AWS Administration – The Definitive Guide

AWS is at the forefront of cloud computing today. Because of its versatility and flexible design, AWS can be used to accomplish a variety of simple and complicated tasks, such as hosting multilayer websites, running large scale parallel processing, content delivery, petabyte storage and archival, and lots more.

Whether you are a seasoned system admin or a rookie, AWS Administrative Guide will provide you with all the necessary skills to design, deploy, and manage your applications on the AWS cloud platform. The book guides you through the core AWS services, such as IAM, EC2, VPC, RDS, and S3, using a simple real-world application hosting example that you can relate to. Each chapter is designed to provide you with the most information possible about a particular AWS service, coupled with easy-to-follow hands-on steps, best practices, tips, and recommendations.

By the end of the book, you will be able to create a highly secure, fault tolerant, and scalable environment for your applications to run on.

Who this book is written for
This book is for those who want to learn and leverage AWS. Although no prior experience with AWS is required, it is recommended that you have some hands-on experience of Linux, web services, and basic networking.

What you will learn from this book
- Get a brief introduction to Cloud Computing and AWS accompanied by learning how to sign up for your first AWS account
- Create and manage users, groups, and permissions using AWS Identity and Access Management services
- Get started with deploying and accessing EC2 instances, and working with EBS volumes and snapshots
- Customize and create your very own Amazon Machine Image
- Design and deploy your instances on a highly secured, network isolated environment using Amazon VPC
- Effectively monitor your AWS environment using specialized alarms, custom monitoring metrics, and much more
- Explore the various benefits of Database-as-a-Service offerings and leverage them using Amazon RDS and Amazon DynamoDB
- Take an in-depth look at what’s new with AWS, including EC2 Container Service and Elastic File System
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 3 'Images and Instances'
- A synopsis of the book’s content
- More information on AWS Administration – The Definitive Guide
About the Author

Yohan Wadia is a client-focused virtualization and cloud expert with 6 years of experience in the IT industry.

He has been involved in conceptualizing, designing, and implementing large-scale solutions for a variety of enterprise customers based on VMware vCloud, Amazon Web Services, and Eucalyptus Private Cloud.

His community-focused involvement also enables him to share his passion for virtualization and cloud technologies with peers through social media engagements, public speaking at industry events, and through his personal blog—yoyoclouds.com

He is currently working with an IT services and consultancy company as a Cloud Solutions Lead and is involved in designing and building enterprise-level cloud solutions for internal as well as external customers. He is also a VMware Certified Professional and a vExpert (2012 and 2013).

I wish to dedicate this book to both my loving parents, Ma and Paa. Thank you for all your love, support, encouragement, and patience. I would also like to thank the entire Packt Publishing team, especially Ruchita Bhansali, Athira Laji, and Gaurav Sharma, for their excellent guidance and support.

And finally, a special thanks to one of my favorite bunch of people: the amazing team of developers, support staff, and engineers who work at AWS for such an "AWSome" cloud platform!

Not all those who wander are lost.
- J. R. R. Tolkien
Preface

Cloud computing has definitely matured and evolved a lot ever since its conception. Practically all major industries and top fortune 500 companies today run their application workloads on clouds to reap all sorts of benefits, ranging from reduced costs, better availability of their applications, and easier manageability to on-demand scalability, and much more! At the forefront of this cloud innovation is a market leader like no other: Amazon Web Services (AWS).

AWS provides a ton of easy-to-use products and services that you can leverage to build, host, deploy, and manage your applications on the cloud. It also provides a variety of ways to interact with these services, such as SDKs, APIs, CLIs, and even a web-based management console.

This book is a one stop shop where you can find all there is to getting started with the core AWS services, which include EC2, S3, RDS, VPCs, and a whole lot more! If you are a sysadmin or an architect or someone who just wants to learn and explore various aspects of administering AWS services, then this book is the right choice for you! Each chapter of this book is designed to help you understand the individual services' concepts as well as gain hands-on experience by practicing simple and easy to follow steps. The chapters also highlight some key best practices and recommendations that you ought to keep in mind when working with AWS.

What this book covers

Chapter 1, Introducing Amazon Web Services, covers the introductory concepts and general benefits of cloud computing along with an overview of Amazon Web Services and its overall platform. The chapter also walks you through your first AWS signup process, and finally ends with the configuration of the AWS CLI.
Chapter 2, *Security and Access Management*, discusses the overall importance of security and how you can achieve it using an AWS core service known as Identity and Access Management (IAM). The chapter walks you through the steps required to create and administer AWS users, groups, as well as how to create and assign permissions and policies to them.

Chapter 3, *Images and Instances*, provides hands-on knowledge about EC2 instances and images, and how you can create and manage them using both the AWS Management Console as well as the AWS CLI.

Chapter 4, *Security, Storage, Networking and Lots More!*, discusses some of the key aspects that you can leverage to provide added security for your applications and instances. The chapter also provides an in-depth overview of EC2 instance storage as well as networking options followed by some recommendations and best practices.

Chapter 5, *Building Your Own Private Clouds Using Amazon VPC*, introduces you to the concept and benefits provided by AWS Virtual Private Cloud (VPC) service. The chapter also provides an in-depth look at various VPC deployment strategies and how you can best leverage them for your own environments.

Chapter 6, *Monitoring Your AWS Infrastructure*, covers AWS's primary monitoring service, called as Amazon CloudWatch. In this chapter, you will learn how to effectively create and manage alerts, loggings, and notifications for your EC2 instances, as well as your AWS environment.

Chapter 7, *Manage Your Applications with Auto Scaling and Elastic Load Balancing*, discusses some of the key AWS services that you should leverage to create a dynamically scalable and highly available web application.

Chapter 8, *Database-as-a-Service Using Amazon RDS*, provides an in-depth look at how you can effectively design, create, manage, and monitor your RDS instances on AWS.

Chapter 9, *Working with Simple Storage Service*, provides practical knowledge and design considerations that you should keep in mind when working with Amazon's infinitely scalable and durable object storage known as Amazon S3.

