest packages as security updates or bugfixes are published. Older AMIs remain available as new ones are issued by the vendor, the timing of when to switch to newer images being entirely under the admin’s control, not AWS’. All these factors conspire to make finding the correct Ubuntu image a rather non-trivial task.

Ubuntu AMIs can be most easily found using Canonical’s [AMI Locator](http://cloud-images.ubuntu.com/locator/ec2/), which shows only the most recent release by default and which updates results as you search by substring or select from pre-populated pulldown menus. This is an essential resource for navigating the sea of Ubuntu images found on AWS. At the time of this writing, the Locator narrows down our options to twelve images varying in storage and system bit width.

Equally interesting to power users is the collection of [official Ubuntu cloud images found on Ubuntu.com](http://cloud-images.ubuntu.com/). This site includes both daily builds and official releases. Finding the latter is accomplished by navigating to the release/ subdirectory of any Ubuntu version, which is http://cloud-images.ubuntu.com/releases/14.04/release/ for Trusty.

At the time of writing, the latest stable Ubuntu long-term support (LTS) version is 14.04.4, going by the nickname Trusty Tahr. In the Eastern US EC2 region, the latest version of Canonical’s official AMI is ami-c80b0aa2 (64b, HVM, EBS storage), which is used in the examples. Make sure to update this with your chosen AMI. If you are not sure which to use, or have no real preference, the authors recommend using the latest LTS version of Ubuntu for 64-bit systems.



###### *Figure 2-6. Ubuntu EC2 AMI Locator. Clicking the selected AMI ID launches it from the Management Console*

The command used to launch an instance is aws ec2 run-instances. The most basic invocation is simply aws ec2 run-instances--image-id ami-6d060707, which will launch an older m1.small instance in the default us-east-1 region. If you are paying attention, you noticed we used a different AMI ID with paravirtualization support as the older m1.small instance type does not support the newer HVM virtualization style. However, if you run this command and attempt to log in to the instance, you will soon notice a rather large problem: because no key pair name was specified, there is no way to log in to the instance. Instead, try running the command with the -key option to specify one of the SSH key pairs you created earlier. In the following example, we have also changed the instance type to t2.micro , the smallest instance type all AWS operating systems are currently confortable with:

$ **aws ec2 run-instances --image-id ami-c80b0aa2 --region us-east-1 \**

**--key federico --instance-type t2.micro --output text**

740376006796 r-bbcfff10

INSTANCES 0 x86\_64 False xen ami-c80b0aa2 i-64a8a6fe t2.micro federico 2016-04-03T07:40:48.000Z ip-172-31-52-118.ec2.internal 172.31.52.118 /dev/sda1 ebs True subnet-2a45b400 hvm vpc-934935f7

[ output truncated ]

Once EC2 receives the request to launch an instance, it prints some information about the pending instance. The value we need for the next command is the instance ID, in this case,i-64a8a6fe.

Although this command returns almost immediately, you will still need to wait a short while before your instance is ready to accept SSH connections. You can check on the status of the instance while it is booting with the aws ec2 describe-instance-status command. While the instance is still booting, its status will be listed as pending. This will change to running once the instance is ready. Remember that ready in this context means that the virtual instance has been created, and the operating system’s boot process has started. It does not necessarily mean that the instance is ready to receive an SSH connection, which is important when writing scripts that automate these commands.

###### TIP

Granting access to an already running image can involve multiple manual steps adding the new user’s SSH credentials to the authorized keys file. Juggling files can be avoided working with Ubuntu images thanks to the ssh-import-id command. Just invoking:

ssh-import-id lp:f2

Will retrieve Federico’s SSH identity from Launchpad.net and grant him access as the instance’s user the command was run under. You can accomplish the same for Mike by using his Github.com user id:

ssh-import-id gh:mikery

All that is required is the user id from either site. This is roughly equivalent to running wget https://launchpad.net/~f2/+sshkeys -O - >> ~/.ssh/authorized\_keys && echo >> ~/.ssh/authorized\_keys, which could be used to derive alternative import strategies for other sites.

When your instance is running, the output should look similar to this:

$ **aws ec2 describe-instance-status --instance-ids i-64a8a6fe --region us-east-1 --output text**

INSTANCESTATUSES us-east-1a i-64a8a6fe

INSTANCESTATE 16 running

INSTANCESTATUS ok

DETAILS reachability passed

SYSTEMSTATUS ok

DETAILS reachability passed

Another way to display information about your instance is with aws ec2 describe-instances, which will show much more detail. In particular, it will show the public DNS name (for example, ec2-54-247-40-225.eu-west-1.compute.amazonaws.com), which you can use to SSH into your instance.

$ **aws ec2 describe-instances --instance-ids i-64a8a6fe --region us-east-1 --output text**

RESERVATIONS 740376006796 r-bbcfff10

INSTANCES 0 x86\_64 False xen ami-c80b0aa2 i-64a8a6fe t2.micro federico 2016-04-03T07:40:48.000Z ip-172-31-52-118.ec2.internal 172.31.52.118 ec2-52-90-56-122.compute-1.amazonaws.com 52.90.56.122 /dev/sda1 ebs True subnet-2a45b400 hvm vpc-934935f7

BLOCKDEVICEMAPPINGS /dev/sda1

[ output truncated ]

EBS 2016-04-03T07:40:48.000Z True attached vol-e9c0c637

MONITORING disabled

NETWORKINTERFACES 12:5a:33:b3:b5:97 eni-ce4084ea 740376006796 ip-172-31-52-118.ec2.internal 172.31.52.118 True in-use subnet-2a45b400 vpc-934935f7

ASSOCIATION amazon ec2-52-90-56-122.compute-1.amazonaws.com 52.90.56.122

ATTACHMENT 2016-04-03T07:40:48.000Z eni-attach-2545d3d4 True 0 attached

GROUPS sg-384f3a41 default

PRIVATEIPADDRESSES True ip-172-31-52-118.ec2.internal 172.31.52.118

ASSOCIATION amazon 52.90.56.122

PLACEMENT us-east-1a default

SECURITYGROUPS sg-384f3a41 default

STATE 16 running

To terminate the running instance, issue aws ec2 terminate-instance. To verify that this instance has indeed been terminated, you can use the aws ec2 describe-instances command again:

$ **aws ec2 terminate-instances --instance-ids i-64a8a6fe --region us-east-1**

INSTANCE i-fc2067b7 running shutting-down

$ **aws ec2 describe-instances --instance-ids i-64a8a6fe --region us-east-1**

RESERVATION r-991230d1 612857642705 default

INSTANCE i-fc2067b7 ami-00b11177 terminated mike 0 t1.micro 2012-11-25T15:51:45+0000

[ output truncated ]

As you find yourself using the command-line tools more frequently, and for more complex tasks, you will probably begin to identify procedures that are good candidates for automation. Besides saving you both time and typing, automating the more complex tasks has the additional benefits of reducing the risk of human error and simply removing some thinking time from the process.

The command-line tools are especially useful when it comes to documenting these procedures. Processes become more repeatable. Tasks can be more easily delegated and shared among the other members of the team.

