Designing AWS Environments

Architect large-scale cloud infrastructures with AWS

Packt
www.packt.com

Mitesh Soni and Wayde Gilchrist

Designing AWS Environments

Architect large-scale cloud infrastructures with AWS

Mitesh Soni
Wayde Gilchrist

BIRMINGHAM - MUMBAI

Designing AWS Environments

Commissioning Editor: Vijin Boricha
Acquisition Editor: Rahul Nair
Content Development Editor: Deepti Thore
Technical Editor: Sayali Thanekar
Copy Editor: Safis Editing
Project Coordinator: Kinjal Bari
Proofreader: Safis Editing
Indexer: Mariammal Chettiyar
Graphics: Jisha Chirayil
Production Coordinator: Nilesh Mohite

First published: September 2018

Production reference: 1290918

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78953-554-9

www.packtpub.com

mapt.io

Mapt is an online digital library that gives you full access to over 5,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Mapt is fully searchable
- Copy and paste, print, and bookmark content

Packt.com

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.packt.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `customercare@packtpub.com` for more details.

At `www.packt.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Contributors

About the authors

Mitesh Soni is an avid learner with 10 years' experience in the IT industry. He is an SCJP, SCWCD, and VCP. He is IBM Urbancode- and IBM Bluemix-certified, and is also a Certified Jenkins Engineer. He loves DevOps and cloud computing, and he also has an interest in programming in Java. He finds design patterns fascinating and believes that a picture is worth a thousand words. He occasionally contributes to clean-clouds and e-Tutorials World websites. He loves to play with his kids, fiddle with his camera, and take photographs at Indroda Park.

> *Dedicated to Shreyu, Jigi, Parents, Grand Parents, Priyanka, Varsha, Radhika+Mukund, Mayur, Ashish, Navrang, Dharmesh, Vinay Kher, Yohan, Rohini, Teachers, Anupama+Mihir and Priyanka+Hemant, Sourabh, Gowri, Sudeep, Aishwarya, and Rohan.*

Wayde Gilchrist started moving customers of his IT consulting business into the cloud and away from traditional hosting environments back in 2010. In addition to consulting, he delivers AWS training for Fortune 500 companies, government agencies, and international consulting firms. When he is not out visiting customers, he is delivering training virtually from his home in Florida.

About the reviewer

Sunil Gulabani is a software engineer based in India. He is currently working on Java EE and the AWS cloud platform. He is also a cloud evangelist who helps IT professionals to leverage the AWS cloud platform for their business needs. He has insightful knowledge on designing microservices, system architecture and integration, data modeling, relational databases, and NoSQL, so as to enable applications to achieve high throughput.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Preface

Amazon Web Services (**AWS**) provides trusted, cloud-based solutions to help you meet your business needs. Running your solutions on AWS can help you get your applications up and running faster while providing the security necessary to meet your compliance requirements.

This book begins by familiarizing you with the key capabilities to architect and host applications, websites, and services on AWS. We'll explain the available options for virtual instances and demonstrate launching and connecting to them. Using practical examples, you will be able to design and deploy networking and hosting solutions for large deployments. Finally, the book focuses on security and the important elements of scalability and high availability.

Who this book is for

This book is for new and aspiring individuals who are gearing up for a solutions architect role. You'll also find this useful if you're an IT professional or DevOps engineer who is preparing to design and deploy large solutions on AWS. No experience with AWS is required.

What this book covers

`Chapter 1`, *Installation and Setup*, helps us understand how to sign up for a free AWS account, and how to use the Management Console.

`Chapter 2`, *Launching an EC2 Instance*, provides us with information on how to launch an EC2 instance, and during that process, we will learn about AMIs, instance types, storage options, and security groups.

`Chapter 3`, *Logging in to EC2 Instances*, teaches us about key pairs, which we will then use to authenticate an SSH to Linux instances. Finally, we will use them to decrypt the administrator password and remote desktop to a Windows instance.

`Chapter 4`, *Networking on AWS*, covers designating private IP address ranges for your VPC. We also cover the three types of IP address used for EC2s, as well as elastic network interfaces, which hold the instance's network attributes for IP connections. We discuss subnets and route tables, and how a route in the route table can make a subnet public or private. Then, we'll talk about NAT instances and NAT gateways, to give instances and private subnets access to the internet.

`Chapter 5`, *Creating a VPC*, covers classic EC2s and EC2s in a VPC, using the default VPC, creating your own VPC with the VPC wizard or from scratch, connecting to your VPC, and securing your VPC with network ACLs, bastions, and NAT instances. Finally, we cover making your architecture highly available by means of multiple availability zones, elastic load balancing, and auto scaling.

To get the most out of this book

Before starting to read the book, basic knowledge of networking concepts, basic knowledge of cloud computing, cloud service models, and cloud deployment models, basic knowledge of Linux and Windows operating systems would be useful.

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://www.packtpub.com/sites/default/files/downloads/9781789535549_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "A `/16` is a typical size, and this gives your VPC `65536` private IP addresses it can use."