Chapter 10, *Extended AWS Services for Your Application*, provides a brief overview of add-on AWS services that you can leverage for enhancing your applications' performance and availability.
In the previous chapter, we learnt a lot about how AWS provides top of the line security and access management capabilities to its users in the form of IAM and various other tools.

In this chapter, we will explore one of the most popular and widely used AWS’s core services, that is, Elastic Compute Cloud (EC2). This chapter will cover many important aspects about EC2, such as its use cases, its various terms and terminologies, and cost-effective pricing strategies to name a few. It will also show you how to get started with the service using both the AWS Management Console and the AWS CLI; so buckle up and get ready for an awesome time!

Introducing EC2!

Remember the never ending hassles of a long and tedious procurement process? All that time you spent waiting for a brand new server to show up at your doorstep so that you could get started on it? Something we all as sysadmins have gone through. Well, that all changed on August 25, 2006 when Amazon released the first beta version of one of its flagship service offerings called the Elastic Compute Cloud or EC2.

EC2 is a service that basically provides scalable compute capacity on an on-demand, pay-per-use basis to its end users. Let’s break it up a bit to understand the terms a bit better. To start with, EC2 is all about server virtualization! And with server virtualization, we get a virtually unlimited capacity of virtual machines or, as AWS calls it, instances. Users can dynamically spin up these instances, as and when required, perform their activity on them, and then shut down the same while getting billed only for the resources they consume.
EC2 is also a highly scalable service, which means that you can scale up from just a couple of virtual servers to thousands in a matter of minutes, and vice versa—all achieved using a few simple clicks of a mouse button! This scalability accompanied by dynamicity creates an elastic platform that can be used for performing virtually any task you can think of! Hence, the term Elastic Compute Cloud! Now that's awesome!

But the buck doesn't just stop there! With virtually unlimited compute capacity, you also get added functionality that helps you to configure your virtual server's network, storage, as well as security. You can also integrate your EC2 environment with other AWS services such as IAM, S3, SNS, RDS, and so on. To provide your applications with add-on services and tools such as security, scalable storage and databases, notification services, and so on and so forth.

**EC2 use cases**

Let's have a quick look at some interesting and commonly employed use cases for AWS EC2:

- **Hosting environments**: EC2 can be used for hosting a variety of applications and software, websites, and even games on the cloud. The dynamic and scalable environment allows the compute capacity to grow along with the application's needs, thus ensuring better quality of service to end users at all times. Companies such as Netflix, Reddit, Ubisoft, and many more leverage EC2 as their application hosting environments.

- **Dev/Test environments**: With the help of EC2, organizations can now create and deploy large scale development and testing environments with utmost ease. The best part of this is that they can easily turn on and off the service as per their requirements as there is no need for any heavy upfront investments for hardware.

- **Backup and disaster recovery**: EC2 can be also leveraged as a medium for performing disaster recovery by providing active or passive environments that can be turned up quickly in case of an emergency, thus resulting in faster failover with minimal downtime to applications.

- **Marketing and advertisements**: EC2 can be used to host marketing and advertising environments on the fly due to its low costs and rapid provisioning capabilities.

- **High Performance Computing (HPC)**: EC2 provides specialized virtualized servers that provide high performance networking and compute power that can be used to perform CPU-intensive tasks such as Big Data analytics and processing. NASA's JPL and Pfizer are some of the companies that employ the use of HPC using EC2 instances.
Introducing images and instances

To understand images and instances a bit better, we first need to travel a little back in time; don't worry, a couple of years back is quite enough! This was the time when there was a boom in the implementation and utilization of the virtualization technology!

Almost all IT companies today run their workloads off virtualized platforms such as VMware vSphere or Citrix XenServer to even Microsoft's Hyper-V. AWS, too, got into the act but decided to use and modify a more off the shelf, open sourced Xen as its virtualization engine. And like any other virtualization technology, this platform was also used to spin up virtual machines using either some type of configuration files or some predefined templates. In AWS's vocabulary, these virtual machines came to be known as instances and their master templates came to be known as images.

By now you must have realized that instances and images are nothing new! They are just fancy nomenclature that differentiates AWS from the rest of the plain old virtualization technologies, right? Well, no. Apart from just the naming convention, there are a lot more differences to AWS images and instances as compared to your everyday virtual machines and templates. AWS has put in a lot of time and effort from time to time in designing and structuring these images and instances, such that they remain lightweighted, spin up more quickly, and can even be ported easily from one place to another. These factors make a lot of difference when it comes to designing scalable and fault tolerant application environments in the cloud.

We shall be learning a lot about these concepts and terminologies in the coming sections of this, as well as in the next chapter, but for now, let's start off by understanding more about these images!

Understanding images

As discussed earlier, images are nothing more than preconfigured templates that you can use to launch one or more instances from. In AWS, we call these images Amazon Machine Images (AMIs). Each AMI contains an operating system which can range from any modern Linux distro to even Windows Servers, plus some optional software application, such as a web server, or some application server installed on it.
Images and Instances

It is important, however, to understand a couple of important things about AMIs. Just like any other template, AMIs are static in nature, which basically means that once they are created, their state remains unchanged. You can spin up or launch multiple instances using a single AMI and then perform any sort of modifications and alterations within the instance itself. There is also no restriction on the size of instances that you can launch based on your AMI. You can select anything from the smallest instance (also called as a micro instance) to the largest ones that are generally meant for high performance computing. Take a look at the following image of EC2 AMI:

Secondly, an AMI can contain certain launch permissions as well. These permissions dictate whether the AMI can be launched by anyone (public) or by someone or some account which I specify (explicit) or I can even keep the AMI all to myself and not allow anyone to launch instances from it but me (implicit). Why have launch permissions? Well, there are cases where some AMIs can contain some form of propriety software or licensed application, which you do not want to share freely among the general public.

In that case, these permissions come in really handy! You can alternatively even create something called as a paid AMI. This feature allows you to share your AMI to the general public, however, with some support costs associated with it.