## Launching from Your Own Programs and Scripts

The command-line tools are useful from an automation perspective, as it is trivial to call them from Bash or any other scripting language. While the output for some of the services can be rather complex, it is relatively straightforward to parse this output and perform dynamic actions based on the current state of your infrastructure. At a certain level of complexity, though, calling all of these external commands and parsing their output becomes time-consuming and error prone. At this point, it can be useful to move to a programming language with a client library to help you work AWS directly.

Officially supported client libraries are available for many programming languages and platforms, including:

* Java
* PHP
* Python
* Ruby
* .NET
* iOS
* Android

The full set of AWS programming resources can be found at the [AWS Sample Code](http://aws.amazon.com/code) site.

Most of the examples in this book use the popular Python-based [Boto library](http://boto.cloudhackers.com/en/latest/index.html) although other, equally capable libraries exist. Even if you are not a Python developer, the examples should be easy to transfer to your language of choice, because each library is calling the same underlying AWS API.

Regardless of your language choice, the high-level concepts for launching an instance remain the same: first, decide which attributes you will use for the instance, such as which AMI it will be created from, and then issue a call to the RunInstances method of the EC2 API.

When exploring a new API from Python, it can often be helpful to use the interactive interpreter. This lets you type in lines of Python code one at a time, instead of executing them all at once in a script. The benefit here is that you have a chance to explore the API and quickly get to grips with the various functions and objects that are available. We will use this method in the upcoming examples. If you prefer, you can also copy the example code to a file and run it all in one go with python *filename.py*.

If you do not already have the Boto library installed, you will need to install it with pip (pip install boto) before continuing with the examples. Once this is done, open the Python interactive interpreter by running python without any arguments:

$ **python**

Python 2.7.6 (default, Jun 22 2015, 17:58:13)

[GCC 4.8.2] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>>

When you connect to an AWS service with Boto, Boto needs to know which credentials it should use to authenticate. You can explicitly pass the aws\_access\_key\_id andaws\_secret\_access\_key keyword arguments when calling connect\_to\_region, as shown here:

>>> **AWS\_ACCESS\_KEY\_ID = "your-access-key"**

>>> **AWS\_SECRET\_ACCESS\_KEY = "your-secret-key"**

>>> **from boto.ec2 import connect\_to\_region**

>>> **ec2\_conn = connect\_to\_region('us-east-1',**

... **aws\_access\_key\_id=AWS\_ACCESS\_KEY\_ID,**

... **aws\_secret\_access\_key=AWS\_SECRET\_ACCESS\_KEY)**

Alternatively, if the AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY environment variables are set, Boto will use these automatically:

$ **export AWS\_SECRET\_ACCESS\_KEY='*your access key*'**

$ **export AWS\_ACCESS\_KEY\_ID='*your secret key*'**

$ **python**

Python 2.7.6 (default, Jun 22 2015, 17:58:13)

[GCC 4.8.2] on linux2

Type "help", "copyright", "credits" or "license" for more information.

>>> **from boto.ec2 import connect\_to\_region**

>>> **ec2\_conn = connect\_to\_region('us-east-1')**

Boto will also automatically attempt to retrieve your credentials from the file ~/.aws/credentials if one is present, in which case exporting them to environment variables is not necessary.

Once you have connected to the EC2 API, you can issue a call to run\_instances to launch a new instance. You will need two pieces of information before you can do this: the ID of the AMI you would like to launch, and the name of the SSH key pair you will use when connecting to the instance:

>>> **ssh = ec2\_conn.create\_security\_group('ssh', 'SSH access group')**

>>> **ssh**

SecurityGroup:ssh

>>> **ssh.authorize('tcp', 22, 22, '0.0.0.0/0')**

True

>>> **reservation = ec2\_conn.run\_instances('ami-c80b0aa2',**

... **instance\_type='t2.micro', key\_name='*your-key-pair-name*',**

... **security\_group\_ids=['ssh'])**

>>> **instance = reservation.instances[0]**

The call to run\_instances does not, as might initially be suspected, return an object representing an instance. Because you can request more than one instance when calling the run\_instances function, it returns a reservation, which is an object representing one or more instances. The reservation object lets you iterate over the instances. In our example, we requested only one instance, so we simply took the first element of the list of instances in the reservation (in Python, that is done with reservation.instances[0]) to get our instance.

Now the instance is launching, and we have an instance (in the programming sense) of the instance (in the EC2 sense), so we can begin to query its attributes. Some of these are available immediately, whereas others do not get set until later in the launch process. For example, the DNS name is not available until the instance is nearly running. The instance will be in the pending state initially. We can check on the current state by calling the update()function:

>>> **instance.state**

u'pending'

>>> **instance.update()**

u'pending'

# After some time…

>>> **instance.update()**

u'running'

Once the instance reaches the running state, we should be able to connect to it via SSH. But first we need to know its hostname or IP address, which are available as attributes on the instance object:

>>> **instance.public\_dns\_name**

u'ec2-54-152-96-69.compute-1.amazonaws.com'

>>> **instance.private\_ip\_address**

u'172.31.51.214'

>>> **instance.id**

u'i-53f2e7c9'

Terminating a running instance is just a matter of calling the terminate() function. Before we do that, let’s take a moment to look at how Boto can work with EC2 tags to help make administration easier. A tag is a key/value pair that you can assign to any number of instances to track arbitrary properties. The metadata stored in tags can be used as a simple but effective administration database for your EC2 resources. Setting a tag is simple:

>>> **ec2\_conn.create\_tags([instance.id], {'environment': 'staging'})**

True

Once an instance has been tagged, we can use the get\_all\_instances() method to find it again. get\_all\_instances() returns a list of reservations, each of which, in turn, contains a list of instances. These lists can be iterated over to perform an action on all instances that match a specific tag query. As an example, we will terminate any instances that have been tagged as being part of our staging environment:

>>> **tagged\_reservations = ec2\_conn.get\_all\_instances(filters={'tag:environment': 'staging'})**

>>> **tagged\_reservations**

[Reservation:r-6a4a76c1]

>>> **tagged\_reservations[0]**

Reservation:r-6a4a76c1

>>> **tagged\_reservations[0].instances[0]**

Instance:i-53f2e7c9

>>> **for res in tagged\_reservations:**

...  **for inst in res.instances:**

...  **inst.terminate()**

>>>

###### NOTE

Given that nearly all resource types support tagging, and that Amazon provides this feature free of charge, it would be a shame not to take advantage of the many ways this can help you automate and control your infrastructure. Think of it as an incredibly simple query language for your infrastructure. Conceptually, our previous example was similar to SELECT \* FROM instances WHERE tag\_environment='staging'.

The previous example iterated over all the matching instances (only one, in this case) and terminated them. We can now check on the status of our instance and see that it is heading toward the terminated state.

>>> **instance.update()**

u'shutting-down'

# After a moment or two…

>>> **instance.update()**

u'terminated'

This example only scratches the surface of what Boto and other client libraries are capable of. The [Boto documentation](http://boto.cloudhackers.com/en/latest/ref/) provides a more thorough introduction to other AWS services. Having the ability to dynamically control your infrastructure is one of the best features of AWS from a system administration perspective, and it gives you plenty of opportunities to automate recurring processes.

###### NOTE

Managing AWS with Python is the subject of Mitch Garnaat’s [Python and AWS Cookbook](http://shop.oreilly.com/product/0636920020202.do) (O’Reilly). Written by the very author of Boto, this cookbook gets you started with more than two dozen complete recipes.