Any command-line input or output is written as follows:

```
$ sudo yum update
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "We could, of course, switch the **Volume Type** to **Provisioned IOPS SSD (io1)**, or use the cheaper **Magnetic (standard)** storage."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packt.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

1
Installation and Setup

You will need an AWS account in order to follow the examples in this book. Since the account is free to open, you need not worry about payment for now. However, the free tier has defined usage limits for specific services. If your usage of these services exceeds the monthly quota for 12 months, you will need to pay the pay-as-you-go AWS service rates.

In this chapter, we will describe the process of opening an AWS account, including adding your payment information. Then we will discuss the free tier and demonstrate all the things you can do for free in the first year and beyond. Finally, we will introduce you to the AWS Management Console.

Here are the topics we'll cover:

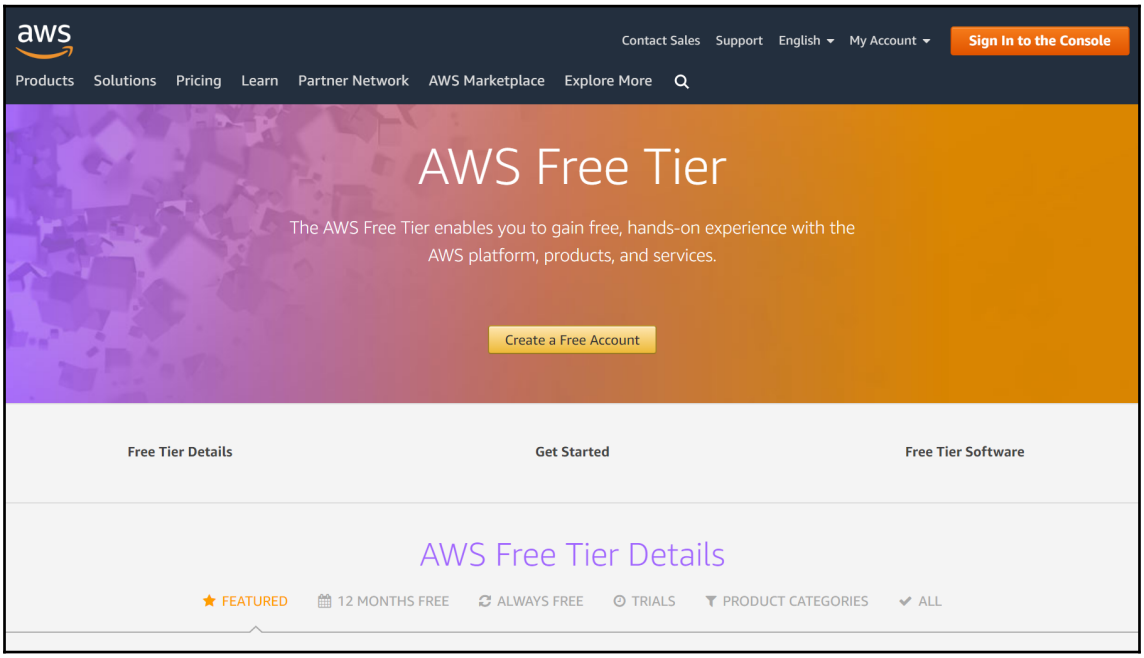
- Creating an AWS account
- The free tier
- The AWS Management Console

Let's begin by showing you how to open your AWS account.

Opening an AWS account

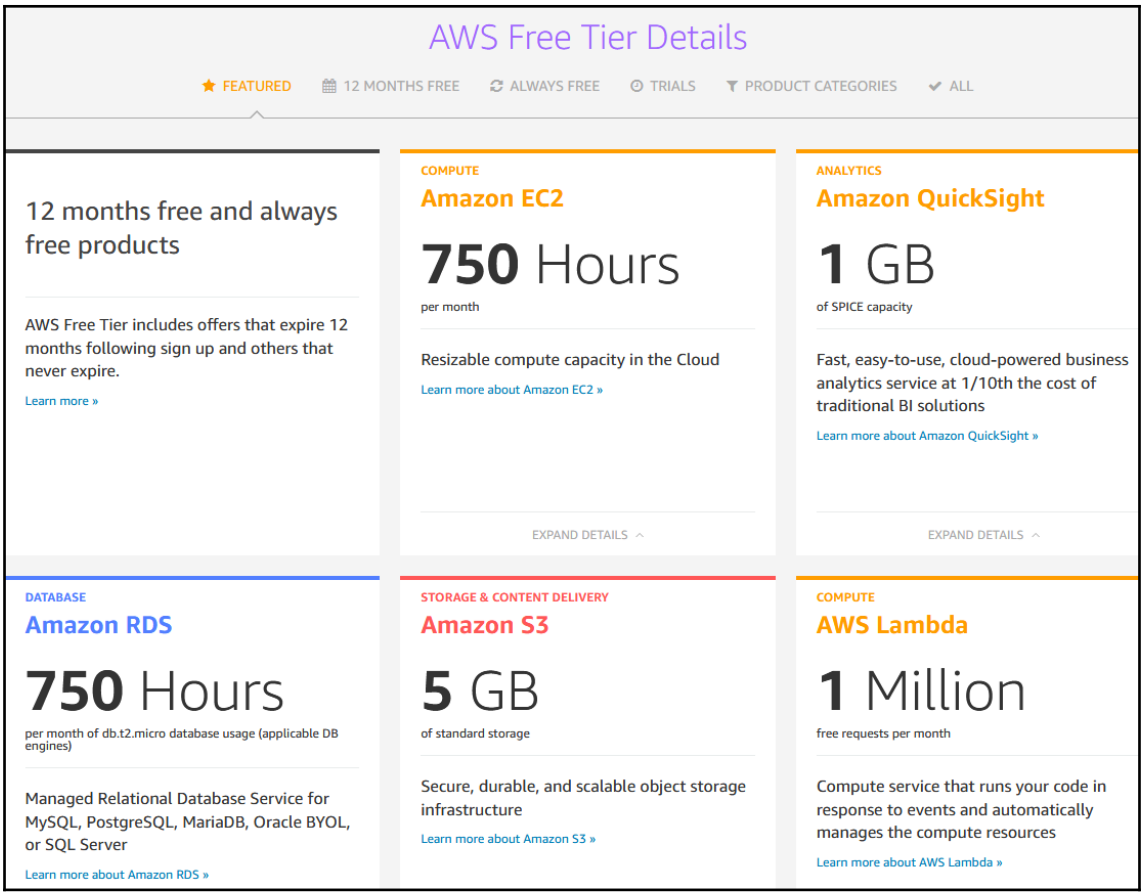
In this section, we're going to describe how to sign up for an AWS account and add your payment information:

1. Begin by visiting `https://aws.amazon.com/free/`:



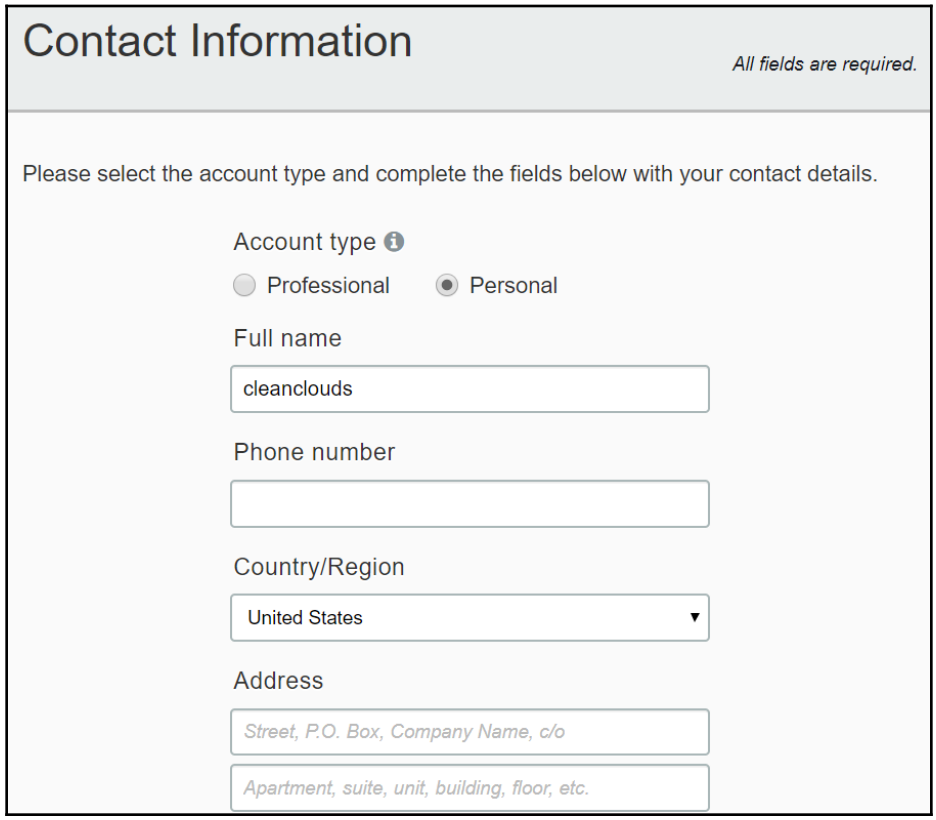
2. You can see there are different categories available in the free-tier:

- **FEATURED**
- **12 MONTHS FREE**
- **ALWAYS FREE**
- **TRIALS**
- **PRODUCT CATEGORIES**



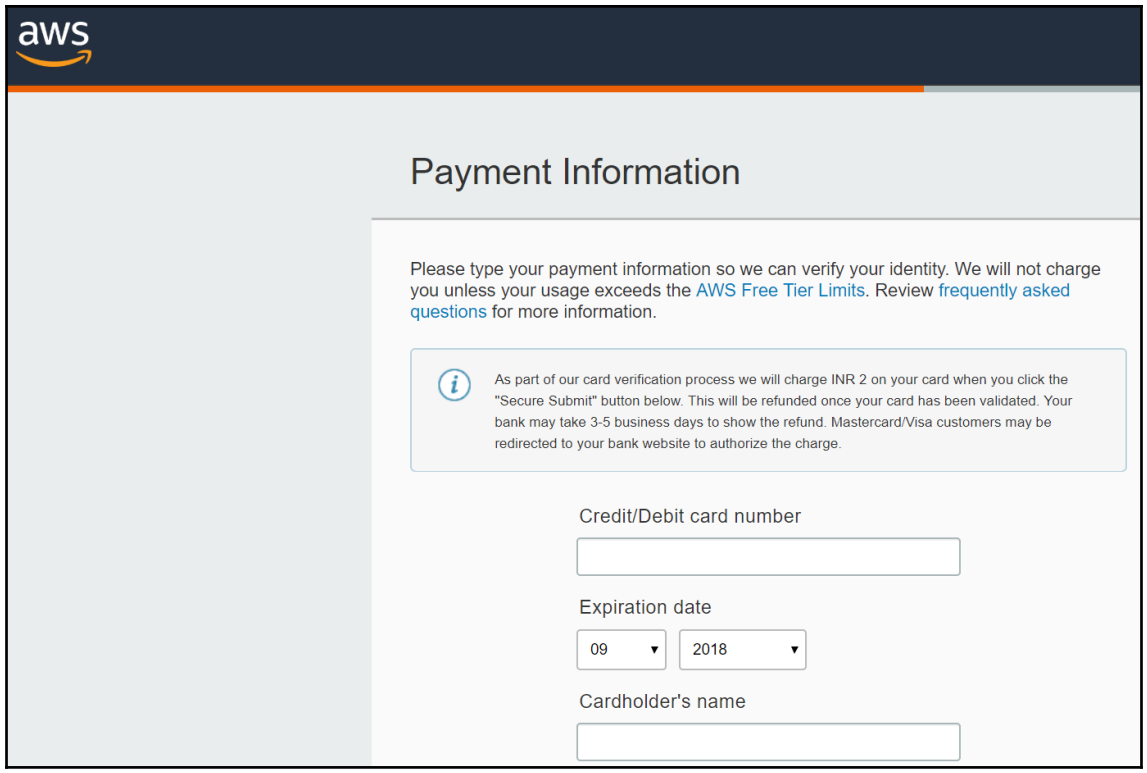
You can find more details about **AWS Free Tier (Non-expiring Offers)** for products such as **Amazon DynamoDB**, **Amazon Cognito**, **Amazon CodeCommit**, **AWS Lambda**, and so on can be found on `https://aws.amazon.com/free/`. You will also get to know about **AWS Free Tier (12 Month Introductory Period)** offers for products such as **Elastic Compute Cloud (EC2)**, **Amazon Simple Storage Service (S3)**, **Amazon Elastic File System (EFS)**, **Amazon CloudFront** and so on. Furthermore you can also check what **AWS Free Tier (12 Month Introductory Period)** offers for products such as **Amazon Relational Database Service (RDS)**, **Amazon CloudDirectory**, **AWS IOT**, and so on, on the same page.

3. Click **Create an AWS Account**, or **Create a Free Account** (either button will take you to the same place).
4. Select the tier with the **12 Months Free** option.
5. Now enter your email address, and choose a password and AWS account name. Click on **Continue**.
6. Fill in your contact information by selecting an **Account type**. You will probably want to make this a personal account, so click **Personal**. Now enter your **Full name** again, select your **Country/Region**, and enter your **Address** and your **Phone number**:

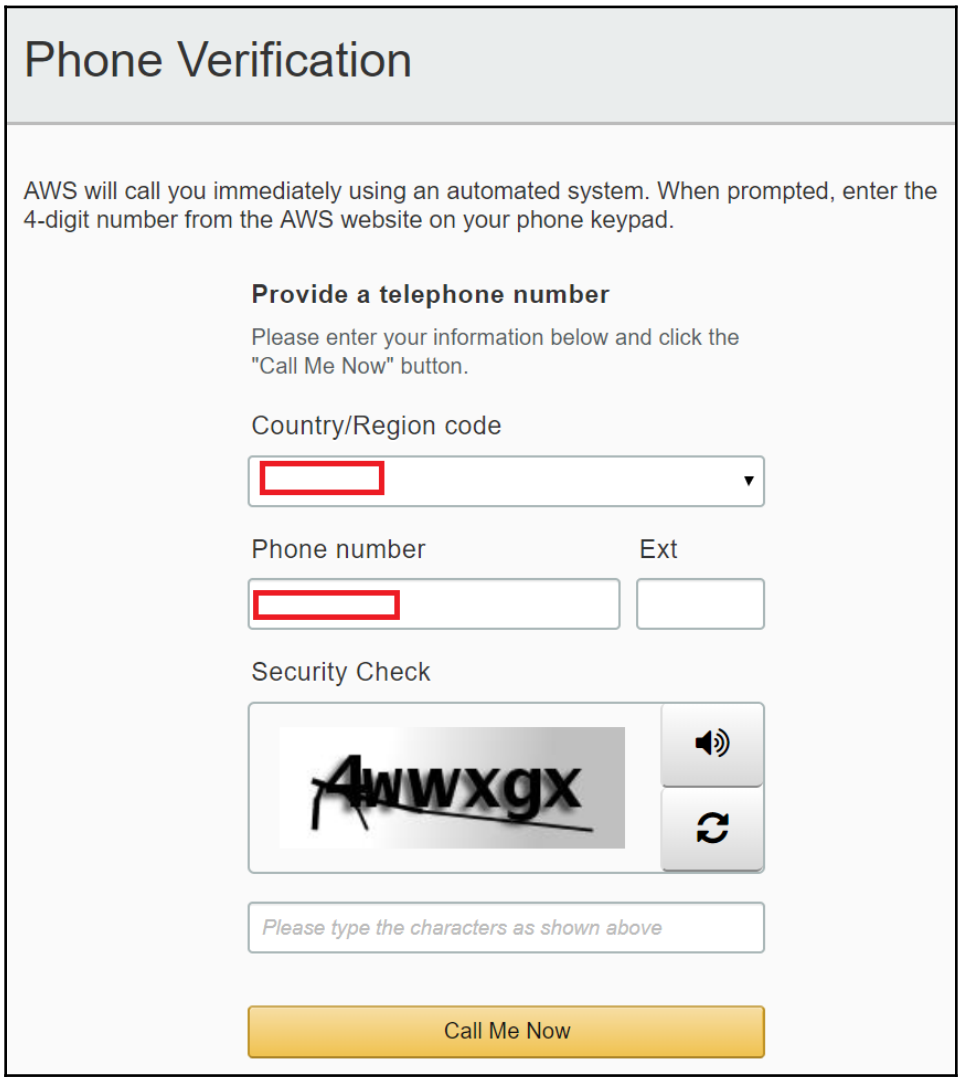


Provide **City**, and **Postal code**, details check the agreement details, and click on **Create Account and Continue**. You probably should review this first, and when you are satisfied, then click Create **Account and Continue**.

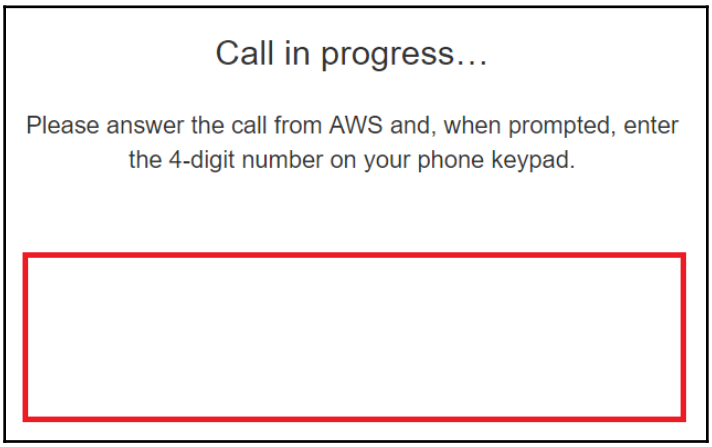
7. Next, enter your credit card number. Even though this is a free account, you still need to provide billing information. Once you have filled in your credit card information, click on **Continue**. Provide **Payment Information** and **Authenticate Transaction** details:



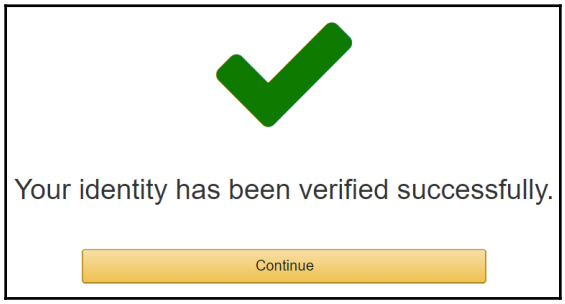
8. To verify your information, you need to let AWS call you. So, enter your phone number, and then click **Call Me Now**:



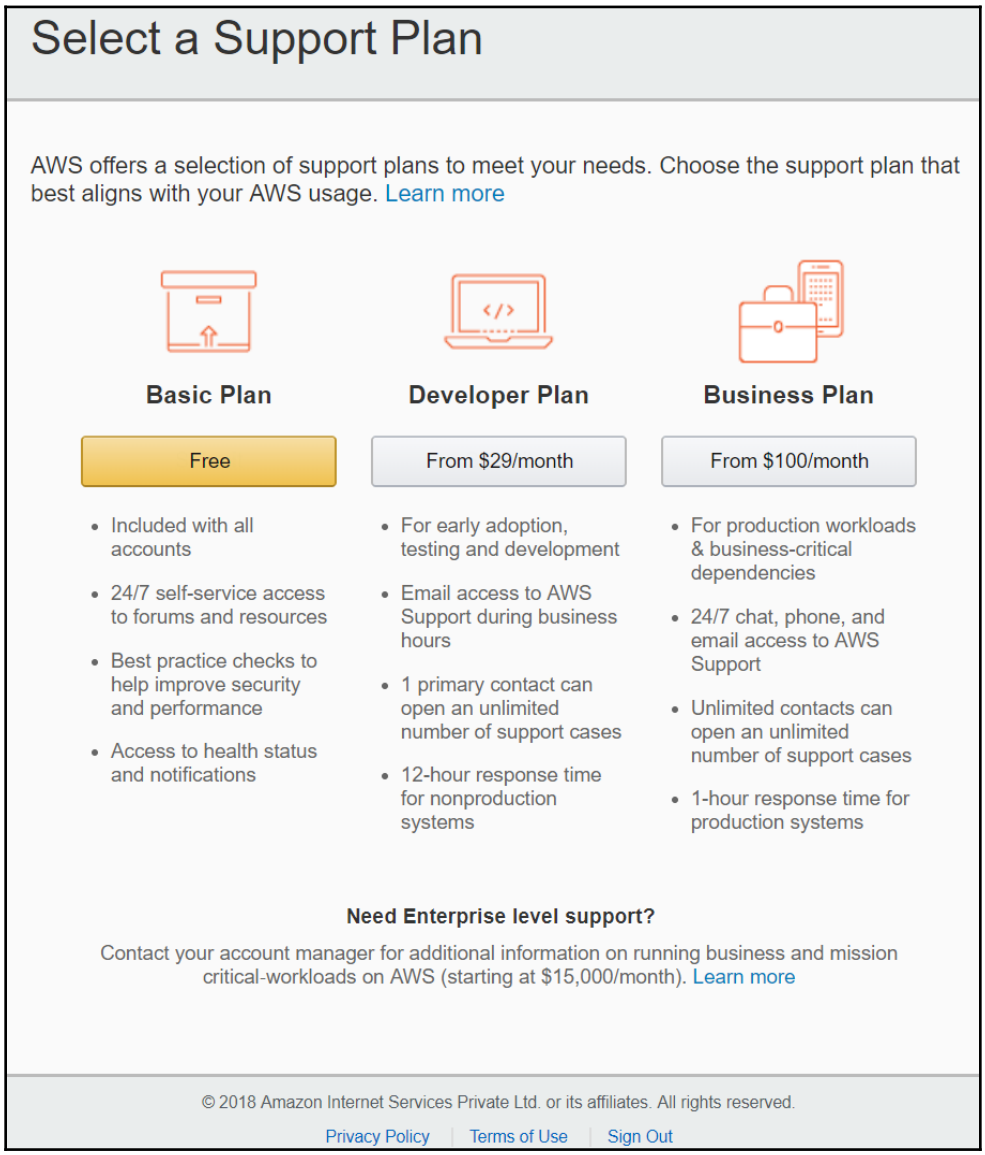
9. You will receive a phone call giving you a PIN number, which you need to enter on the screen:



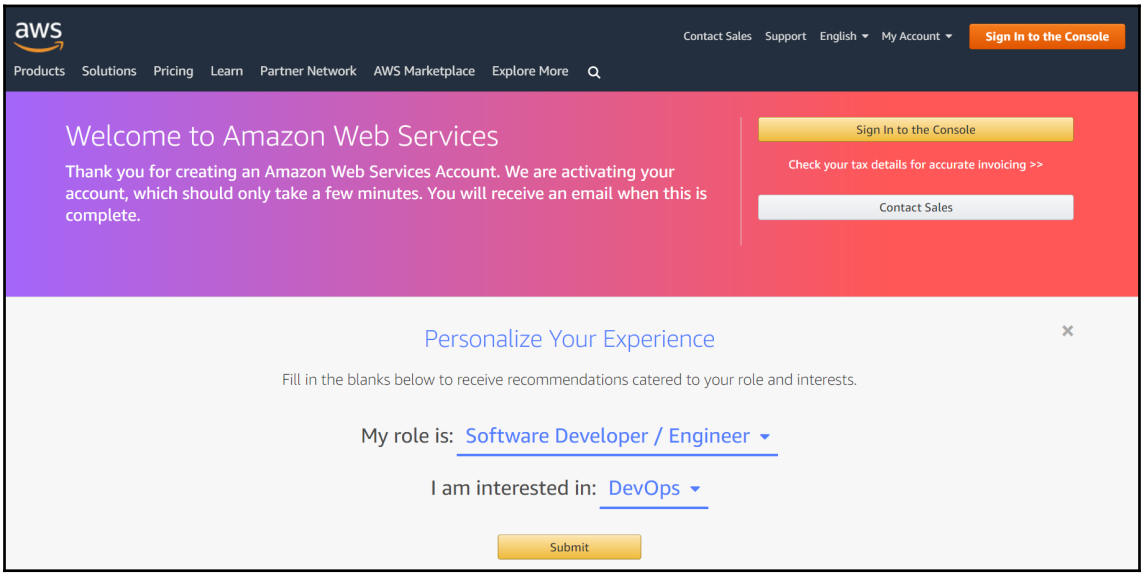
10. After your identity is verified, click **Continue**:



11. After the phone call, and your identity has been verified, click **Continue** the select your support plan:



12. The next screen lets you **Personalize Your Experience**. Select a role and the service you'are interested in services:

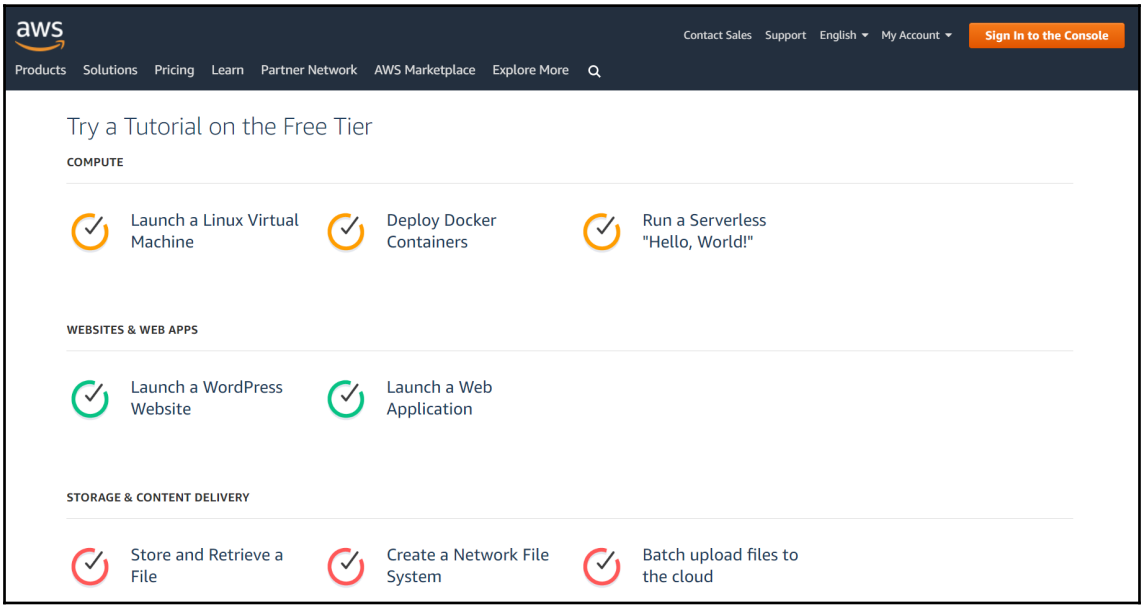


Congratulations, you now have an AWS account! A confirmation message will be sent to your email address.

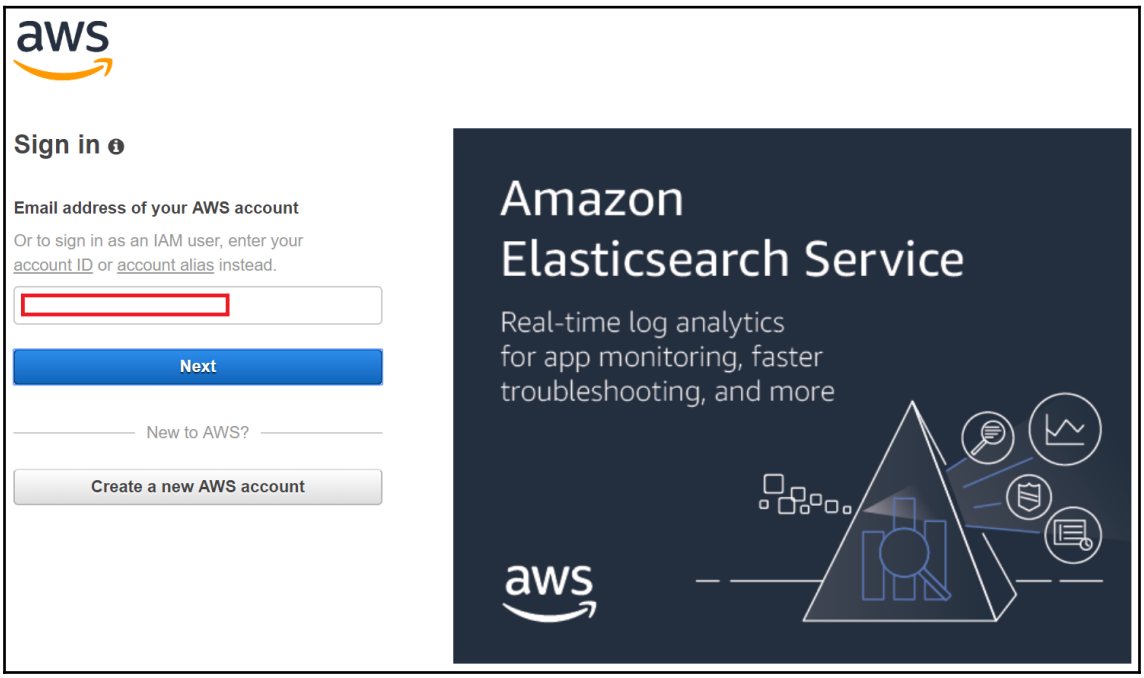
The AWS Management Console