AMIs can be bought and sold using something called as the AWS Marketplace as well—a one stop shop for all your AMI needs! Here, AMIs are categorized according to their contents and you as an end user can choose and launch instances off any one of them. Categories include software infrastructure, development tools, business and collaboration tools, and much more! These AMIs are mostly created by third parties or commercial companies who wish to either sell or provide their products on the AWS platform.
AMIs can be broadly classified into two main categories depending on the way they store their root volume or hard drive:

- **EBS-backed AMI**: An EBS-backed AMI simply stores its entire root device on an **Elastic Block Store (EBS)** volume. EBS functions like a network shared drive and provides some really cool add on functionalities like snapshotting capabilities, data persistence, and so on. Even more, EBS volumes are not tied to any particular hardware as well. This enables them to be moved anywhere within a particular availability zone, kind of like a **Network Attached Storage (NAS)** drive. We shall be learning more about EBS-backed AMIs and instances in the coming chapter.

- **Instance store-backed AMI**: An instance store-backed AMI, on the other hand, stores its images on the AWS S3 service. Unlike its counterpart, instance store AMIs are not portable and do not provide data persistence capabilities as the root device data is directly stored on the instance's hard drive itself. During deployment, the entire AMI has to be loaded from an S3 bucket into the instance store, thus making this type of deployment a slightly slow process.

The following image depicts the deployments of both the instance store-backed and EBS-backed AMIs. As you can see, the root and data volumes of the instance store-backed AMI are stored locally on the **HOST SERVER** itself, whereas the second instance uses EBS volumes to store its root device and data.
The following is a quick differentiator to help you understand some of the key differences between EB-backed and Instance store-backed AMIs:

<table>
<thead>
<tr>
<th></th>
<th>EBS backed</th>
<th>Instance store backed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root device</td>
<td>Present on an EBS volume.</td>
<td>Present on the instance itself.</td>
</tr>
<tr>
<td>Disk size limit</td>
<td>Up to 16 TB supported.</td>
<td>Up to 10 GB supported.</td>
</tr>
<tr>
<td>Data persistence</td>
<td>Data is persistent even after the instance is terminated.</td>
<td>Data only persists during the lifecycle of the instance.</td>
</tr>
<tr>
<td>Boot time</td>
<td>Less than a minute. Only the parts of the AMI that are required for the boot process are retrieved for the instance to be made ready.</td>
<td>Up to 5 minutes. The entire AMI has to be retrieved from S3 before the instance is made ready.</td>
</tr>
<tr>
<td>Costs</td>
<td>You are charged for the running instance plus the EBS volume's usage.</td>
<td>You are charged for the running instance plus the storage costs incurred by S3.</td>
</tr>
</tbody>
</table>

**Amazon Linux AMI**

Amazon Linux AMI is a specially created, lightweight Linux-based image that is supported and maintained by AWS itself. The image is based on a **RedHat Enterprise Linux (RHEL)** distro, which basically means that you can execute almost any and all RHEL-based commands, such as yum and system-config, on it.

The image also comes pre-packaged with a lot of essential AWS tools and libraries that allow for easy integration of the AMI with other AWS services. All in all, everything from the yum repos to the AMIs security and patching is taken care of by AWS itself!

The Amazon Linux AMI comes at no additional costs. You only have to pay for the running instances that are created from it. You can read more about the Amazon Linux AMI at [http://aws.amazon.com/amazon-linux-ami/](http://aws.amazon.com/amazon-linux-ami/).

Later on, we will be using this Amazon Linux AMI itself and launching our very first, but not the last, instance into the cloud, so stick around!
Understanding instances

So far we have only been talking about images; so now let's shift the attention over to instances! As discussed briefly earlier, instances are nothing but virtual machines or virtual servers that are spawned off from a single image or AMI. Each instance comes with its own set of resources, namely CPU, memory, storage, and network, which are differentiated by something called as instance families or instance types. When you first launch an instance, you need to specify its instance type. This will determine the amount of resources that your instance will obtain throughout its lifecycle.

AWS currently supports five instance types or families, which are briefly explained as follows:

• **General purpose**: This group of instances is your average, day-to-day, balanced instances. Why balanced? Well, because they provide a good mix of CPU, memory, and disk space that most applications can suffice with while not compromising on performance. The general purpose group comprises the commonly used instance types such as t2.micro, t2.small, t2.medium, and the m3 and m4 series which comprises m4.large, m4.xlarge, and so on and so forth. On average, this family contains instance types that range from 1 VCPU and 1 GB RAM (t2.micro) all the way to 40 VCPUs and 160 GB RAM (m4.10xlarge).

• **Compute optimized**: As the name suggests, these are specialized group of instances that are commonly used for CPU-intensive applications. The group comprises two main instances types, that is, C3 and C4. On an average, this family contains instances that can range from 2 VCPUs and 2.75 GB RAM (c4.large) to 36 VCPUs and 60 GB RAM (c4.8xlarge).

• **Memory optimized**: Similar to the compute optimized, this family comprises instances that require or consume more RAM than CPU. Ideally, databases and analytical applications fall into this category. This group consists of a single instance type called R3 instances, and they can range anywhere from 2 VCPUs and 15.25 GB RAM (r3.large) to 32 VCPUs and 244 GB RAM (r3.8xlarge).

• **Storage optimized**: This family of instances comprises specialized instances that provide fast storage access and writes using SSD drives. These instances are also used for high I/O performance and high disk throughput applications. The group also comprises two main instance types, namely the I2 and D2 (no, this doesn't have anything to do with R2D2!). These instances can provide SSD enabled storage ranging from 800 GB (i2.large) all the way up to 48 TB (d2.8xlarge)—now that's impressive!
• **GPU instances**: Similar to the compute optimized family, the GPU instances are specially designed for handling high CPU-intensive tasks but by using specialized NVIDIA GPU cards. This instance family is generally used for applications that require video encoding, machine learning or 3D rendering, and so on. This group consists of a single instance type called G2, and it can range between 1 GPU (g2.2xlarge) and 4 GPU (g2.8xlarge).