###### WARNING

Are there limits on just how many servers you can dynamically request from AWS? New accounts are usually limited to 20 on-demand instances, an additional 20 reserved instances, and up to [20](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-limits.html#spot-limits-general) spot instances in each region. [Additional restrictions](http://aws.amazon.com/ec2/faqs/#How_many_instances_can_I_run_in_Amazon_EC2) are enforced on certain instance types, and filing a [support request](https://console.aws.amazon.com/support/home?region=us-east-1#/case/create?issueType=service-limit-increase&limitType=service-code-ec2-instances&serviceLimitIncreaseType=ec2-instances&type=service_limit_increase) is all that is necessary to increase these limits.

# Introducing CloudFormation

There is another method of launching instances that deserves its own section. Among the many Amazon Web Services features, a favorite is CloudFormation. It fundamentally changes how AWS infrastructure is managed, and is something whose absence is strongly felt when working in non-AWS environments. In a nutshell, CloudFormation is a resource-provisioning tool that accepts a JSON file describing the resources you require and then creates them for you. Such a simple idea, yet so powerful.

Consider this example checklist for launching an instance. Using the three methods of launching instances we have already looked at, how could you most efficiently perform these tasks? More importantly, how would you document the process so it is repeatable?

1. Launch a t2.micro instance of ami-c80b0aa2 in the us-east-1 region. The instance should have a 10 GB EBS volume attached to the sdf device and belong to the security group named webservers. It should be given the string webserver as user data and have a role tag with the value of webserver.
2. Create a CNAME for www.example.com that points to the public hostname of the instance.

If the task is a one-off procedure, it might make sense to perform it using the Management Console, but the documentation would be time-consuming to write and tedious to follow. Automating the task through programming (either by calling the EC2 command-line tools, or using one of the client libraries) means the documentation could be reduced to a single command: “run this script.” While benefiting the person following the documentation, this comes at a cost to whomever must write and maintain the script.

Using CloudFormation, the burden of maintaining the tool itself is shifted to Amazon, with the user retaining responsability solely for maintaining the configuration itself. You simply create a JSON-formatted file (a stack template) describing the attributes of the instance, and then let AWS do the rest. The documentation is reduced to one step: “Create a stack named webservers, using the stack template webserver.json.” A stack can be thought of as a collection of resources, along with a list of events associated with changes to those resources and the stack itself.

Successfully submitting a stack template to CloudFormation will result in the creation of a stack, which will, in turn, create one or more AWS resources (such as EC2 instances or Elastic Load Balancers).There are no additional scripts to write or maintain, although writing and maintaining stack templates can become rather complicated as well once your infrastructure starts growing. The CloudFormation stack template language has its own learning curve.

Being plain-text files, stack templates can be stored in your revision control system alongside your application code and server configurations. The same processes used to review changes to your code can be applied to changes in your infrastructure. By browsing the history of commits to your stack templates, you can quickly audit changes to your infrastructure, as long as you have a consistent policy in place to run stacks only after they have been committed to version control.

An additional benefit of stack templates is that they can be reused: it is possible to create multiple stacks from the same template. This can be used to give each developer a self-contained copy of their development stack. When new members join the team, they simply need to launch a new copy of the stack, and they can start familiarizing themselves with the application and infrastructure almost immediately.

The same stack template can also be used to create multiple copies of the stack in the different AWS regions. Operating an application across multiple AWS regions requires a lot of careful planning at both the application and infrastructure layers, but CloudFormation makes one aspect of the task very easy: by deploying a stack template to multiple regions, you can be sure that your infrastructure is identical in each region, without needing to manually configure a series of resources in each one.

Aside from the cost of the underlying resources, CloudFormation is free of charge. Although it adds a small bump in the AWS learning curve, it is well worth taking the time to deploy your infrastructure with CloudFormation, especially if you find yourself managing complicated or frequently changing infrastructures. Routing all changes to your infrastructure through a single process (i.e., updating the CloudFormation stack) is imperative when working with a team, as it gives you an easy way to answer those questions of “who changed what, and when.”

For more examples of what can be achieved with CloudFormation, have a look at the [example templates provided by Amazon](http://aws.amazon.com/cloudformation/aws-cloudformation-templates/).

# Working with CloudFormation Stacks

CloudFormation stacks are themselves a type of AWS resource, and can thus be managed in similar ways. They can be created, updated, and deleted via the same methods we use for interacting with other AWS services—the Management Console, command-line tools, or client libraries. They can also be tagged for ease of administration.

## Creating the Stack

In this section, we will start with a basic stack template that simply launches an EC2 instance. [Example 2-1](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch02.html#cloudformation_simple) shows one of the simplest CloudFormation stacks.

##### ***Example 2-1. Basic CloudFormation Stack in JSON***

{

"AWSTemplateFormatVersion" : "2010-09-09",

"Description" : "A simple stack that launches an instance.",

"Resources" : {

"Ec2Instance" : {

"Type" : "AWS::EC2::Instance",

"Properties" : {

"InstanceType": "t2.micro",

"ImageId" : "ami-c80b0aa2"

}

}

}

}

###### WARNING

CloudFormation requires stack templates to be strictly valid JSON, so keep an eye out for trailing commas when copying or modifying templates.

Templates can be validated and checked for errors with the AWS command-line tool, for example:

aws cloudformation validate-template \

--template-body file://MyStack.json

Some editors, including Eclipse and Vim, can be extended with plug-ins to help produce and validate JSON files.

The Resources section is an object that can contain multiple children, although this example includes only one (EC2Instance). The EC2Instance object has attributes that correspond to the values you can choose when launching an instance through the Management Console or command-line tools.

CloudFormation stacks can be managed through the Management Console, with the command-line tools, or with scripts leveraging client-side libraries such as Boto.

One advantage of using the Management Console is that a list of events is displayed in the bottom pane of the interface. With liberal use of the refresh button, this will let you know what is happening when your stack is in the creating or updating stages. Any problems encountered while creating or updating resources will also be displayed here, which makes it a good place to start when debugging CloudFormation problems. These events can also be read by using the command-line tools, but the Management Console output is a much more friendly human interface.

It is not possible to simply paste the stack template file contents into the Management Console. Rather, you must create a local text file and upload it to the Management Console when creating the stack. Alternatively, you can make the stack accessible on a website and provide the URL instead. The same applies when using the command-line tools and API.