Now that you have an AWS account, you're ready to begin using the Management Console. In this section, we're going to describe three ways to log into the Console, navigating, viewing your build, and switching between regions.

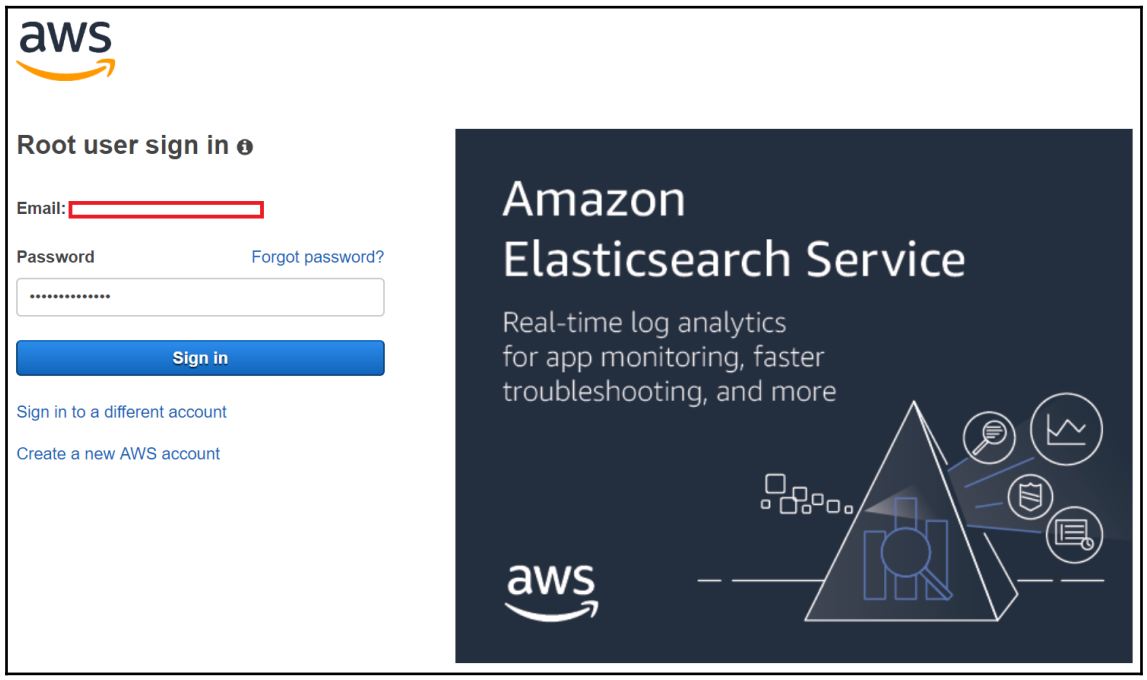
You can log into the console by visiting `https://aws.amazon.com/` and clicking **Sign In to the Console**:



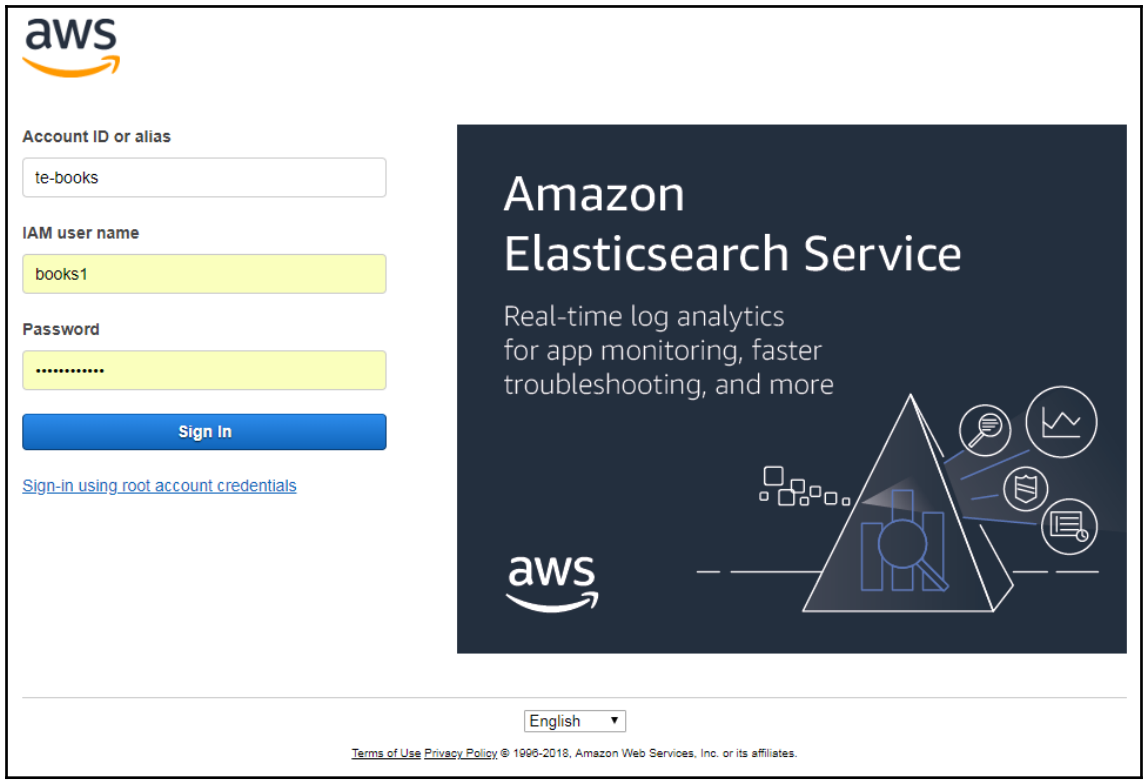
If your screen looks like the following screenshot, then you are signing in with your root account credentials. Simply enter an e-mail address and click on **Next**:



Enter the password you used when you created your AWS account, and click the **Sign In** button:



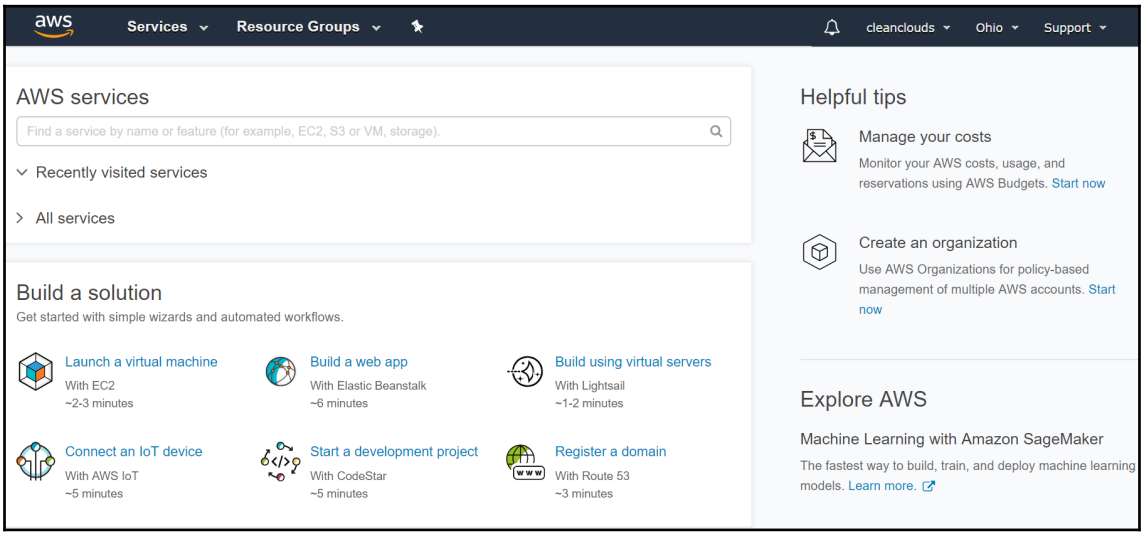
If you are signing in as an IAM user, your screen will look like the following:



IAM users are created in the console to grant login and management permissions to others. If you would rather log in as your root account, click the link under **Sign In** that reads **Sign-in using root account credentials**.

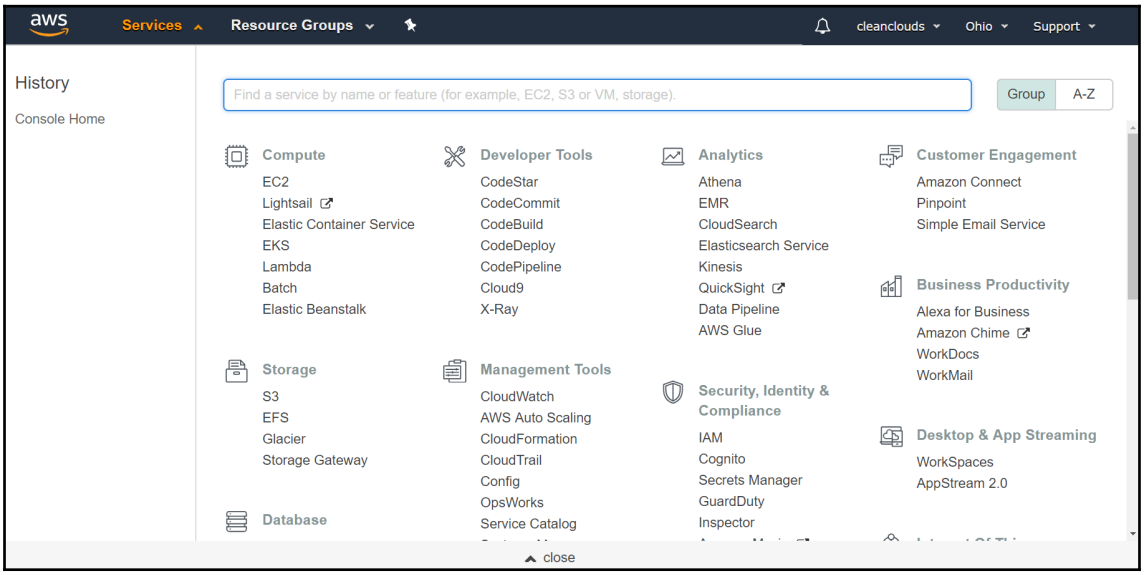
Otherwise, click **Sign In** after you enter your username and password.

After you log in, you will see the dashboard:



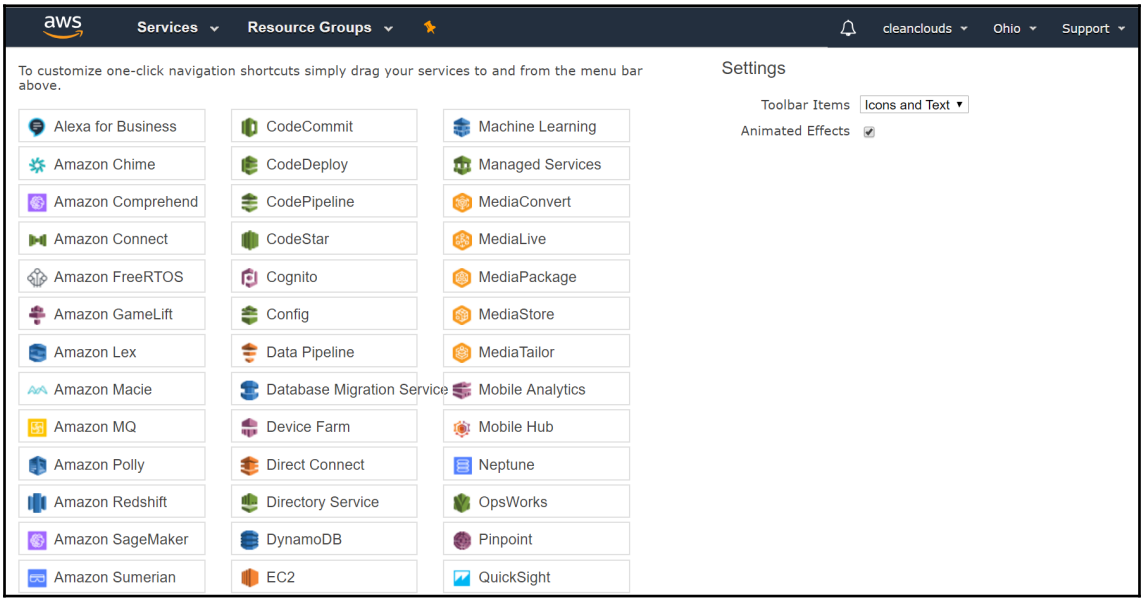
The dashboard is a quick way to locate AWS services.

The **Services** menu is another way to locate AWS services:

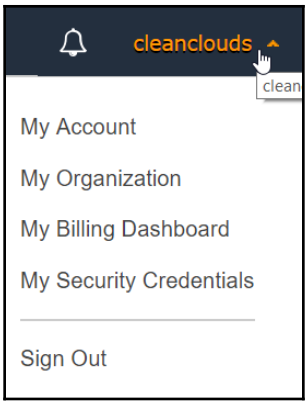


It includes a recent **History** of services used, and more detailed descriptions of services.

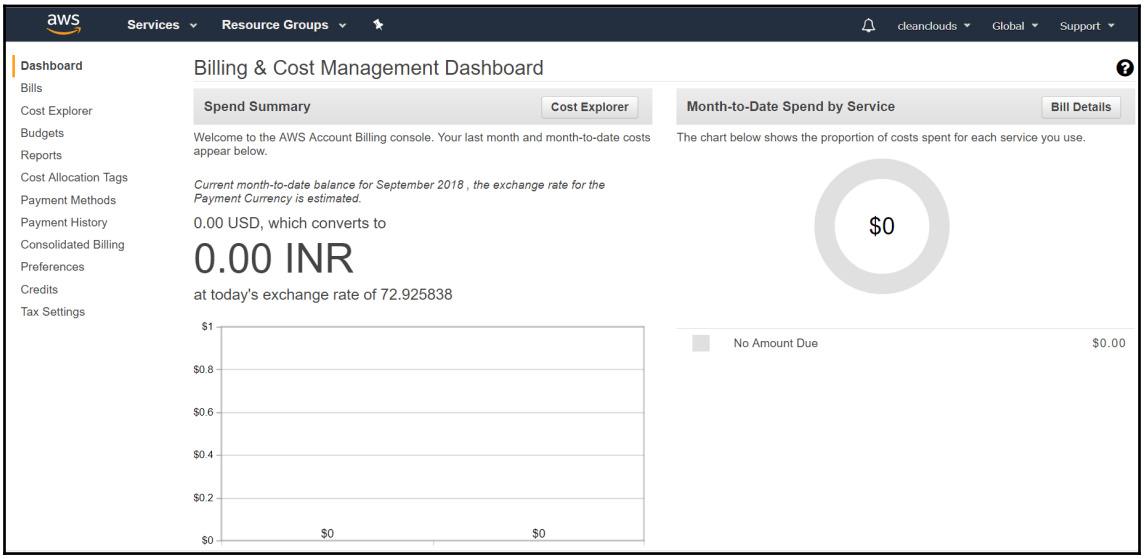
Next, under the Edit icon, any services that you use frequently can be dragged to the top bar, where you can easily find them:



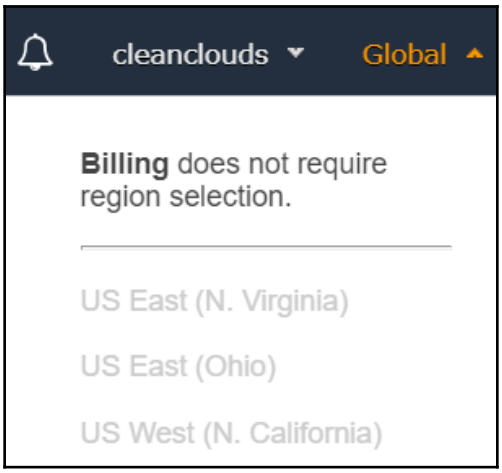
If you click on your name, you will see a submenu that will allow you to access your billing information:



This is where you find your AWS charges for this month the last month, and also a projection of next month's charges:

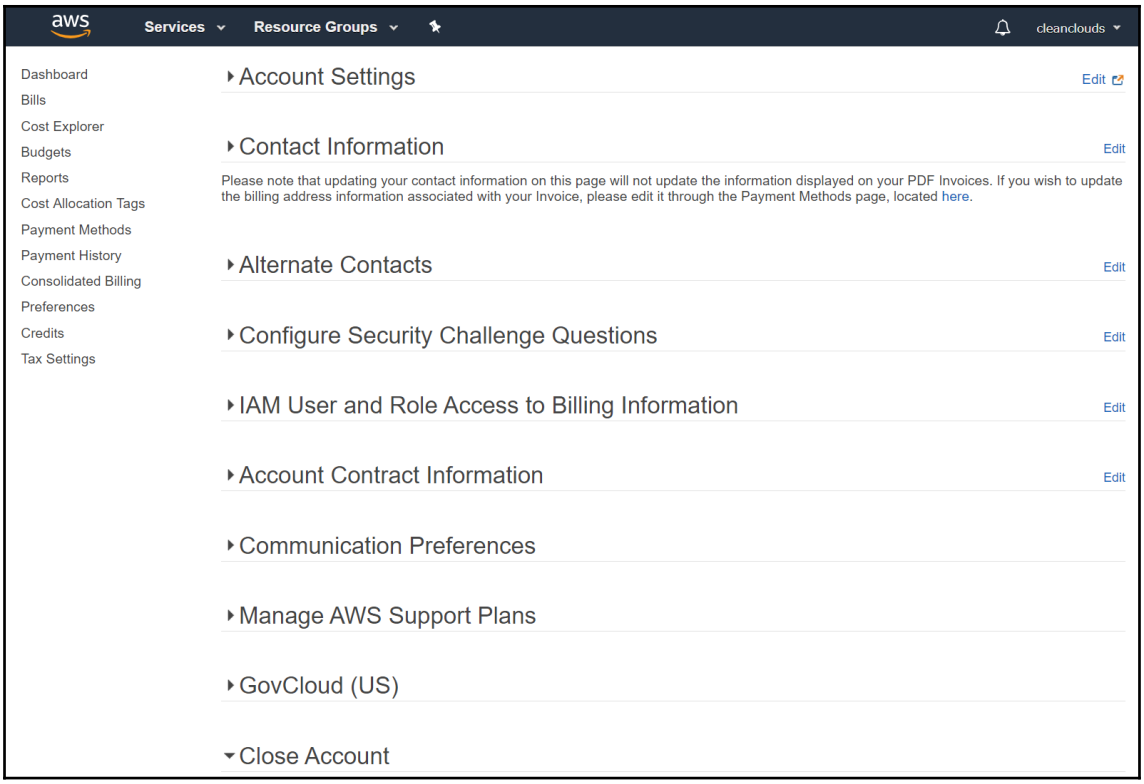


Next to your name, you will see a drop down for region selection:



Some services, such as billing, IAM, CloudFront, and Route 53, do not require you to select a region. However, most other services will require you to select a region from this drop-down.

You can close the account from **My Billing Dashboard**:



AWS free tier does not automatically expire at the end of your 12-month term. However, Microsoft Windows Server 2008 R2 with SQL Server Web, Microsoft Windows Server 2008 R2 with SQL Server Standard, Microsoft Windows 2008 R2 64-bit for Cluster Instances, and Microsoft Windows 2008 R2 SQL Server 64-bit for Cluster Instances are not eligible for the free tier. Currently, the free tier is not available in the China (Beijing) region.

Summary

In this chapter, we described how to open a free AWS account. We also discussed the AWS feature that you can use for free. We introduced you to the AWS Management Console. In `Chapter 2`, *Launching an EC2 Instance*, we'll discuss virtual servers on AWS, known as EC2.

2
Launching an EC2 Instance

In this chapter, we will see how to launch an EC2 instance using your own AWS account. We will discuss some of the choices you will have to make when you launch an instance, including choosing **Amazon Machine Images** (**AMIs**), an instance type and size, storage options, and configuring your security group.

The topics that we will cover are as follows:

- Instance types
- Storage
- Security groups
- AMIs

EC2 instance types

In this section, we're going to take a look at selecting the instance type and size as we continue launching our instance. We'll discuss the general types their features, and some good use cases for each.

General purpose instance

The first instance we're going to discuss is a general purpose instance type known as T3s and T2s.

T3s – burstable general-purpose instance type

T3 instances are good candidates for the development environment, micro-services, and so on.

Here are some of the features of T3:

- Based on Intel® Xeon® Scalable processors and the AWS Nitro System
- They provide consistent baseline performance

Before we move on, we should note that instance types are designated by a letter followed by a number, which is the generation of the instance type:

Model	vCPU	CPU credits/hour	Mem (GiB)	Storage
t3.nano	2	6	0.5	EBS-only
t3.micro	2	12	1	EBS-only
t3.small	2	24	2	EBS-only
t3.mediun	2	24	4	EBS-only
t3.large	2	36	8	EBS-only
t3.xlarge	4	96	16	EBS-only
t3.2xlarge	8	192	32	EBS-only

T2s – burstable general-purpose instance type