As of late, AWS EC2 supports close to 38 instance types, each with their own set of pros and cons and use cases. In such times, it actually becomes really difficult for an end user to decide which instance type is right for his/her application. The easiest and most common approach taken is to pick out the closet instance type that matches your application’s set of requirements - for example, it would be ideal to install a simple MongoDB database on a memory optimized instance rather than a compute or GPU optimized instance. Not that compute optimized instances are a wrong choice or anything, but it makes more sense to go for memory in such cases rather than just brute CPU. From my perspective, I have always fancied the general purpose set of instances simply because most of my application needs seem to get balanced out correctly with it, but feel free to try out other instance types as well.

**EC2 instance pricing options**

Apart from the various instance types, EC2 also provides three convenient instance pricing options to choose from, namely on-demand, reserved, and spot instances. You can use either or all of these pricing options at the same time to suit your application’s needs. Let’s have a quick look at all three options to get a better understanding of them.

**On-demand instances**

Pretty much the most commonly used instance deployment method, the on-demand instances are created only when you require them, hence the term on-demand. On-demand instances are priced by the hour with no upfront payments or commitments required. This, in essence, is the true pay-as-you-go payment method that we always end up mentioning when talking about clouds. These are standard computational resources that are ready whenever you request them and can be shut down anytime during its tenure.
By default, you can have a max of 20 such on-demand instances launched within a single AWS account at a time. If you wish to have more such instances, then you simply have to raise a support request with AWS using the AWS Management Console's Support tab. A good use case for such instances can be an application running unpredictable workloads, such as a gaming website or social website. In this case, you can leverage the flexibility of on-demand instances accompanied with their low costs to only pay for the compute capacity you need and use and not a dime more!

On-demand instance costs vary based on whether the underlying OS is a Linux or Windows, as well as in the regions that they are deployed in.

Consider this simple example: A t2.micro instance costs $0.013 per hour to run in the US East (N. Virginia) region. So, if I was to run this instance for an entire day, I would only have to pay $0.312! Now that's cloud power!

**Reserved instances**

Deploying instances using the on-demand model has but one slight drawback, which is that AWS does not guarantee the deployment of your instance. Why, you ask? Well to put it simply, using on-demand model, you can create and terminate instances on the go without having to make any commitments whatsoever. It is up to AWS to match this dynamic requirement and make sure that adequate capacity is present in its datacenters at all times. However, in very few and rare cases, this does not happen, and that's when AWS will fail to power on your on-demand instance.

In such cases, you are better off by using something called as reserved instances, where AWS actually guarantees your instances with resource capacity reservations and significantly lower costs as compared to the on-demand model. You can choose between three payment options when you purchase reserved instances: all upfront, partial upfront, and no upfront. As the name suggests, you can choose to pay some upfront costs or the full payment itself for reserving your instances for a minimum period of a year and maximum up to three years.

Consider our earlier example of the t2.micro instance costing $0.0013 per hour. The following table summarizes the upfront costs you will need to pay for a period of one year for a single t2.micro instance using the reserved instance pricing model:

<table>
<thead>
<tr>
<th>Payment method</th>
<th>Upfront cost</th>
<th>Monthly cost</th>
<th>Hourly cost</th>
<th>Savings over on-demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>No upfront</td>
<td>$0</td>
<td>$6.57</td>
<td>$0.009</td>
<td>31%</td>
</tr>
<tr>
<td>Partial upfront</td>
<td>$51</td>
<td>$2.19</td>
<td>$0.0088</td>
<td>32%</td>
</tr>
<tr>
<td>All upfront</td>
<td>$75</td>
<td>$0</td>
<td>$0.0086</td>
<td>34%</td>
</tr>
</tbody>
</table>
Reserved instances are the best option when the application loads are steady and consistent. In such cases, where you don't have to worry about unpredictable workloads and spikes, you can reserve a bunch of instances in EC2 and end up saving on additional costs.

**Spot instances**

Spot instances allow you to bid for unused EC2 compute capacity. These instances were specially created to address a simple problem of excess EC2 capacity in AWS. How does it all work? Well, it's just like any other bidding system. AWS sets the hourly price for a particular spot instance that can change as the demand for the spot instances either grows or shrinks. You as an end user have to place a bid on these spot instances, and when your bid exceeds that of the current spot price, your instances are then made to run! It is important to also note that these instances will stop the moment someone else out bids you, so host your application accordingly. Ideally, applications that are non-critical in nature and do not require large processing times, such as image resizing operations, are ideally run on spot instances.

Let's look at our trusty t2.micro instance example here as well. The on-demand cost for a t2.micro instance is $0.013 per hour; however, I place a bid of $0.0003 per hour to run my application. So, if the current bid cost for the t2.micro instance falls below my bid, then EC2 will spin up the requested t2.micro instances for me until either I choose to terminate them or someone else out bids me on the same—simple, isn't it?

Spot instances compliment the reserved and on-demand instances; hence, ideally, you should use a mixture of spot instances working on-demand or reserved instances just to be sure that your application has some compute capacity on standby in case it needs it.

**Working with instances**

Okay, so we have seen the basics of images and instances along with various instance types and some interesting instance pricing strategies as well. Now comes the fun part! Actually deploying your very own instance on the cloud!

In this section, we will be using the AWS Management Console and launching our very first t2.micro instance on the AWS cloud. Along the way, we shall also look at some instance lifecycle operations such as start, stop, reboot, and terminate along with steps, using which you can configure your instances as well. So, what are we waiting for? Let's get busy!
To begin with, I have already logged in to my AWS Management Console using the IAM credentials that we created in our previous chapter. If you are still using your root credentials to access your AWS account, then you might want to revisit Chapter 2, Security and Access Management, and get that sorted out! Remember, using root credentials to access your account is a strict no no!

Although you can use any web browser to access your AWS Management Console, I would highly recommend using Firefox as your choice of browser for this section.