To see the example stack in action, copy the JSON shown in [Example 2-1](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch02.html#cloudformation_simple) into a text file. You may need to substitute the AMI (ami-c80b0aa2) with the ID of an AMI in your chosen EC2 region (our pre-set value comes from the ever-popular default us-east-1). Use the command-line tools or Management Console to create the stack. Assuming you have stored your stack template in a file named example-stack.json, you can create the stack with this command:

aws cloudformation create-stack --template-body file://example-stack.json \

--stack-name example-stack

If your JSON file is not correctly formed, you will see a helpful message letting you know the position of the invalid portion of the file. If CloudFormation accepted the request, it is now in the process of launching an EC2 instance of your chosen AMI. You can verify this with the aws cloudformation describe-stack-resources and aws cloudformation describe-stack-events commands:

$ **aws cloudformation describe-stack-events--stack-name example-stack \**

**--output text**

STACKEVENTS 9b5ea230-fcc7-11e5-89de-500c217b26c6 example-stack arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 CREATE\_COMPLETE AWS::CloudFormation::Stack arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 example-stack 2016-04-07T13:49:53.884Z

STACKEVENTS Ec2Instance-CREATE\_COMPLETE-2016-04-07T13:49:52.222Z Ec2Instance i-ebe00376 {"ImageId":"ami-c80b0aa2","InstanceType":"t2.micro"}

CREATE\_COMPLETE AWS::EC2::Instance arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758dexample-stack 2016-04-07T13:49:52.222Z

STACKEVENTS Ec2Instance-CREATE\_IN\_PROGRESS-2016-04-07T13:49:05.313Z Ec2Instance i-ebe00376 {"ImageId":"ami-c80b0aa2","InstanceType":"t2.micro"}

CREATE\_IN\_PROGRESS Resource creation Initiated AWS::EC2::Instance arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 example-stack 2016-04-07T13:49:05.313Z

STACKEVENTS Ec2Instance-CREATE\_IN\_PROGRESS-2016-04-07T13:49:04.113Z Ec2Instance {"ImageId":"ami-c80b0aa2","InstanceType":"t2.micro"}

CREATE\_IN\_PROGRESS AWS::EC2::Instance arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 example-stack 2016-04-07T13:49:04.113Z

STACKEVENTS 7b1fc800-fcc7-11e5-a700-50d5cd2758d2 example-stack arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 CREATE\_IN\_PROGRESS User Initiated AWS::CloudFormation::Stack arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 example-stack 2016-04-07T13:48:59.905Z

$ **aws cloudformation describe-stack-resources --stack-name example-stack \**

**--output text**

STACKRESOURCES Ec2Instance i-ebe00376 CREATE\_COMPLETE AWS::EC2::Instance arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 example-stack 2016-04-07T13:49:52.222Z

aws cloudformation describe-stack-events prints a list of events associated with the stack, in reverse chronological order. Following the chain of events, we can see that AWS first created the stack, and then spent around 60 seconds creating the instance, before declaring the stack creation complete. The second command shows us the ID of the newlylaunched instance: i-5689cc1d.

## Updating the Stack

Updating a running stack is an equally straightforward task. But before doing this, a brief digression into the way AWS handles resource updates is called for.

Some attributes of AWS resources cannot be modified once the instance has been created. Say, for example, you launch an EC2 instance with some user data. You then realize that the User Data was incorrect, so you would like to change it. Although the Management Console provides the option to View/Change User Data, the instance must be in the stopped state before the user data can be modified. This means you must stop the instance, wait for it to enter the stopped state, modify the user data, and then start the instance again.

This has an interesting implication for CloudFormation. Using the previous example, imagine you have a CloudFormation stack containing an EC2 instance that has some user data. You want to modify the user data, so you update the stack template file and run the aws cloudformation update-stack command. Because CloudFormation is unable to modify the user data on a running instance, it must instead either reload or replace the instance, depending on whether this is an EBS-backed or instance store−backed instance.

If this instance was your production web server, you would have had some unhappy users during this period. Therefore, making changes to your production stacks requires some planning to ensure that you won’t accidentally take down your application. We won’t list all of the safe and unsafe types of updates here, simply because there are so many permutations to consider that it would take an additional book to include them all. The simplest thing is to try the operation you want to automate by using the Management Console or command-line tools to find out whether they require stopping the server.

We already know that user data cannot be changed without causing the instance to be stopped, because any attempt to change the user data of a running instance in the Management Console will fail. Conversely, we know that instance tags can be modified on a running instance via the Management Console; therefore, updating instance tags with CloudFormation does not require instance replacement.

Changing some other attributes, such as the AMI used for an instance, will also require the instance to be replaced. CloudFormation will launch a new instance using the update AMI ID and then terminate the old instance. Obviously, this is not something you want to do on production resources without taking care to ensure that service is not disrupted while resources are being replaced. Mitigating these effects is discussed later, when we look at Auto Scaling and launch configurations.

If in doubt, test with an example stack first. CloudFormation lets you provision your infrastructure incredibly efficiently—but it also lets you make big mistakes with equal efficiency. With great power (which automation offers) comes great responsibility.

###### WARNING

Be careful when updating stacks that provide production resources. Once you submit the request to modify the stack, there is no going back. Furthermore, you cannot request any additional changes until the update is complete, so if you accidentally terminate all of your production resources, you will have no option but to sit back and watch it happen, after which you can begin re-creating the stack as quickly as possible.

To remove any doubt, review the CloudFormation documentation for the resource type you are modifying. The documentation will let you know if this resource can be updated in place, or if a replacement resource is required in order to apply changes.

To see this in action, we will first update the instance to include some tags. Update the example-stack.json file so that it includes the following line in bold—note the addition of the comma to the end of the first line:

…

"InstanceType": "t2.micro"**,**

**"Tags": [ {"Key": "foo", "Value": "bar"}]**

}

…

Now we can update the running stack with aws cloudformation update-stackand watch the results of the update process with aws cloudformation describe-stack-events:

$ **aws cloudformation update-stack --template-body file://example-stack.json \**

**--stack-name example-stack --output text**

arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2

$ **aws cloudformation describe-stack-events --stack-name example-stack \**

**--output text**

STACKEVENTS beaeaca0-fcca-11e5-a119-500c2866f062 example-stack arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 UPDATE\_COMPLETE AWS::CloudFormation::Stack arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 example-stack 2016-04-07T14:12:21.550Z

STACKEVENTS bda21ea0-fcca-11e5-b6dd-50d5ca6e60ae example-stack arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 UPDATE\_COMPLETE\_CLEANUP\_IN\_PROGRESS AWS::CloudFormation::Stack arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 example-stack 2016-04-07T14:12:19.911Z

STACKEVENTS Ec2Instance-UPDATE\_COMPLETE-2016-04-07T14:12:18.229Z Ec2Instance i-ebe00376 {"ImageId":"ami-c80b0aa2","Tags":[{"Value":"bar","Key":"foo"}],"InstanceType":"t2.micro"}

UPDATE\_COMPLETE AWS::EC2::Instance arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758dexample-stack 2016-04-07T14:12:18.229Z

STACKEVENTS Ec2Instance-UPDATE\_IN\_PROGRESS-2016-04-07T14:12:02.354Z Ec2Instance i-ebe00376 {"ImageId":"ami-c80b0aa2","Tags":[{"Value":"bar","Key":"foo"}],"InstanceType":"t2.micro"}

UPDATE\_IN\_PROGRESS AWS::EC2::Instance arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 example-stack 2016-04-07T14:12:02.354Z

STACKEVENTS b0537140-fcca-11e5-bc1e-500c286f3262 example-stack arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 UPDATE\_IN\_PROGRESS User Initiated AWS::CloudFormation::Stack arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 example-stack 2016-04-07T14:11:57.557Z

$ **aws cloudformationdescribe-stack-resources --stack-name example-stack \**

**--output text**

STACKRESOURCES Ec2Instance i-ebe00376 UPDATE\_COMPLETE AWS::EC2::Instance arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2 example-stack 2016-04-07T14:12:18.229Z

Finally, the aws ec2 describe-tags command will show that the instance is now tagged with foo=bar:

$ **aws ec2 describe-tags --filters Name=resource-type,Values=instance \**

**Name=resource-id,Values=i-ebe00376 --output text**

TAGS aws:cloudformation:logical-id i-ebe00376 instance Ec2Instance

TAGS aws:cloudformation:stack-id i-ebe00376 instance arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/7b1d5700-fcc7-11e5-a700-50d5cd2758d2

TAGS aws:cloudformation:stack-name i-ebe00376 instance example-stack

TAGS foo i-ebe00376 instance bar

Notice the additional tags in the aws:cloudformation namespace. When provisioning resources which support tagging, CloudFormation will automatically apply tags to the resource. These tags let you keep track of that stack, which “owns” each resource, and make it easy to find CloudFormation-managed resources in the Management Console.