For proper workloads, such as development environments, T2 instances should never run out of credits. These are the only types of burstable CPU capacity allocation.

Here are some of the advantages of T2:

- Free tier-eligible (t2.micro)
- T2s are the least expensive instance type
- Instead of dedicated CPU capacity, these instances share CPU capacity with other instances
- They operate at a low baseline CPU performance, and accumulate credits when idle
- When they need to use the CPU, they can burst up to the full utilization of the virtual CPUs until they exhaust their credits

Before we move on, we should note that instance types are designated by a letter followed by a number, which is the generation of the instance type:

Model	vCPU	CPU Credits / hour	Mem (GiB)	Storage
t2.nano	1	3	0.5	EBS-only
t2.micro	1	6	1	EBS-only
t2.small	1	12	2	EBS-only
t2.medium	2	24	4	EBS-only
t2.large	2	36	8	EBS-only
t2.xlarge	4	54	16	EBS-only
t2.2xlarge	8	81	32	EBS-only

So, T2 designates the second generation of the T instances.

Another general-purpose instance type is the M's. These are the latest ones.

M5

M5 instances are utilized for enterprise applications, data processing tasks, and small and medium size databases.

Here are some of the advantages of M5:

- They provide enhanced networking—up to 25 Gbps network bandwidth
- They provide instance storage using EBS
- They provide larger instance sizes

You can see the example here for the different type of m5 models:

Model	vCPU	Mem (GiB)	Instance storage (GiB)	Dedicated EBS bandwidth (Mbps)
m5.large	2	8	EBS-only	Up to 3,500
m5.xlarge	4	16	EBS-only	Up to 3,500
m5.2xlarge	8	32	EBS-only	Up to 3,500
m5.4xlarge	16	64	EBS-only	3500
m5.12xlarge	48	192	EBS-only	7000
m5.24xlarge	96	384	EBS-only	14000
m5d.large	2	8	1 x 75 NVMe SSD	Up to 3,500
m5d.xlarge	4	16	1 x 150 NVMe SSD	Up to 3,500
m5d.2xlarge	8	32	1 x 300 NVMe SSD	Up to 3,500
m5d.4xlarge	16	64	2 x 300 NVMe SSD	3500
m5d.12xlarge	48	192	2 x 900 NVMe SSD	7000
m5d.24xlarge	96	384	4 x 900 NVMe SSD	14000

M4

M4s are very suitable for small-and mid-sized databases and many web applications.

Here are some of the advantages of M4:

- M4s use the advanced Broadwell or Haswell architecture microprocessor
- They are EBS optimized, which means they will provide consistent network connectivity to EBS volumes
- They also support enhanced networking, which provides higher bandwidth, higher number of packets per second, and consistently lower latencies between instances
- They do not provide local instance storage

Here you can see list of m4 models with its details:

Model	vCPU	Mem (GiB)	SSD storage (GB)	Dedicated EBS bandwidth (Mbps)
m4.large	2	8	EBS-only	450
m4.xlarge	4	16	EBS-only	750
m4.2xlarge	8	32	EBS-only	1000
m4.4xlarge	16	64	EBS-only	2000
m4.10xlarge	40	160	EBS-only	4000
m4.16xlarge	64	256	EBS-only	10000

Compute optimized

The first compute optimized instance we're going to look at is the C5s.

C5

C5 instances are utilized for machine learning, deep learning, batch processing, HPC, and so on.

Here are some of its features:

- They provide low cost and high-performance workloads
- They provide enhanced networking—up to 25 Gbps network bandwidth
- They provide instance storage using EBS
- They provide larger instance sizes

Lets take a look at c5 models and its specifications:

Model	vCPU	Mem (GiB)	Instance storage (GiB)	Dedicated EBS bandwidth (Mbps)
c5.large	2	4	EBS-only	Up to 3,500
c5.xlarge	4	8	EBS-only	Up to 3,500
c5.2xlarge	8	16	EBS-only	Up to 3,500
c5.4xlarge	16	32	EBS-only	3500
c5.9xlarge	36	72	EBS-only	7000
c5.18xlarge	72	144	EBS-only	14000
c5d.large	2	4	1 x 50 NVMe SSD	Up to 3,500
c5d.xlarge	4	8	1 x 100 NVMe SSD	Up to 3,500
c5d.2xlarge	8	16	1 x 200 NVMe SSD	Up to 3,500
c5d.4xlarge	16	32	1 x 400 NVMe SSD	3500
c5d.9xlarge	36	72	1 x 900 NVMe SSD	7000
c5d.18xlarge	72	144	2 x 900 NVMe SSD	14000

C4

C4 instances are utilized for high-performance front end fleets, analytics, and batch processing.

Here are some of its features:

- They use the most powerful Haswell processors
- They are EBS-optimized and support enhanced networking and clustering
- They do not come with any instance storage

Here is a list of c4 models:

Model	vCPU	Mem (GiB)	Storage	Dedicated EBS bandwidth (Mbps)
c4.large	2	3.75	EBS-only	500
c4.xlarge	4	7.5	EBS-only	750
c4.2xlarge	8	15	EBS-only	1000
c4.4xlarge	16	30	EBS-only	2000
c4.8xlarge	36	60	EBS-only	4000

Memory-optimized

Next up are the memory optimized instance types.

X1e