Once you have logged into the AWS Management Console, finding the EC2 option isn't that hard. Select the EC2 option from under the Compute category, as shown in the following screenshot:

This will bring up the EC2 dashboard on your browser. Feel free to have a look around the dashboard and familiarize yourself with it. To the left, you have the Navigation pane that will help you navigate to various sections and services provided by EC2, such as Instances, Images, Network and Security, Load Balancers, and even Auto Scaling. The centre dashboard provides a real-time view of your EC2 resources, which includes important details such as how many instances are currently running in your environment, how many volumes, key pairs, snapshots, or elastic IPs have been created, so on and so forth.

The dashboard also displays the current health of the overall region as well as its subsequent availability zones. In our case, we are operating from the US West (Oregon) region that contains additional AZs called as us-west-2a, us-west-2b, and us-west-2c. These names and values will vary based on your preferred region of operation.
Next up, we launch our very first instance from this same dashboard by selecting the [Launch Instance] option, as shown in the following screenshot:

On selecting the **Launch Instance** option, you will be directed to a wizard driven page that will help you create and customize your very first instance. This wizard divides the entire instance creation operation into seven individual stages, each stage having its own set of configurable items. Let's go through these stages one at a time.

### Stage 1 – choose AMI

Naturally, our first instance has to spawn from an AMI, so that's the first step! Here, AWS provides us with a whole lot of options to choose from, which includes a [Quick Start] guide, which lists out the most frequently used and popular AMIs, and includes the famous Amazon Linux AMI as well, as shown in the following screenshot:

There are also a host of other operating systems provided here as well which includes Ubuntu, SUSE Linux, Red Hat, and Windows Servers.
Each of these AMIs has a uniquely referenced AMI ID, which looks something like this: `ami-e75272d7`. We can use this AMI ID to spin up instances using the AWS CLI, something which we will perform in the coming sections of this chapter. They also contain additional information such as whether the root device of the AMI is based on an EBS volume or not, whether the particular AMI is eligible under the Free tier or not, and so on and so forth.

Besides the Quick Start guide, you can also spin up your instances using the AWS Marketplace and the Community AMIs section as well. Both these options contain an exhaustive list of customized AMIs that have been created by either third-party companies or by developers and can be used for a variety of purposes. But for this exercise, we are going to go ahead and select Amazon Linux AMI itself from the Quick Start menu.

### Stage 2 – choose an instance type

With the AMI selected, the next step is to select the particular instance type or size as per your requirements. You can use the Filter by option to group and view instances according to their families and generations as well. In this case, we are going ahead with the general purpose `t2.micro` instance type, which is covered under the free tier eligibility and will provide us with 1 VCPU and 1 GB of RAM to work with! The following screenshot shows the configurations of the instance:

![Instance Types Configuration](image)

Ideally, now you can launch your instance right away, but this will not allow you to perform any additional configurations on your instance, which just isn't nice! So, go ahead and click on the Next: Configure instance Details button to move on to the third stage.
Stage 3 – configure instance details

Now here it gets a little tricky for first timers. This page will basically allow you to configure a few important aspects about your instance, including its network settings, monitoring, and lots more. Let's have a look at each of these options in detail:

- **Number of instances**: You can specify how many instances the wizard should launch using this field. By default, the value is always set to one single instance.

- **Purchasing option**: Remember the spot instances we talked about earlier? Well here you can basically request for spot instance pricing. For now, let's leave this option all together:

<table>
<thead>
<tr>
<th>Number of instances</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purchasing option</td>
<td>Request Spot Instances</td>
</tr>
<tr>
<td>Network</td>
<td>vpc-939c835 (172.31.0.0/16) (default)</td>
</tr>
<tr>
<td>Subnet</td>
<td>Default in us-w</td>
</tr>
<tr>
<td>Auto-assign Public IP</td>
<td>Use subnet setting (Enable)</td>
</tr>
<tr>
<td>IAM role</td>
<td>None</td>
</tr>
<tr>
<td>Shutdown behavior</td>
<td>Stop</td>
</tr>
<tr>
<td>Enable termination protection</td>
<td>Protect against accidental termination</td>
</tr>
</tbody>
</table>

- **Network**: Select the default Virtual Private Cloud (VPC) network that is displayed in the dropdown list. You can even go ahead and create a new VPC network for your instance, but we will leave all that for later chapters where we will actually set up a VPC environment.

In our case, the VPC has a default network of 172.31.0.0/16, which means we can assign up to 65,536 IP addresses using it.

- **Subnet**: Next up, select the Subnet in which you wish to deploy your new instance. You can either choose to have AWS select and deploy your instance in a particular subnet from an available list or you can select a particular choice of subnet on your own. By default, each subnet's Netmask defaults to /20, which means you can have up to 4,096 IP addresses assigned in it.
• **Auto-assign Public IP**: Each instance that you launch will be assigned a Public IP. The Public IP allows your instance to communicate with the outside world, a.k.a. the Internet! For now, select the use **Subnet setting (Enable)** option as shown.

• **IAM role**: You can additionally select a particular IAM role to be associated with your instance. In this case, we do not have any roles particularly created.

• **Shutdown behaviour**: This option allows you to select whether the instance should stop or be terminated when issued a shutdown command. In this case, we have opted for the instance to stop when it is issued a shutdown command.

• **Enable termination protection**: Select this option in case you wish to protect your instance against accidental deletions.

• **Monitoring**: By default, AWS will monitor few basic parameters about your instance for free, but if you wish to have an in-depth insight into your instance's performance, then select the **Enable CloudWatch detailed monitoring** option.

• **Tenancy**: AWS also offers you to power on your instances on a single-tenant, dedicated hardware in case your application’s compliance requirements are too strict. For such cases, select the **Dedicated** option from the **Tenancy** dropdown list, else leave it to the default **Shared** option. Do note, however, that there is a slight increase in the overall cost of an instance if it is made to run on a dedicated hardware.

Once you have selected your values, move on to the fourth stage of the instance deployment process by selecting the **Next: Add Storage** option.

### Stage 4 – add storage

Using this page, you can add additional EBS volumes to your instances. To add new volumes, simply click on the **Add New Volume** button. This will provide you with options to provide the size of the new volume along with its mount points. In our case, there is an 8 GB volume already attached to our instance. This is the t2.micro instance's root volume, as shown in the following screenshot:
Try and keep the volume's size under 30 GB to avail the free tier eligibility.