## Looking Before You Leap

When you have more than one person working on a stack template, it can be easy to find yourself in a situation where your local copy of the stack template does not match the template used by the running stack.

Imagine that two people are both making changes to a stack template stored in a Git repository. If one person makes a change and updates the stack without committing that change to Git, the next person to make an update will be working with an out-of-date stack template. The next update will then revert the previous changes, which, as previously discussed, could have negative consequences. This is a typical synchronization problem whenever you have two independent activities that could be happening concurrently: in this case, updating Git and updating the actual AWS stack.

Happily, Amazon has provided a tool that, in combination with a couple of Linux tools, will let you make certain that your local copy of the stack does indeed match the running version. Use the aws cloudformation get-template command to get a JSON file describing the running template, clean the output with sed and head, and finally use diff to compare the local and remote versions. If we did this before updating the example stack to include tags, we would have obtained the following results:

$ **aws cloudformation get-template --stack-name example-stack \**

**| grep -v "TemplateBody" | head -n -1 > example-stack.running**

$ **diff <(jq '.' example-stack.running) <(jq '.' example-stack.json)**

9c9,10

4a5,10

< "Tags": [

< {

< "Value": "bar",

< "Key": "foo"

< }

< ],

###### WARNING

We use jq to pretty-print the JSON in a consistent format that saves difffrom getting caught up in formatting or whitespace differences. Element ordering is still a factor however, one easily addressed by making your initial git commit of a new stack with the format exported by aws cloudformation get-template.

These commands could be wrapped in a simple script to save typing. Changes to production CloudFormation stacks should always be preceded by a check like this, especially if working in a team. This check should be incorporated into the script used for updating the stack: if it happens automatically, there is no chance of forgetting it.

## Deleting the Stack

Deleting a running stack will, by default, result in the termination of its associated resources. This is quite frequently the desired behavior, so it makes for a sensible default, but at times, you would like the resources to live on after the stack itself has been terminated. This is done by setting the DeletionPolicy attribute on the resource. This attribute has a default value of Delete.

All resource types also support the Retain value. Using this means that the resource will not be automatically deleted when the stack is deleted. For production resources, this can be an added safety net to ensure that you don’t accidentally terminate the wrong instance. The downside is that, once you have deleted the stack, you will need to manually hunt down the retained resources if you want to delete them at a later date.

The final option for the DeletionPolicy attribute is Snapshot, which is applicable only to the subset of resources that support snapshots. Currently, these include the Relational Database Service (RDS) database instances, Amazon Redshift data warehouse clusters, and EBS volumes. With this value, a snapshot of the database or volume will be taken when the stack is terminated.

Remember that some resources will be automatically tagged with the name of the CloudFormation stack to which they belong. This can save some time when searching for instances that were created with the Retain deletion policy.

Deleting a stack is done with the aws cloudformation delete-stack command. Again, you can view the changes made to the stack with aws cloudformation describe-stack-events:

$ **aws cloudformation delete-stack \**

**--stack-name example-stack**

$ **aws cloudformation describe-stack-events --stack-name example-stack --output text**

STACKEVENTS Ec2Instance-DELETE\_IN\_PROGRESS-2016-04-08T07:22:22.919Z Ec2Instance i-05fe1098 {"ImageId":"ami-c80b0aa2","InstanceType":"t2.micro"}

DELETE\_IN\_PROGRESS AWS::EC2::Instance arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/76b64990-fd5a-11e5-9750-500c286f3262 example-stack 2016-04-08T07:22:22.919Z

[ output truncated ]

Events are available only until the stack has been deleted. You will be able to see the stack while it is in the DELETE\_IN\_PROGRESS state, but once it has been fully deleted, aws cloudformation describe-stack-events will fail.

# Which Method Should I Use?

As we have already seen, AWS provides a lot of choices. When deciding which method is best for your use case, there are several things to consider. One of the most important is the return on investment of any effort spent automating your system administration tasks.

The main factors to consider are as follows:

* How frequently is the action performed?
* How difficult is it?
* How many people will have to perform it?

If you are part of a small team that does not make frequent changes to your infrastructure, the Management Console might be all you need. Personal preference will also play a part: some people are more at home in a web interface than they are on the command line. Once you have a complicated infrastructure or a larger team, it becomes more important that processes are documented and automated, which is not a strong point of the Management Console.

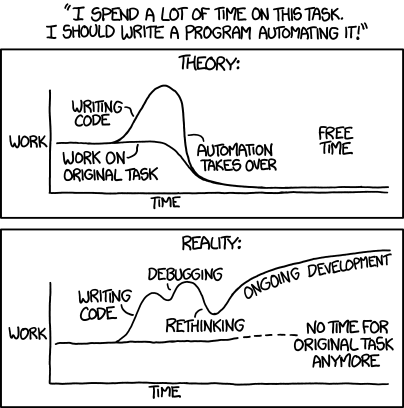
For production services, we cannot recommend using CloudFormation strongly enough. Given the benefits outlined in the previous section—an audit trail, stack templates stored in source control—how could any sysadmin not immediately fall in love with this technology? Unless you have a compelling reason not to, you should be using CloudFormation for any AWS resources that are important to your infrastructure.

Mike’s golden rule for any infrastructure he is responsible for is “If it’s in production, it’s in Git.” Meaning that if a resource—application code, service configuration files, and so on—is required for that infrastructure to operate, it must be under version control. CloudFront ties into this philosophy perfectly.

No matter how useful CloudFormation is, at times you will need to perform tasks that fall outside its capabilities. For these occasions, some combination of the command-line tools and client libraries are the next best thing in terms of ease of documentation and automation.

Combining the AWS client libraries with existing system management software can be a powerful tool. Packages such as [Fabric](http://fabfile.org/) (Python) and [Capistrano](https://github.com/capistrano/capistrano/wiki) (Ruby) make it easy to efficiently administer large numbers of systems. By combining these with the respective language’s client library, you can use them to administer a fleet of EC2 instances.

Automating too early can waste as much time as automating too late, as demonstrated in [Figure 2-7](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch02.html#xkcd_automation). Especially at the beginning of a project, processes can change frequently, and updating your automation scripts each time can be a drain on resources. For this reason, I recommend using the Management Console when first learning a new AWS service—once you have performed the same task a few times, you will have a clear idea of which tasks will provide the most “automation ROI.”



###### *Figure 2-7.*[*XKCD’s take on automation*](https://xkcd.com/1319/)*, courtesy of Randall Munroe*

###### NOTE

If you are interested in strategies to help decide which tasks warrant automation, the authors would like to recommend [Time Management for System Administrators](http://shop.oreilly.com/product/9780596007836.do) by Thomas A. Limoncelli (O’Reilly).