X1e instances are utilized in memory databases and high-performance databases.

Some of its features are:

- Comparatively less costly
- By default EBS-optimized.

Lets take a look at x1e models with its specifications:

Model	vCPU	Mem (GiB)	SSD storage (GB)	Dedicated EBS bandwidth (Mbps)
x1e.xlarge	4	122	1 x 120	500
x1e.2xlarge	8	244	1 x 240	1000
x1e.4xlarge	16	488	1 x 480	1750
x1e.8xlarge	32	976	1 x 960	3500
x1e.16xlarge	64	1952	1 x 1,920	7000
x1e.32xlarge	128	3904	2 x 1,920	14000

X1

X1 instances are for in-memory databases, SAP HANA, and big data processing.

Some of its features are as follows:

- Haswell processors
- SSD-based instance storage
- Lowest price per GB of RAM.

Here is an example of x1 instance model:

Model	vCPU	Mem (GiB)	SSD storage (GB)	Dedicated EBS bandwidth (Mbps)
x1.16xlarge	64	976	1 x 1,920	7000
x1.32xlarge	128	1952	2 x 1,920	14000

R5

R5 instances are utilized for enterprise applications, in-memory databases, and big data analytics.

Here are some of its features:

- They use the Intel Xeon Platinum 8000 series
- No virtualization overhead

Lets take a look at some of the R5 instances:

Model	vCPU	Mem (GiB)	Networking perf.	SSD storage (GB)
r5.large	2	16	Up to 10 Gigabit	EBS-only
r5.xlarge	4	32	Up to 10 Gigabit	EBS-only
r5.2xlarge	8	64	Up to 10 Gigabit	EBS-only
r5.4xlarge	16	128	Up to 10 Gigabit	EBS-only
r5.12xlarge	48	384	10 Gigabit	EBS-only
r5.24xlarge	96	768	25 Gigabit	EBS-only
r5d.large	2	16	Up to 10 Gigabit	1 x 75 NVMe SSD
r5d.xlarge	4	32	Up to 10 Gigabit	1 x 150 NVMe SSD
r5d.2xlarge	8	64	Up to 10 Gigabit	1 x 300 NVMe SSD
r5d.4xlarge	16	128	Up to 10 Gigabit	2 x 300 NVMe SSD
r5d.12xlarge	48	384	10 Gigabit	2 x 900 NVMe SSD
r5d.24xlarge	96	768	25 Gigabit	4 x 900 NVMe SSD

R4

R4 instances are utilized for enterprise applications, in-memory databases, and Hadoop clusters.

Here are some of its features:

- They have Broadwell processors
- Support DDR4 memory
- Enhanced networking

Below you can find R4 instances product details:

Model	vCPU	Mem (GiB)	Networking perf.	SSD storage (GB)
r4.large	2	15.25	Up to 10 Gigabit	EBS-only
r4.xlarge	4	30.5	Up to 10 Gigabit	EBS-only
r4.2xlarge	8	61	Up to 10 Gigabit	EBS-only
r4.4xlarge	16	122	Up to 10 Gigabit	EBS-only
r4.8xlarge	32	244	10 Gigabit	EBS-only
r4.16xlarge	64	488	25 Gigabit	EBS-only

z1d

z1d instances are utilized for design automation and relational database workloads.

Here are some of its features:

- They provide high compute capacity and a high memory
- No virtualization overhead
- Fastest cloud instances

Lets look at z1d product details:

Model	vCPU	Mem (GiB)	Networking perf.	SSD storage (GB)
z1d.large	2	16	Up to 10 Gigabit	1 x 75 NVMe SSD
z1d.xlarge	4	32	Up to 10 Gigabit	1 x 150 NVMe SSD
z1d.2xlarge	8	64	Up to 10 Gigabit	1 x 300 NVMe SSD
z1d.3xlarge	12	96	Up to 10 Gigabit	1 x 450 NVMe SSD
z1d.6xlarge	24	192	10 Gigabit	1 x 900 NVMe SSD
z1d.12xlarge	48	384	25 Gigabit	2 x 900 NVMe SSD

Accelerated computing – general purpose GPU instances

If your applications can benefit from GPU acceleration, G2s come with NVIDIA GPUs with an onboard video encoder. One example is the Intel Xeon E5-2670 (Sandy Bridge) processors. These have SSD-based instance storage, and 3D application streaming and video encoding.

P3

P3 instances are utilized for high-performance computing and speech recognition. They are general purpose GPU instances.

Below you can find P3 instance model details:

Model	GPUs	vCPU	Mem (GiB)	GPU mem (GiB)	GPU P2P
p3.2xlarge	1	8	61	16	-
p3.8xlarge	4	32	244	64	NVLink
p3.16xlarge	8	64	488	128	NVLink

P2

P2 instances are utilized for machine learning and high-performance databases.

Here are some of its features:

- These instances are by default EBS-optimized
- They support enhanced networking

Below you can find P2 instance model details:

Model	GPUs	vCPU	Mem (GiB)	GPU memory (GiB)
p2.xlarge	1	4	61	12
p2.8xlarge	8	32	488	96
p2.16xlarge	16	64	732	192

G3

G3 instances are utilized for 3D rendering, application streaming, and video encoding.

Here are some of its features:

- Optimized for graphics-intensive usage
- Enhanced networking using the elastic network adapter

Below you can find G3 instance model details:

Model	GPUs	vCPU	Mem (GiB)	GPU memory (GiB)
g3.4xlarge	1	16	122	8
g3.8xlarge	2	32	244	16
g3.16xlarge	4	64	488	32

F1

F1 instances are utilized for video processing and analytic:

Here are some of its features:

- They provide customizable hardware acceleration with field programmable gate arrays
- Enhanced networking