You can optionally increase the size of the volume and enable add-on features such as **Delete on Termination** as per your requirement. Once done, proceed to the next stage of the instance deployment process by selecting the **Next: Tag instance** option.

**Stage 5 – tag instances**

The tag instances page will allow you to specify tags for your EC2 instance. Tags are nothing more than normal key-value pairs of text that allow you to manage your AWS resources a lot easily. You can start, stop, and terminate a group of instances or any other AWS resources using tags. Each AWS resource can have a maximum of 10 tags assigned to it. For example, in our case, we have provided a tag for our instance as **ServerType:WebServer**. Here, **ServerType** is the key and **WebServer** its corresponding value. You can have other group of instances in your environment tagged as **ServerType:DatabaseServer** or **ServerType:AppServer** based on their application. The important thing to keep in mind here is that AWS will not assign a tag to any of your resources automatically. These are optional attributes that you assign to your resources in order to facilitate in easier management:

![Tag Instance Table]

Once your tags are set, click on the **Next: Configure Security Group** option to proceed.
Stage 6 – configure security groups

Security groups are an essential tool used to safeguard access to your instances from the outside world. Security groups are nothing but a set of firewall rules that allow specific traffic to reach your instance. By default, the security groups allow for all outbound traffic to pass while blocking all inbound traffic. By default, AWS will auto-create a security group for you when you first start using the EC2 service. This security group is called as *default* and contains only a single rule that allows all inbound traffic on port 22.

In the **Configure Security Groups** page, you can either choose to Create a new security group or Select an existing security group. Let's go ahead and create one for starters. Select the Create a new security group option and fill out a suitable Security group name and Description. By default, AWS would have already enabled inbound SSH access by enabling port 22:

![Security Group Configuration](image)

You can add additional rules to your security group based on your requirements as well. For example, in our instance's case, we want the users to receive all inbound HTTP traffic as well. So, select the Add Rule option to add a firewall rule. This will populate an additional rule line, as shown in the preceding screenshot. Next, from the Type dropdown, select HTTP and leave the rest of the fields to their default values. With our security group created and populated, we can now go ahead with the final step in the instance launch stage.
Stage 7 – review instance launch

Yup! Finally, we are here! The last step toward launching your very first instance! Here, you will be provided with a complete summary of your instance's configuration details, including the AMI details, instance type selected, instance details, and so on. If all the details are correct, then simply go ahead and click on the Launch option. Since this is your first instance launch, you will be provided with an additional popup page that will basically help you create a key pair.

A key pair is basically a combination of a public and a private key, which is used to encrypt and decrypt your instance's login info. AWS generates the key pair for you which you need to download and save locally to your workstation. Remember that once a particular key pair is created and associated with an instance, you will need to use that key pair itself to access the instance. You will not be able to download this key pair again; hence, save it in a secure location. Take a look at the following screenshot to get an idea of selecting the key pair:

In EC2, the Linux instances have no login passwords by default; hence, we use key pairs to log in using SSH. In case of a Windows instance, we use a key pair to obtain the administrator password and then log in using an RDP connection.

Select the Create a new key pair option from the dropdown list and provide a suitable name for your key pair as well. Click on the Download Key Pair option to download the .PEM file. Once completed, select the Launch Instance option. The instance will take a couple of minutes to get started. Meanwhile, make a note of the new instance's ID (in this case, i-53fc559a) and feel free to view the instance's launch logs as well:
Phew! With this step completed, your instance is now ready for use! Your instance will show up in the EC2 dashboard, as shown in the following screenshot:

The dashboard contains and provides a lot of information about your instance. You can view your instance's ID, instance type, power state, and a whole lot more info from the dashboard. You can also obtain your instance's health information using the **Status Checks** tab and the **Monitoring** tab. Additionally, you can perform power operations on your instance such as start, stop, reboot, and terminate using the **Actions** tab located in the preceding instance table.

Before we proceed to the next section, make a note of your instance's **Public DNS** and the **Public IP**. We will be using these values to connect to the instances from our local workstations.

**Connecting to your instance**

Once your instance has launched successfully, you can connect to it using three different methods that are briefly explained as follows:

- **Using your web browser**: AWS provides a convenient Java-based web browser plugin called as MindTerm, which you can use to connect to your instances. Follow the next steps to do so:
  1. From the EC2 dashboard, select the instance which you want to connect to and then click on the **Connect** option.
  2. In the **Connect To Your Instance** dialog box, select the option **A Java SSH Client directly from my browser (Java required)** option. AWS will autofill the **Public IP** field with your instance's public IP address.
3. You will be required, however, to enter the **User name** and the **Private key path**, as shown in the following screenshot:

![Connect To Your Instance](image)

4. The **User Name** for an Amazon Linux AMIs is **ec2-user** by default. You can optionally choose to store the location of your private key in the browser's cache; however, it is not at all required. Once all the required fields are filled in, select the **Launch SSH Client** option.

   For most RHEL-based AMIs, the user name is either root or the ec2-user, and for Ubuntu-based AMIs, the user name is generally Ubuntu itself.

5. Since this is going to be your first SSH attempt using the MindTerm plugin, you will be prompted to accept an end user license agreement.

6. Select the **Accept** option to continue with the process. You will be prompted to accept few additional prompts along the way, which include the setting up of your home directory and known hosts directory on your local PC.
7. Confirm all these settings and you should now see the MindTerm console displaying your instance's terminal, as shown in the following screenshot:

• Using Putty: The second option is by far the most commonly used and one of my favorites as well! Putty, or PuTTY, is basically an SSH and telnet client that can be used to connect to your remote Linux instances. But before you get working on Putty, you will need a tool called PuttyGen to help you create your private key (*.ppk).

You can download Putty, PuttyGen, and various other SSH and FTP tools from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html.

After creating your private key, follow the next steps to use Putty and PuttyGen:

1. First up, download and install the latest copy of Putty and PuttyGen on your local desktops.
2. Next, launch PuttyGen from the start menu. You should see the PuttyGen dialog as shown in the following screenshot.
3. Click on the **Load** option to load your PEM file. Remember, this is the same file that we downloaded during stage 7 of the instance launch phase.