Regardless of which method you choose, it is important to have a documented process describing how to perform updates. Errors will inevitably occur if there is no policy in place to organize everyone in a coherent fashion.

# Amazon Machine Images

Some AMIs are virtual appliances—preconfigured server images running a variety of operating systems and software stacks. Amazon provides a number of its own images, running open source and commercial software, and allows any third-party to distribute their images through the AWS Marketplace. You can also create your own images, configured exactly to meet your requirements, and share them with a few selected accounts or choose to make them public altogether.

Building your own AMIs has a number of benefits. You get to customize the software selection and configure which services will start when the instance is launched. Any services that are not required can be disabled to cut down on wasted resources. Later chapters show how to launch instances automatically in response to external conditions such as traffic levels (when instances are launched in response to growing demand, it is important they are ready for service as soon as possible).

Once an instance has been configured and an image created from it, that configuration is baked into the AMI. As we look at configuration management tools in [Chapter 4](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch04.html#aws_config_management), we will see how tools like Puppet can be used to dynamically configure an instance. This raises the question of how much of the configuration should be baked into the AMI, and how much should be dynamically configured.

At one end of the spectrum, you can deploy an entirely vanilla Ubuntu image, automatically install a configuration management tool such as Puppet, and then apply your desired configuration to start up the correct services (such as Nginx for a web server). At the other end of the spectrum, you could create a custom AMI for each specific role within the application: one for the database server, one for the web server, and so on. In the latter case, all configuration options are baked into the AMI, and no dynamic configuration is performed when the instance is launched.

In our experience, the best option is somewhere in the middle: some roles have their own AMI, whereas other AMIs perform multiple roles. The most efficient place will depend on various factors, including the type of software you deploy and how frequently you modify the server configuration. If it is important for newly launched instances to start serving requests as quickly as possible (which includes practically all uses of Auto Scaling), you’ll want to reduce the amount of automatic configuration that takes place on boot.

At its core, an AMI is essentially a disk image and a metadata file describing how that disk image can be used to launch a virtual server. The metadata file keeps track of some internal information that is required when launching instances from this AMI, such as which Linux kernel to use.

In the early days of EC2, the only available AMI type was what is now known as an instance store−backed AMI. As the Elastic Block Store service was introduced and evolved, an additional type of AMI was created: the EBS-backed AMI. The key architectural difference between the two is in where the disk image that contains the root volume is stored.

For EBS-backed AMIs, this is simply an EBS snapshot. When launching a new instance from such an image, a volume is created using this snapshot, and this new volume is used as the root device on the instance.

Instance store−backed AMIs are created from template disk images stored in S3, which means the disk image must be copied from S3 each time an instance is launched, introducing a startup delay over EBS-backed instances. Because the image must be downloaded from S3 each time, the root volume size is also limited to 10 GB, whereas EBS-backed instances have their root volumes limited to a more generous 16 TB.

In practice, an EBS-backed AMI is nearly always the best option. This type of AMI can be temporarily stopped and restarted without losing any data, whereas instance store−backed AMIs can only be terminated, at which point all data stored on the volume is lost.

##### UPGRADING A RUNNING INSTANCE’S HARDWARE

AWS’s capability to change the hardware underlying your instance with just a few API calls is a great perk: you can upgrade your systems (or scale back your expense) with unprecendented ease. This complements AWS’s ability to scale out by adding more instances with the ability to scale up by moving existing instances to more powerful virtual hardware.

To take advantage of this capability your instances must be EBS-backed, as they will need to be restarted in order to change instance type.You will want to use an elastic IP address to be able to maintain the same network endpoint for your existing service. Lastly you should standardize on 64-bit AMIs across your EC2 deployment, as changing bit width requires replacing the AMI itself. For this very same reason the instance’s existing virtualization technology and root volume support choices cannot be altered.

Let’s play it all out in practice:

$ **aws ec2 run-instances --image-id ami-c80b0aa2 --region us-east-1 \**

**--instance-type t2.micro --output text**

740376006796 r-40546f92

INSTANCES 0 x86\_64 False xen ami-c80b0aa2 i-995fa01e t2.micro 2016-04-10T14:52:12.000Z ip-172-31-5-195.ec2.internal 172.31.5.195 /dev/sda1 ebs True subnet-d14ae8a7 hvm vpc-934935f7

[ output truncated ]

$ **aws ec2 describe-instances --instance-ids i-995fa01e | grep Type**

"InstanceType": "t2.micro",

"RootDeviceType": "ebs",

"VirtualizationType": "hvm",

$ **aws ec2 stop-instances --instance-ids i-995fa01e --output text**

STOPPINGINSTANCES i-995fa01e

CURRENTSTATE 64 stopping

PREVIOUSSTATE 16 running

$ **aws ec2 modify-instance-attribute --instance-type m4.xlarge --instance-id i-995fa01e**

$ **aws ec2 start-instances --instance-ids i-995fa01e --output text**

STARTINGINSTANCES i-995fa01e

CURRENTSTATE 0 pending

PREVIOUSSTATE 80 stopped

$ **aws ec2 describe-instances --instance-ids i-995fa01e | grep Type**

"InstanceType": "m4.xlarge",

"RootDeviceType": "ebs",

"VirtualizationType": "hvm",

After you re-associate the elastic IP address, your server will have just received an on-the-fly upgrade from a flimsy single core with only one gigabyte of RAM to a considerably beefier 16 GBs powered by four cores-not bad for a few seconds’ work.

## Building Your Own AMI

AMI builds should be automated as soon as possible, if you do it with any kind of regularity. It is tedious work and involves a lot of waiting around. Automating the process means you’ll probably update AMIs more frequently, reducing a barrier to pushing out new features and software upgrades. Imagine you learn of a critical security flaw in your web server software that must be updated immediately. Having a procedure in place to create new AMIs and push them into production will help you respond to such scenarios rapidly and without wasting lots of time.

To demonstrate the procedures for creating an AMI and some of the useful features that AMIs provide, let’s create an AMI using the command-line tools. This AMI will run an Nginx web server that displays a simple welcome page. We will look at a method of automating this procedure later in the book, in [“Building AMIs with Packer”](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch04.html#building_amis_with_packer).