Below you can find F1 instance model details

Model	FPGAs	vCPU	Mem (GiB)	SSD storage (GB)	Networking performance
f1.2xlarge	1	8	122	470	Up to 10 Gigabit
f1.16xlarge	8	64	976	4 x 940	25 Gigabit

Storage-optimized instance types

There are three storage optimized instance types

H1

H1 instances are utilized for big data workload clusters, distributed file systems, and network file systems.

Here are some of its features:

- H1 instances provides high disk throughput
- Enhanced Networking with ENA

Please see the details of H1 instances here:

Model	vCPU	Mem (GiB)	Networking performance	Storage (GB)
h1.2xlarge	8	32	Up to 10 Gigabit	1 x 2,000 HDD
h1.4xlarge	16	64	Up to 10 Gigabit	2 x 2,000 HDD
h1.8xlarge	32	128	10 Gigabit	4 x 2,000 HDD
h1.16xlarge	64	256	25 Gigabit	8 x 2,000 HDD

I3

I3 instances are utilized for NoSQL databases and in-memory databases.

Here are some of its features:

- It provides Broadwell processors
- **Elastic network adapter** (**ENA**)-based enhanced networking

Refer to the below I3 instance details:

Model	vCPU	Mem (GiB)	Networking performance	Storage (TB)
i3.large	2	15.25	Up to 10 Gigabit	1 x 0.475 NVMe SSD
i3.xlarge	4	30.5	Up to 10 Gigabit	1 x 0.95 NVMeSSD
i3.2xlarge	8	61	Up to 10 Gigabit	1 x 1.9 NVMe SSD
i3.4xlarge	16	122	Up to 10 Gigabit	2 x 1.9 NVMe SSD
i3.8xlarge	32	244	10 Gigabit	4 x 1.9 NVMe SSD
i3.16xlarge	64	488	25 Gigabit	8 x 1.9 NVMe SSD
i3.metal	72*	512	25 Gigabit	8 x 1.9 NVMe SSD

D2

D2 instances are utilized for MapReduce and Hadoop-distributed computing, distributed file systems and network file systems.

Here are some of its features:

- They provide high disk throughput
- They provide high-performance at launch time
- They provide enhanced networking

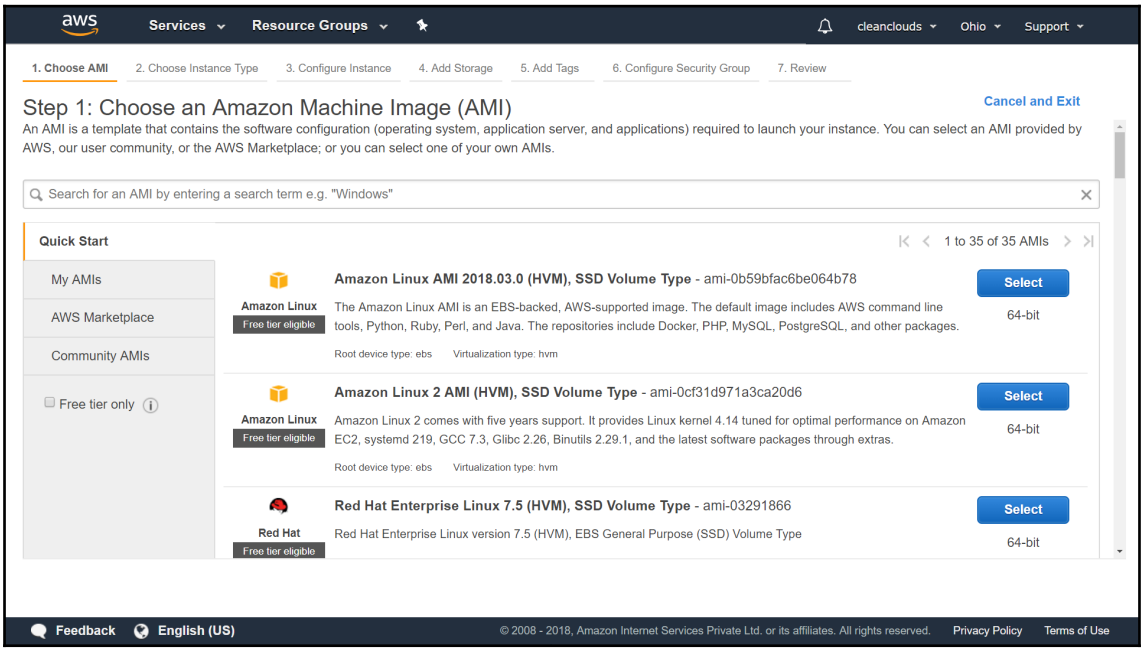
Refer to the below D2 instance details:

Model	vCPU	Mem (GiB)	Storage (GB)
d2.xlarge	4	30.5	3 x 2000 HDD
d2.2xlarge	8	61	6 x 2000 HDD
d2.4xlarge	16	122	12 x 2000 HDD
d2.8xlarge	36	244	24 x 2000 HDD

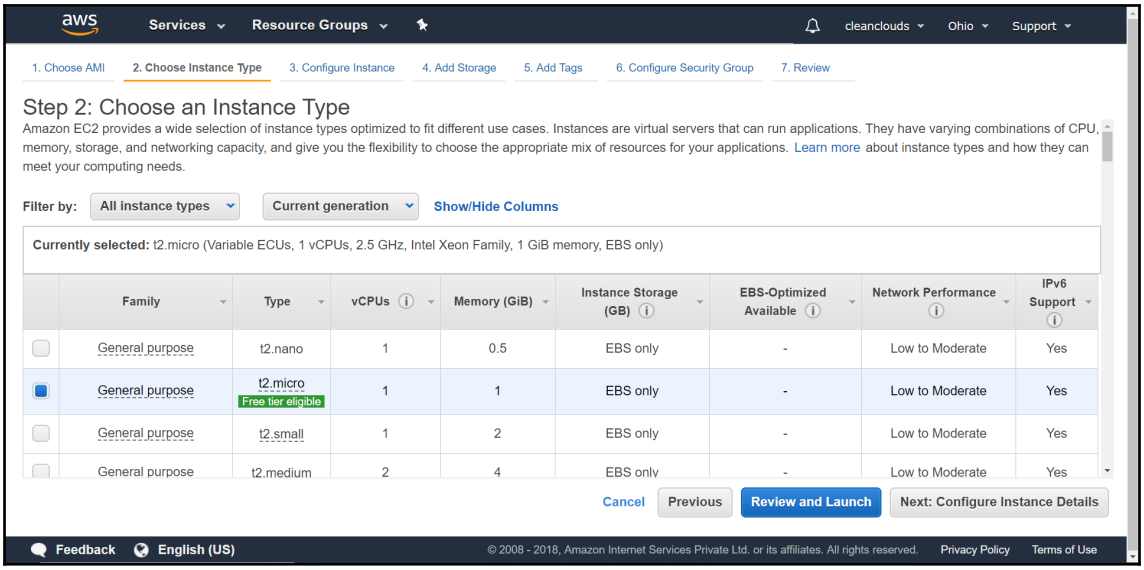
For more details refer to, `https://aws.amazon.com/ec2/instance-types/` .

Launching the instance

Now that we have learned about the instance types, let's continue launching our instance:

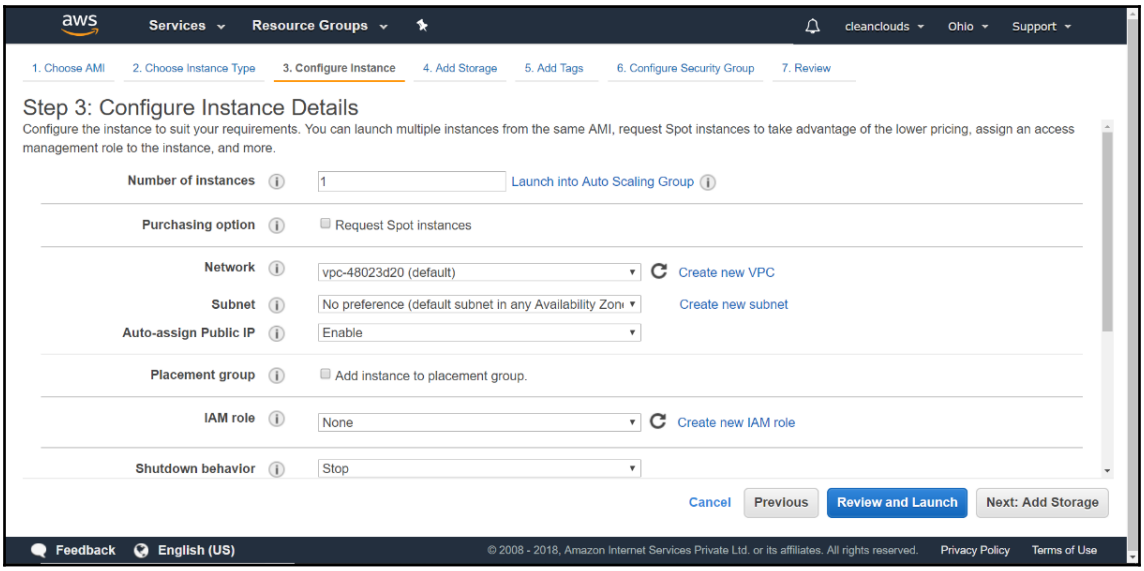


In the second step for launching our instance, we need to select the instance type and size.



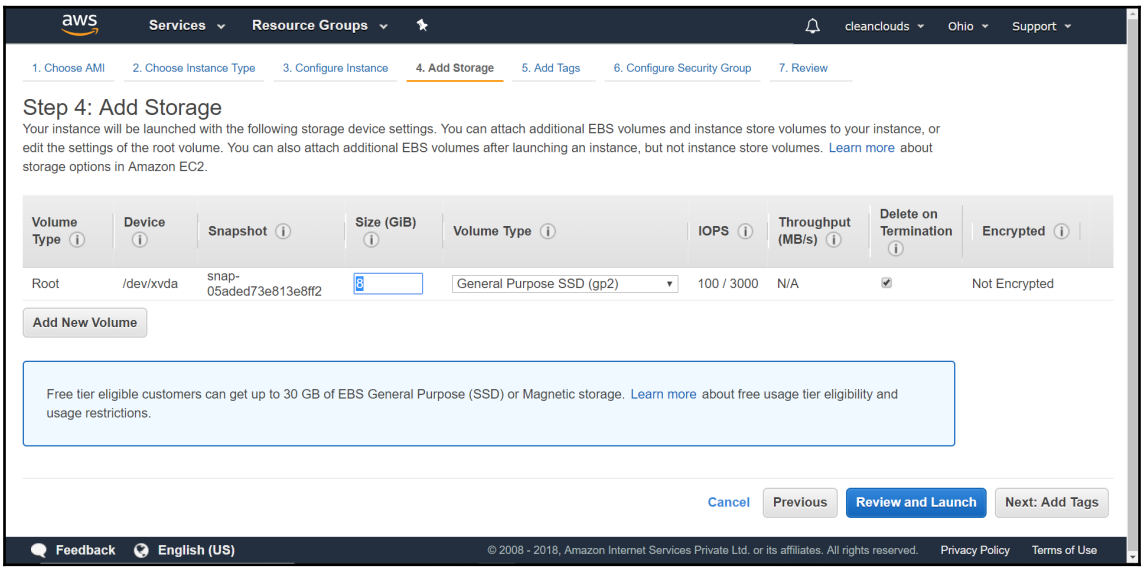
To stay on the free tier, let's select a **t2.micro**. This is a general purpose instance that will provide one virtual CPU and, 1 GB of memory, and will not have any instance storage.

Click on **Next: Configure Instance Details** to go to the next step:



On this page, we're pretty much going to leave everything as the default settings.

However, we do want to make sure that we get a public IP address for our instance so we can log in to it. So, choose **Enable** on **Auto-assign Public IP**, then click **Next: Add Storage**:



The next decision we have to make is the size and type of storage volumes to attach to our instance. So, in the next section we will discuss EC2 storage options.

EC2 storage options

In the previous section, we looked at the available instance types. In this section, we're going to look at the options for filesystem storage for our instance. We will cover the available options, which include local instance storage and network-attached elastic block storage.

Instance storage

Instance storage is SSD, or hard disk storage, that is installed in the host machine. Since it is local, you can get high disk throughput and high input/output per second, known as IOPS. However, one big caveat is the ephemeral, temporary nature of instance storage. When your instance is running, it is occupying a slot in the hypervisor of a host server. So, it has access to its share of the host machine's local storage.

However, when an instance is stopped or terminated, it is removed from the hypervisor slot, and that frees the slot for another instance to occupy it. So, to make sure that no one will be able to access your data, the local storage for your instance is wiped, and any data that you wrote to the volume is gone.

You may be thinking that you could just back up the files before you stop your instance. However, this may not be possible if your instance is impaired and that is the reason you're trying to stop it. So the bottom line is, don't store any important data on these volume, unless the data is also replicated elsewhere. You also don't get to directly provision the amount of instance storage that you want.