4. Once loaded, go ahead and save this key by selecting the **Save private key** option. PuttyGen will probably prompt you with a warning message stating that you are saving this key without a passphrase and would you like to continue.

5. Select **Yes** to continue with the process. Provide a meaningful **name** and save the new file (*.PPK) at a secure and accessible location. You can now use this PPK file to connect to your instance using Putty.
Now comes the fun part! Launch a Putty session from the Start menu. You should see the Putty dialog box as shown in the following screenshot. Here, provide your instance's Public DNS or Public IP in the Host Name (or IP address) field as shown. Also make sure that the Port value is set to 22 and the Connection type is selected as SSH.

6. Next, using Putty's Navigation | Category pane, expand the SSH option and then select Auth, as shown in the following screenshot. All you need to do here is browse and upload the recently saved PPK file in the Private key file for authentication field. Once uploaded, click on Open to establish a connection to your instance.
7. You will be prompted by a security warning since this is the first time you are trying to connect your instance. The security dialog box simply asks whether you trust the instance that you are connecting to or not. Click on the Yes tab when prompted.

8. In the Putty terminal window, provide the user name for your Amazon Linux instance (ec2-user) and hit the Enter key. Voila! Your first instance is now ready for use, as shown in the following screenshot. Isn't that awesome!

![Putty terminal window](image)

• **Using SSH**: The third and final method is probably the most simple and straightforward. You can connect to your EC2 instances using a simple SSH client as well. This SSH client can be installed on a standalone Linux workstation or even on a Mac. Here, we will be using our CentOS 6.5 machine that has the AWS CLI installed and configured in it and following the next steps, we will be able to look into our EC2 dashboard:

1. First up, transfer your private key (*.pem) file over to the Linux server using and SCP tool. In my case, I always use WinSCP to achieve this. It's a simple tool and pretty straightforward to use. Once the key is transferred, run the following command to change the key's permissions:

   ```bash
   # chmod 400 <Private_Key>.pem
   ```

2. Next up, simply connect to the remote EC2 instance by using the following SSH command. You will need to provide your EC2 instance's public DNS or its public IP address, which can be found listed on the EC2 dashboard:

   ```bash
   # ssh -I <Private_Key>.pem ec2-user@<EC2_Instance_PublicDNS>
   ```
And following is the output of the preceding command:

```bash
[ec2-user@ip-172-31-46-172 ~]$ # df -h
```

### Configuring your instances
Once your instances are launched, you can configure virtually anything in it, from packages, to users, to some specialized software or application, anything and everything goes!

Let's begin by running some simple commands first. Go ahead and type the following command to check your instance's disk size:

```bash
# df -h
```

Here is the output showing the configuration of the instance:
You should see an 8 GB disk mounted on the root (/) partition, as shown in the preceding screenshot. Not bad, eh! Let's try something else, like updating the operating system. AWS Linux AMIs are regularly patched and provided with necessary package updates, so it is a good idea to patch them from time to time. Run the following command to update the Amazon Linux OS:

```bash
# sudo yum update -y
```

Why sudo? Well, as discussed earlier, you are not provided with root privileges when you log in to your instance. You can change that by simple changing the current user to root after you login; however, we are going to stick with the ec2-user itself for now.

What else can we do over here? Well, let's go ahead and install some specific software for our instance. Since this instance is going to act as a web server, we will need to install and configure a basic Apache HTTP web server package on it.

Type in the following set of commands that will help you install the Apache HTTP web server on your instance:

```bash
# sudo yum install httpd
```

Once the necessary packages are installed, simply start the Apache HTTP server using the following simple commands:

```bash
# sudo service httpd start
# sudo chkconfig httpd on
```

You can see the server running after running the preceding commands, as shown in the following screenshot:
You can verify whether your instance is actually running a web server or not by launching a web browser on your workstation and typing either in the instance's public IP or public DNS. You should see the Amazon Linux AMI test page, as shown in the following screenshot:

![Amazon Linux AMI Test Page](image)

There you have it! A fully functional and ready-to-use web server using just a few simple steps! Now wasn't that easy!

**Launching instances using the AWS CLI**

So far, we have seen how to launch and manage instances in EC2 using the EC2 dashboard. In this section, we are going to see how to leverage the AWS CLI to launch your instance in the cloud! For this exercise, I'll be using my trusty old CentOS 6.5 machine, which has been configured from *Chapter 2, Security and Access Management*, to work with the AWS CLI. So, without further ado, let's get busy!

**Stage 1 – create a key pair**

First up, let's create a new key pair for our instance. Note that you can use existing key pairs to connect to new instances; however, we will still go ahead and create a new one for this exercise. Type in the following command in your terminal:

```
# aws ec2 create-key-pair --key-name <Key_Pair_Name> \\ > --output text > <Key_Pair_Name>.pem
```

Once the key pair has been created, remember to change its permissions using the following command:

```
# chmod 400 <Key_Pair_Name>.pem
```
And you can see the created key:

```
(root@YoYoNUX ~)# aws ec2 create-key-pair --key-name DumboKey \
> --output text > DumboKey.pem \
> --profile jason
(root@YoYoNUX ~)# chmod 400 DumboKey.pem
(root@YoYoNUX ~)#
```

**Stage 2 – create a security group**

Once again, you can very well reuse an existing security group from EC2 for your new instances, but we will go ahead and create one here. Type in the following command to create a new security group:

```
# aws ec2 create-security-group --group-name <SG_Name> \
> --description "<SG_Description>"
```

For creating security groups, you are only required to provide a security group name and an optional description field along with it. Make sure that you provide a simple yet meaningful name here:

```
(root@YoYoNUX ~)# aws ec2 create-security-group --group-name DumboSG \
> --description "This is yet another simple security group" \
> --profile jason
```

Once executed, you will be provided with the new security group’s ID as the output. Make a note of this ID as it will be required in the next few steps.
Stage 3 – add rules to your security group

With your new security group created, the next thing to do is to add a few firewall rules to it. We will be discussing a lot more on this topic in the next chapter, so to keep things simple, let's add one rule to allow inbound SSH traffic to our instance.