Begin by selecting an AMI to use as a base. We will be using our Trusty Ubuntu 14.04 image with the ID ami-c80b0aa2. Launch an instance of this AMI with aws ec2 run-instances, remembering to specify a valid key pair name and security group granting access to SSH, then use aws ec2 describe-instances to find out the public DNS name for the instance:

$ # if you have not created a security group for SSH access yet,

$ # you need to do that first:

$ **aws ec2 create-security-group --group-name ssh --description "SSH Access"**

{

"GroupId": "sg-4ebd8b36"

}

$ **aws ec2 authorize-security-group-ingress --group-name ssh --protocol tcp \**

**--port 22 --cidr 0.0.0.0/0**

$ **aws ec2 describe-security-groups --group-names ssh --output text**

SECURITYGROUPS SSH Access sg-4ebd8b36 ssh 740376006796 vpc-934935f7

IPPERMISSIONS 22 tcp 22

IPRANGES 0.0.0.0/0

IPPERMISSIONSEGRESS -1

IPRANGES 0.0.0.0/0

$ **aws ec2 run-instances *--image-id ami-c80b0aa2* --region us-east-1 \**

**--key *your-key-pair-name* --security-groups ssh --instance-type t2.micro**

740376006796 r-315a9492

INSTANCES 0 x86\_64 False xen ami-c80b0aa2 i-d9c83544 t2.micro federico 2016-04-10T22:40:26.000Z ip-172-31-55-4.ec2.internal 172.31.55.4 /dev/sda1 ebs True subnet-2a45b400 hvm vpc-934935f7

[ output truncated ]

$ **aws ec2 describe-instances --instance-ids *i-d9c83544* --region us-east-1 \**

**--output text**

RESERVATIONS 740376006796 r-315a9492

INSTANCES 0 x86\_64 False xen ami-c80b0aa2 i-d9c83544 t2.micro federico 2016-04-10T22:40:26.000Z ip-172-31-55-4.ec2.internal 172.31.55.4 ec2-54-84-237-158.compute-1.amazonaws.com 54.84.237.158 /dev/sda1 ebs True subnet-2a45b400 hvm vpc-934935f7

[ output truncated ]

Once the instance has launched, we need to log in via SSH to install Nginx. If you are not using Ubuntu, the installation instructions will differ slightly. On Ubuntu, update the package repositories and install Nginx as follows:

$ **ssh ubuntu@*ec2-54-84-237-158.compute-1.amazonaws.com***

The authenticity of host 'ec2-54-84-237-158.compute-1.amazonaws.com (54.84.237.158)' can't be established.

ECDSA key fingerprint is a0:d1:5a:ef:02:32:bd:72:28:41:fd:f1:b1:c6:75:4e.

Are you sure you want to continue connecting (yes/no)? **yes**

$ **sudo apt update**

[ output truncated ]

$ **sudo apt install nginx-full --assume-yes**

[ output truncated ]

By default, Nginx is installed with a welcome page stored at /usr/share/nginx/www/index.html. If you like, you can modify this file to contain some custom content.

Once the instance is configured, we need to create a matching AMI using aws ec2 create-image. This command will automatically create an AMI from a running instance. Doing so requires that the instance be stopped and restarted, so your SSH session will be terminated when you run this command. In the background, a snapshot of the EBS volumes used for your instance will be made. This snapshot will be used when launching new instances through a newfangled AMI ID. Because it can take some time before snapshots are ready to use, your new AMI will remain in the pending state for a while after aws ec2 create-image completes. The image cannot be used until it enters the availablestate. You can check on the status in the Management Console or with the aws ec2 describe-images command:

$ **aws ec2 create-image --instance-id *i-d9c83544* --region us-east-1 \**

**--name test-image --output text**

ami-4dc5d527

$ **aws ec2 describe-images --region us-east-1 --image-ids *ami-4dc5d527*\**

**--output text**

IMAGES x86\_64 2016-04-10T22:51:06.000Z xen ami-4dc5d527 740376006796/test-image machine test-image 740376006796 False /dev/sda1 ebs simple pending hvm

BLOCKDEVICEMAPPINGS /dev/sda1

EBS True False snap-f407b282 8 standard

BLOCKDEVICEMAPPINGS /dev/sdb ephemeral0

BLOCKDEVICEMAPPINGS /dev/sdc ephemeral1

When your new image is ready, it can be launched by any of the means described previously. Launch a new instance based on this image and get the public DNS name with aws ec2 describe-instances. Connect via SSH, then confirm that Nginx has started automatically:

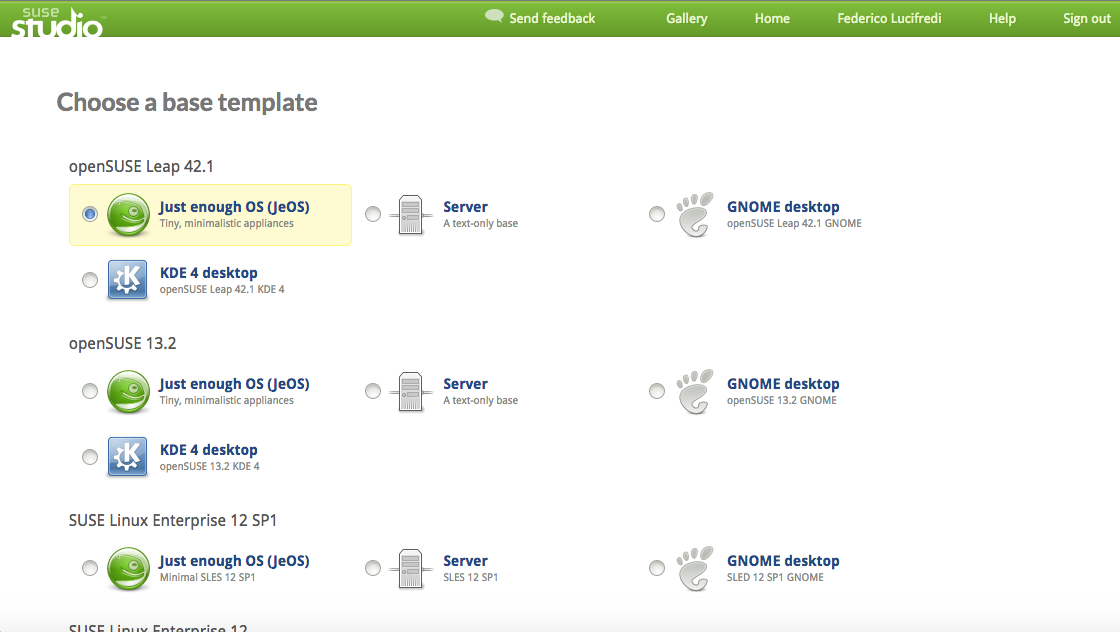
$ **service nginx status**

\* nginx is running

Although we have configured Nginx and have a running web server, you can’t access the Nginx welcome page just yet. If you try to visit the instance’s public DNS name in your web browser, the request will eventually time out. This is because EC2 instances are, by default, protected by a firewall that allows only connections from instances in the same security group-incoming HTTP connections have to be explicitly enabled with the same processes we used to allow inbound SSH connections. These firewalls, known as security groups, are discussed in the next chapter.

###### TIP

We have focused on automation where AMIs are concerned, but an interactive approach to build AMIs reminescent of the Management Console also exists. [SUSE Studio](https://susestudio.com/) provides a web-based interface to building various types of images based on the eponymous Linux distribution. SUSE Studio possibly offers the best interactive path to AMI building, the trade-off being the use of a distribution decidedly off the mainstream of Public Cloud adoption.



###### *Figure 2-8. SUSE Studio provides a friendly web interface to AMI creation*

Remember that both this instance and the original instance from which we created the image are still running. You might want to terminate those before moving on to the next section. The two-line script in [“Parsing JSON Output with jq”](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch01.html#parsing_JSON_output_with_jq) can be used to terminate all running EC2 instances in your account to clean the slate after running a few experiments.