You are allocated a certain amount of instance storage, depending upon the size of the instance.

So then, what can you use this instance storage for? Well, they're good for swapping temporary files for sure, or data that is replicated elsewhere. For example, your application can be run from these as long as the source code is stored in a source code repository and your application is stateless.

This means that the state or session data is stored in a separate storage system, such as a NoSQL database. They're also fine for boot volumes, but they are not as convenient to use as EBS volumes. But one thing is that you can't take a snapshot of them, and so making an AMI with instance storage requires an extra step that's known as bundling. This involves copying the files up to S3 first. Also, you cannot stop and start an instance store back instance, because when you do, you completely wipe the boot volume. So, you have to terminate them and then relaunch. Finally, AWS does not offer full volume encryption for instance storage. Because of the limitations and ephemeral nature of instance storage, AWS offers a more durable alternative: **Elastic Block Storage** (**EBS**).

This storage is not located on the host machine; it is on separate hardware elsewhere in the same availability zone.

Elastic block storage

EBS volumes are attached to your instance via the network. Because of this, they have a separate lifecycle and can persist even after you terminate your instance, so you can freely stop and start EBS-backed instances. To further improve their durability, the data is mirrored on two devices. Since they are network-connected to your instance, you should ideally use them with instances that are EBS-optimized, which will give you a more consistent disk throughput. You can provision the size of your EBS volumes however you want, from 1 GB up to 16 TB. You can also attach multiple EBS volumes to a single instance, and also RAID them if you like. However, like a physical disk drive, they can only be attached to one instance at a time.

So, if you need a shareable filesystem like an `AZ`, then you should consider AWS's elastic filesystem service. One really nice feature is the ability to take snapshots of EBS volumes. These are very durably stored in the S3 storage service. You can restore or create new EBS volumes from snapshots. You can also enable full volume encryption, which will encrypt your data at rest, and also encrypt the snapshots.

General purpose SSD

The most common type of EBS volume is general purpose SSD. Because of the price drops in solid state storage, they are fairly cost effective and deliver good performance. You may remember from the previous section how T instances accumulate credits and burst when they need CPU capacity. Well, general purpose EBS volumes do a similar thing with IOPS.

They are also good for storage for databases, and particularly ones that are in development or test environments. However, if you want consistent IOPS for EBS volumes, then you can upgrade to provisioned IOPS SSD.

Provisioned IOPS SSD

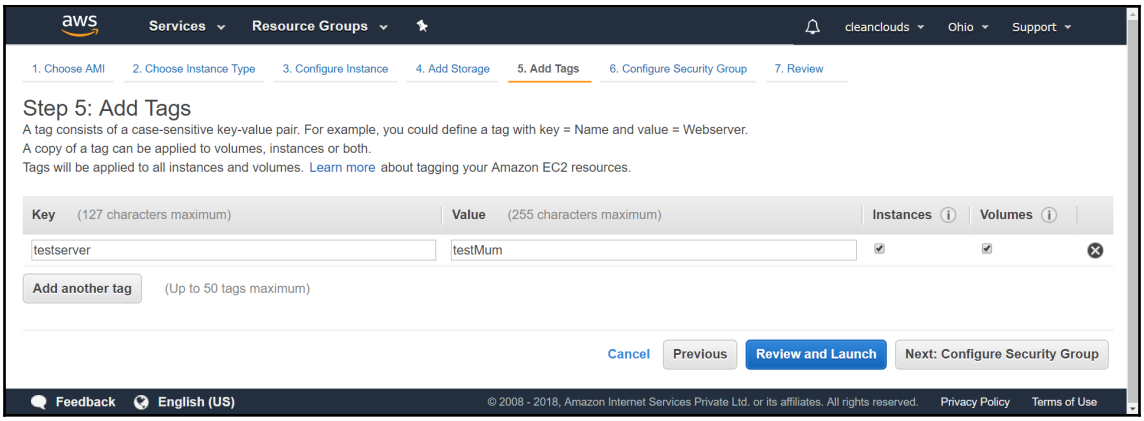
You can provision a consistent amount of IOPS from these volumes without bursting.

There's an extra cost for the amount of IOPS you provision, so it only makes sense to use these in production environments or for things like databases, which really benefit from the high and consistent I/O.

Throughput optimized HDD

Even though SSD is the new standard, AWS offers hard disk EBS volumes for a cheaper price than the SSD ones. These are optimized for sequential I/O and cannot be used for boot volumes. They offer a burstable throughput, similar to the general purpose SSD.

Notice that there is already a default route volume here of 8 GB general purpose SSD. We could, of course, switch the **Volume Type** to **Provisioned IOPS SSD (io1)**, or use the cheaper **Magnetic (standard)** storage. However, we get up to 30 GB of EBS general purpose SSD storage on the free tier, so we should definitely use that. We can also **Add New Volume** at this time if we would like, or we can add one later on. So, let's just leave it as the default and click **Next: Add Tags**:



What is the use of tagging? Tagging instances are an optional step, and you can use them to give a name for this particular instance. So, let's put `Packt Training Instance` as our **Value**, and then click the **Next: Configure Security Group** button. So, now we are ready to learn about security groups in our next section.