Type in the following command to add the new rule:

```
# aws ec2 authorize-security-group-ingress --group-name <SG_Name> \
> --protocol tcp --port 22 --cidr 0.0.0.0/0
```

To add a firewall rule, you will be required to provide the security group's name to which the rule has to be applied. You will also need to provide the protocol, port number, and network CIDR values as per your requirements:

Stage 4 – launch the instance

With the key pair and security group created and populated, the final thing to do is to launch your new instance. For this step, you will need a particular AMI ID along with a few other key essentials such as your security group name, the key pair, and the instance launch type, along with the number of instances you actually wish to launch.

Type in the following command to launch your instance:

```
# aws ec2 run-instances --image-id ami-e7527ed7 \
> --count 1 --instance-type t2.micro \
> --security-groups <SG_Name> \
> --key-name <Key_Pair_Name>
```
Images and Instances

And here is the output of the preceding commands:

```
In this case, we are using the same Amazon Linux AMI (ami-e7527ed7) that we used during the launch of our first instance using the EC2 dashboard.

The instance will take a good two or three minutes to spin up, so be patient! Make a note of the instance's ID from the output of the ec2 run-instance command. We will be using this instance ID to find out the instance's public IP address using the EC2 describe-instance command as shown:

```
# aws ec2 describe-instances --instance-ids <Instance_ID>
```

Make a note of the instance's public DNS or the public IP address. Next, use the key pair created and connect to your instance using any of the methods discussed earlier.

Cleaning up!

Spinning up instances is one thing; you should also know how to stop and terminate them! To perform any power operations on your instance from the EC2 dashboard, all you need to do is select the particular instance and click on the **Actions** tab as shown. Next, from the **Instance State** submenu, select whether you want to **Stop**, **Reboot**, or **Terminate** your instance, as shown in the following screenshot:
It is important to remember that you only have instance stopping capabilities when working with EBS-backed instances. Each time an EBS-backed instance is stopped, the hourly instance billing stops too; however, you are still charged for the EBS volume that your instance is using. Similarly, if your EBS-backed instance is terminated or destroyed, then by default the EBS root volume attached to it is also destroyed, unless specified otherwise, during the instance launch phase.

Planning your next steps

So far, all we have worked with are Linux instances, so the next step that I recommend is that you go ahead and deploy your very first Windows server instance as well. Just a few pointers worth remembering are to make sure you enable the firewall rule for RDP protocol (TCP Port 3389) in the security group and to generate the administrator password using the key pair that you create. For more in-depth steps, check out this simple tutorial at http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/EC2Win_GetStarted.html.

The second thing worth trying out are spot instances. Now, you may be wondering that spot instances seem kind of hard to grasp, but in reality they are a lot easier and cost efficient to work with. Try and spin up a simple t2.micro Linux instance using spot pricing and compare the difference with a traditional on-demand instance. To know more about spot instances, check out http://aws.amazon.com/ec2/purchasing-options/spot-instances/.

Another really cool thing worth the time and effort is the AWS Management Portal for vCenter! Yes! You heard it right! You can actually manage your AWS resources using your standard VMware vCenter Server! All you need to do is install a simple plugin and, voila, your entire AWS infrastructure can be managed using the familiar vCenter dashboard. But the fun doesn’t just stop there. You can also export your on premise virtual machines hosted on the vSphere platform over to AWS using a tool called as VM Import/ Export. Once installed within your VMware vSphere environment, you can easily migrate any Linux and Windows Server based virtual machine to your AWS account using a few simple steps! Now that's really amazing! To know more about the AWS Management Portal for vCenter, refer to http://aws.amazon.com/ec2/vcenter-portal/.

Both the AWS Management portal for vCenter as well as the VM Import/ Export tool are absolutely free of cost! You only have to pay for the AWS resources that you consume and not a penny more!
And last but not least, have some fun with configuring your instances! Don't stop just at a simple Web Server; go ahead and set up a full-fledge WordPress application on your instances or launch multiple instances and set up JBoss Clustering among them and so on. The more you configure and use the instances, the more you will get acquainted with the terms and terminologies and find out how easy it is working with AWS! Just remember to clean up your work after it is done.

**Recommendations and best practices**

Here are a few key takeaways from this chapter:

- First and foremost, create and use separate IAM users for working with EC2. *DO NOT USE* your standard root account credentials!
- Use IAM roles if you need to delegate access to your EC2 account to other people for some temporary period of time. Do not share your user passwords and keys with anyone.
- Use a standard and frequently deployed set of AMIs as they are tried and tested by AWS thoroughly.
- Make sure that you understand the difference between instance store-backed and EBS-backed AMIs. Use the instance store with caution and remember that you are responsible for your data, so take adequate backups of it.
- Don't create too many firewall rules on a single security group. Make sure that you apply the least permissive rules for your security groups.
- Stop your instances when not in use. This will help you save up on costs as well.
- Use tags to identify your EC2 instances. Tagging your resources is a good practice and should be followed at all times.
- Save your key pairs in a safe and accessible location. Use passphrases as an added layer of security if you deem it necessary.
- Monitor your instances at all times. We will be looking at instance monitoring in depth in the coming chapters; however, you don't have to wait until then! Use the EC2 Status and Health Check tabs whenever required.
Summary

So, let's wrap up what we have learnt so far! First up, we looked at what exactly the AWS EC2 service is and how we can leverage it to perform our daily tasks. Next, we understood a bit about images and instances by looking at the various instance types and pricing options provided. Finally, we also managed to launch a couple of instances in EC2 using both the EC2 dashboard as well as the AWS CLI. We topped it all off with some interesting next steps and a bunch of recommendations and best practices!

In the next chapter, we will continue with the EC2 service and explore some of the advanced network, security, and storage options that come along with it, so stay tuned!
Where to buy this book

You can buy AWS Administration – The Definitive Guide from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.