Tagging your images is a good way to keep track of them. This can be done with aws ec2 create-tags command. By using backticks to capture the output of shell commands, you can quickly add useful information, such as who created the AMI, as well as static information like the role:

$ **aws ec2 create-tags --resources *ami-4dc5d527* --tags Key=role,Value=webserver\**

**Key=created-by,Value=`whoami` Key=stage,Value=production**

$ **aws ec2 describe-tags --output text**

TAGS created-by ami-4dc5d527 image federico

TAGS role ami-4dc5d527 image webserver

TAGS stage ami-4dc5d527 image production

##### TAGGING STRATEGY

Your AMI tagging strategy should let you keep track of the purpose of the image, when it was created, and its current state. Consider the life cycle of an image: first it will be created and tested, then used in production for a while, and then finally retired when a new instance is created to replace it. The state tag can be used to keep track of this process, with values such as dev, production, or retired. A companion state-changed tag can track when changes were made.

Automate the process of moving images through the life cycle so that you never forget to add or remove the relevant tags.

## Deregistering AMIs

Once an AMI is no longer required, it should be deregistered, which means it will no longer be available to use for launching new instances. Although they are not particularly expensive, it is important to regularly remove old AMIs because they clutter up the interface and contribute to a gradual increase of your AWS costs.

###### WARNING

A good way to identify snapshots in your account ripe for deletion is to retrieve the complete listing of snapshots associated with your OwnerID and applying additional filtering. The OwnerID for your account can be found in the Account Identifiers section of the [Security Credentials](https://console.aws.amazon.com/iam/home?#security_credential) page, but the handy alias self is always available. To list all your snapshots, enter:

aws ec2 describe-snapshots --owner-ids self --output text

You must also delete the snapshot used to create the root volume. This will not happen automatically.

AWS allows you to delete the snapshot before deregistering the AMI. Doing so means you will have an AMI that looks as though it is available and ready for use, but will, in fact, fail when you try to launch an instance. If the deregistered AMI is referenced in Auto Scaling groups, it might be some time before you notice the problem. The only option in that case is to quickly create a new AMI and update the Auto Scaling group.

You can check to see whether a particular AMI is in use by running instances with the aws ec2 describe-instances command, for example:

$ **aws ec2 describe-instances --filters Name=image-id,Values=ami-4dc5d527 \**

**--output text**

RESERVATIONS 740376006796 r-726ca2d1

INSTANCES 0 x86\_64 False xen ami-4dc5d527 i-8d30cd10 t2.micro federico 2016-04-10T23:05:25.000Z ip-172-31-55-118.ec2.internal 172.31.55.118 ec2-52-91-46-86.compute-1.amazonaws.com 52.91.46.86 /dev/sda1 ebs True subnet-2a45b400 hvm vpc-934935f7

[ output truncated ]

TAGS aws:cloudformation:logical-id Ec2Instance

TAGS aws:cloudformation:stack-id arn:aws:cloudformation:us-east-1:740376006796:stack/example-stack/ffad2160-069c-11e6-b07a-50d5caf92cd2

TAGS aws:cloudformation:stack-name example-stack

This works for individual instance. For instances that were launched as part of an Auto Scaling group, we can use the aws autoscaling describe-launch-configurations command. Unfortunately, this command does not accept a filter argument, so it cannot be used in quite the same way. As a workaround, you can grep the output of aws autoscaling describe-launch-configs for the AMI ID.

Performing these checks before deleting AMIs en masse can save you from a rather irritating cleanup exercise.

Once you are sure the AMI is safe to deregister, you can do so with aws ec2 deregister-image:

$ **aws ec2 deregister-image --image-id ami-4dc5d527 --region us-east-1**

Remember to delete the snapshot that was used as the root volume of the AMI. You can find it through the aws ec2 describe-snapshots command. When AWS creates a new snapshot, it uses the description field to store the ID of the AMI it was created for, as well as the instance and volume IDs referencing the resources it was created from. Therefore, we can use the AMI ID as a filter in our search, returning the ID of the snapshot we want to delete:

$ **aws ec2 describe-snapshots --region us-east-1 \**

**--filters Name=description,Values="Created by CreateImage\*for ami-4dc5d527\*" \**

**--output text**

SNAPSHOTS Created by CreateImage(i-d9c83544) for ami-4dc5d527 from vol-7df10dac False 740376006796 100% snap-f407b282 2016-04-10T22:51:18.000Z completed vol-7df10dac 8

$ **aws ec2 delete-snapshot --region us-east-1 --snapshot-id snap-f407b282**

The same can be achieved with a simple Boto script, shown in [Example 2-2](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch02.html#boto_delete_retired_images). This script will delete all images that have a stage tag with a value of retired.

##### ***Example 2-2. Deleting images with a Python script***

from boto.ec2 import connect\_to\_region

ec2\_conn = connect\_to\_region('us-east-1')

for image in ec2\_conn.get\_all\_images(filters={'tag:stage': 'retired', 'tag-key': 'stage-changed'}):

for id, device in image.block\_device\_mapping:

print 'Deleting snapshot %s for image %s' % (device.snapshot\_id, image.id)

ec2\_conn.delete\_snapshot(device.snapshot\_id)

print 'Deleting image %s' % image.id

ec2\_conn.delete\_image(image.id)

This script relies on your AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY environment variables being set—Boto will attempt to read these automatically. It will delete all images (and their snapshots) that have been in the retired stage for more than a week. To use this script, make sure your instances follow the tagging strategy described in [“Tagging Strategy”](https://www.safaribooksonline.com/library/view/aws-system-administration/9781449342562/ch02.html#tagging_strategy). Save this file as delete-retired-amis.py and use chmod to make it executable.

The call to get\_all\_images specifies some filter conditions: we are interested in images that have a stage tag with a value of retired and a state-changed tag with any value. Because we cannot use date-based filters on the server side, we must retrieve all snapshots with a state-changed tag and then compare the date on the client side to see whether each image is old enough for deletion.

Because deleting an image does not automatically delete the snapshot it uses for its root volume, we must do this separately by calling delete\_snapshot.

# Pets vs. Cattle

Microsoft’s Bill Baker is credited with originating the metaphor popularized by OpenStack’s [Randy Bias](http://www.slideshare.net/randybias/pets-vs-cattle-the-elastic-cloud-story) that so vividly illustrates two radically opposite approaches to managing servers. In this tale, pets are lovingly cared for, taken to the vet when they get sick, and lovingly nursed back to health -- cattle on the other hand are replaced without a second thought, even slaughtered. This distinction is humorously used to illustrate the more formal distinction delineated by Gartner in IT operations: traditional [mode 1 IT](http://blogs.gartner.com/it-glossary/files/2015/01/bimodaltable.png) servers are highly-managed assets that scale up to bigger, more costly hardware and are carefully restored to health should anything go amiss. Mode 2 IT on the other hand espouses a radically different operational philosophy: servers are highly disposable entities that are instantiated through automation, eliminated at the drop of a hat if no longer needed, and “scale out” in herds. Replacing a server with an expensive many-socket system (or virtual machine) and more complex memory architecture is decidedly mode 1, while adding as many equally-sized web servers behind a load balancer as required by the service load is the mode 2 way.

Mode 1 IT is the mainstay of traditional datacenter operations, whereas Mode 2 has emerged as the prim and proper way to design and operate applications in a public cloud environment. AWS gives you plenty of choices in how you achieve your goals, and we have been introducing all the technical details you need in order to scale services either up or out, but as we proceed to design a realistic application in the coming chapters, we will decidedly lean the way a cloud architect would, and adopt a mode 2 mindset in our design exclusively. You should do the same, pets do not belong in your cloud architecture.