Security groups

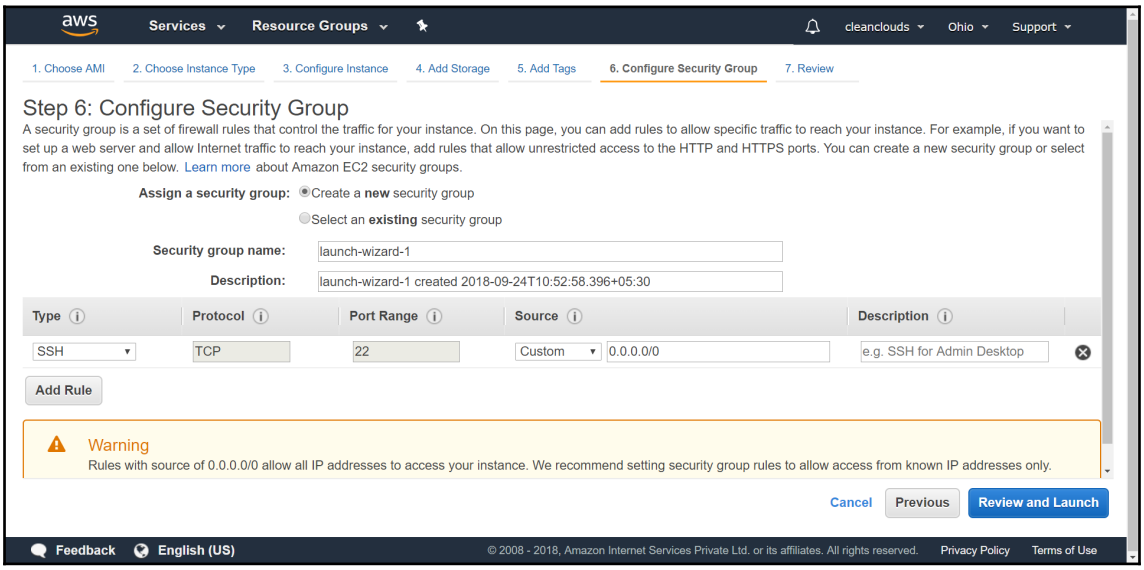
In the previous section, we discussed the available storage options for EC2 instances. The next step to launching our instance is to create a security group. In this section, we're going to show you how to create a security group, add rules, and complete the launch of our instance.

Now we've reached step 6 of launching our instance: configuring the security group.

Security groups allow us to protect our instance with firewall rules. We can allow traffic into our instance by protocol and source. Note that we have to have a security group associated with our instance. A security group could be associated with more than one instance.

So, for example, we can create a single security group for all of our web server instances if we would like. All traffic is implicitly denied by default. So, if you don't specify a rule for a particular protocol, then that traffic is going to be blocked. On the screen, we are adding rules for the inbound traffic. We don't need to worry about outbound, because security groups are stateful, which means that responses to requests will always be allowed. You can edit your security groups later and specify outbound rules as well if you like. However, you usually don't need to do this because there is already a default rule to allow all traffic outbound. Another thing I should mention is that when you modify your security group rules, the changes take place instantly.

There are two options shown on the following screen:

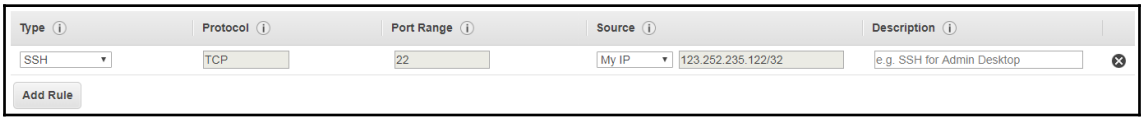


We can either **Create a new security group**, which already has a default rule for SSH traffic, or **Select an existing group** that we created previously. Well, let's go and **Create a new security group**.

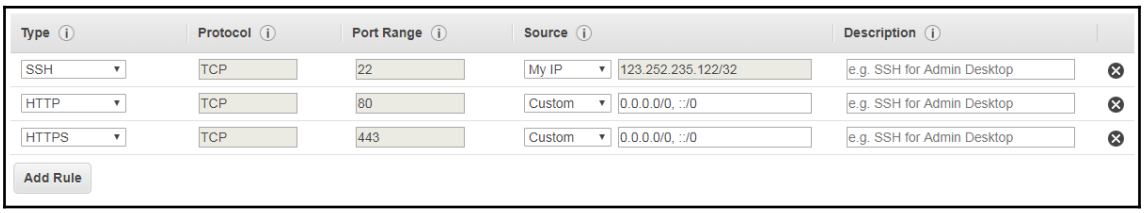
Let's start by giving our **Security group a name**. So, let's call it a `WebServerSG`, and put in a description: `SG for Web Servers`:

In order to SSH into our instance, we have to have the SSH protocol allowed. So, there's already a rule in place for that. Now the **Source** here though is listed as **Anywhere**, or it could be **Custom** or **My IP**. You should lock this down. By saying **Anywhere**, it puts an IP address range of `0.0.0.0/0`, which means the entire internet. Since that's not very secure, there's a warning right at the bottom of the page that says, `Rules with source of 0.0.0.0 allow all IP addresses to access your instance`.

So, instead, you should really put something like **My IP**, or if you know the range of the office where you work, put that in:

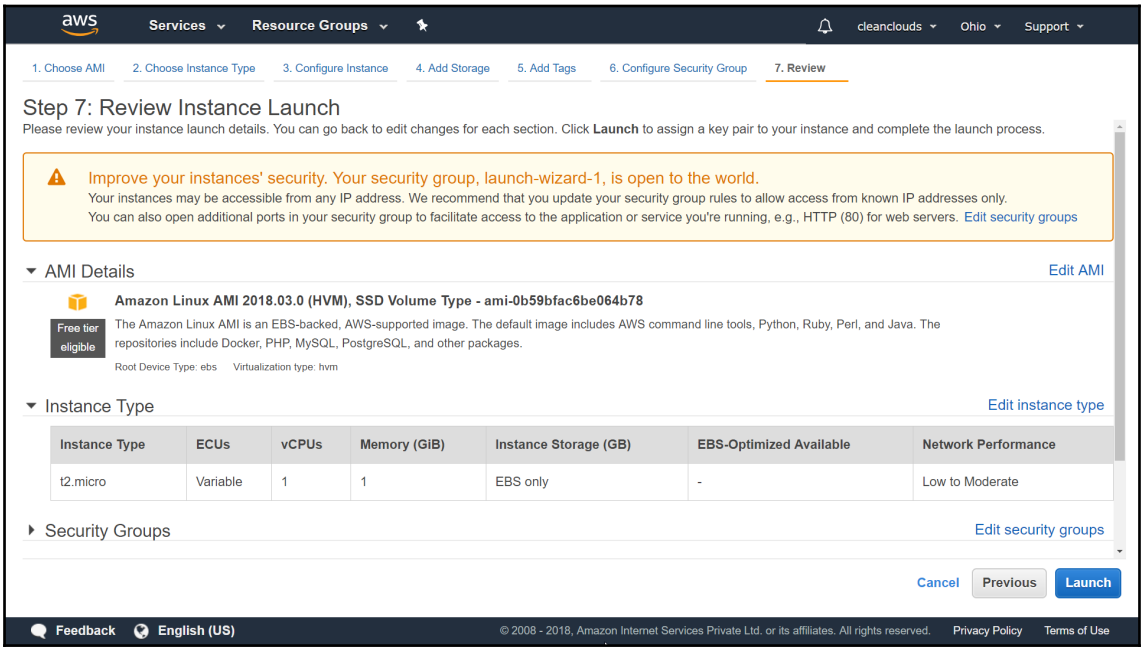


So, that locks it down a lot better. Now since this is going to be for a web server, we need to allow **HTTP** traffic:



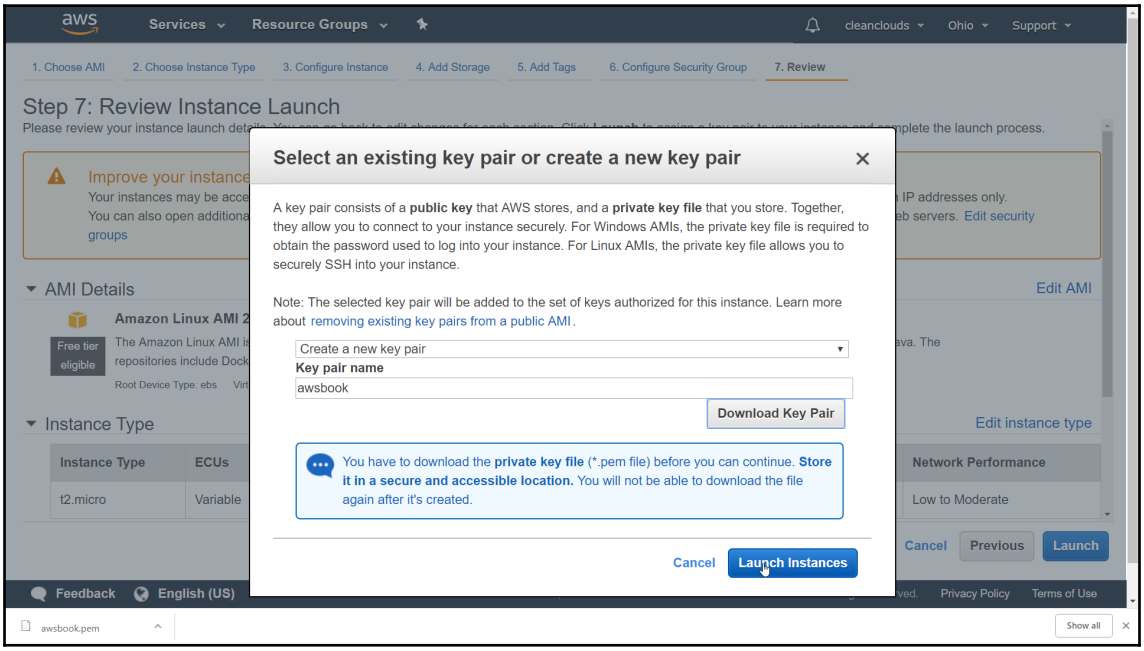
Select the type of traffic you want to allow, and you then can select **HTTP**. That will fill in port `80`. This we do want to allow from anywhere on the internet, and let's do the same thing for `HTTPS`. While we are doing this, notice all the protocols that are there by default. Plus, you can specify your own custom TCP rules, UDP, and ICMP rules, and specify the port numbers that you want to allow. So, now we've got **SSH** from **My IP** mentioned, and **HTTP** and **HTTPS** from the internet. That should do it.

So, let's click **Review and Launch**:



This page allows us to review the settings for our instance. You can see the **AMI Details**, the **Instance Type**, the **Security Groups**, and the **Storage** and **Tags** all on the same page.

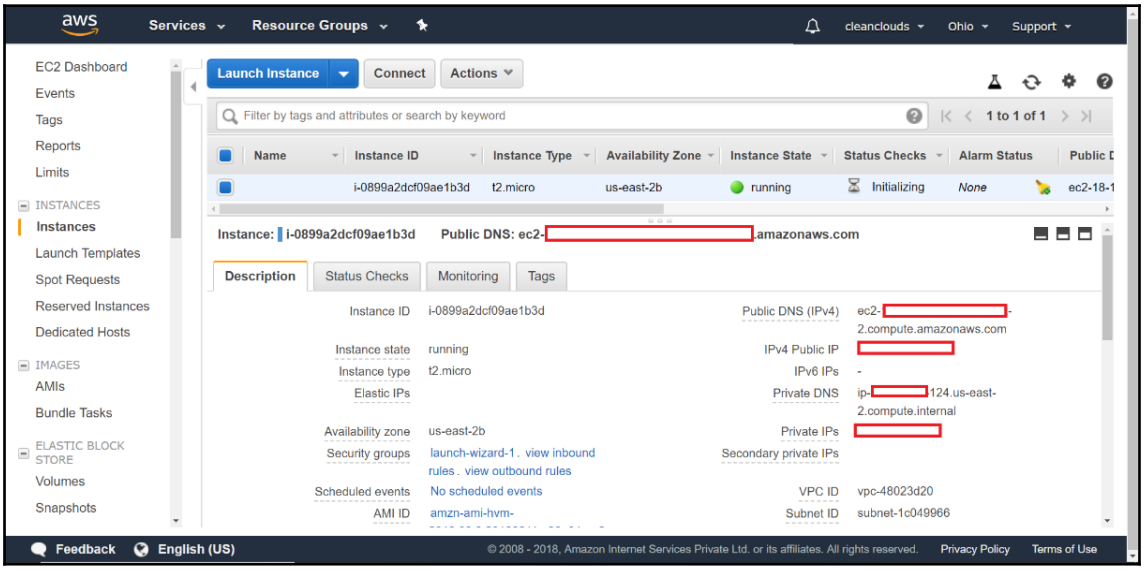
If you're happy with what we got, then click the **Launch** button. Now, you should see a popup, asking us to choose a **key pair**:



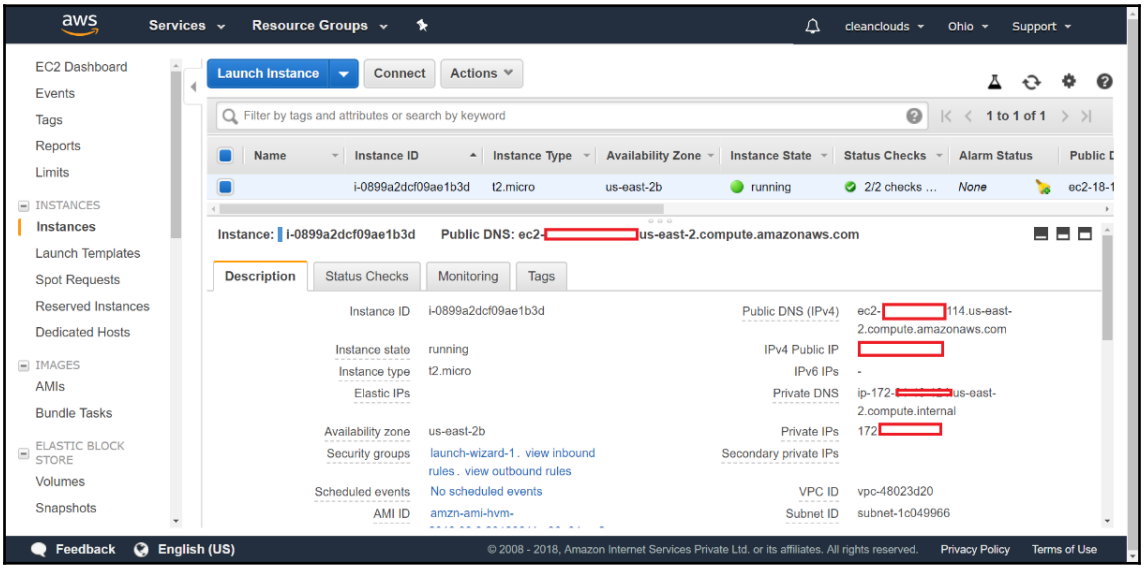
Key pairs are credentials that allow us to log in to our instance. We don't have any created, so we're going to have to create a new one. So, choose **Create a new key pair**.

Give it a name, `awsbook` , and then click **Download Key Pair**. This will save a file to our `Downloads` folder on our desktop. To complete the launch of our instance, just click **Launch Instances**.

Click on the **View Instances** button down at the bottom, and you'll be able to see the instance being created:



Notice the **Instance State** will be `pending` initially, and then, after about a minute, this will switch to `running`:



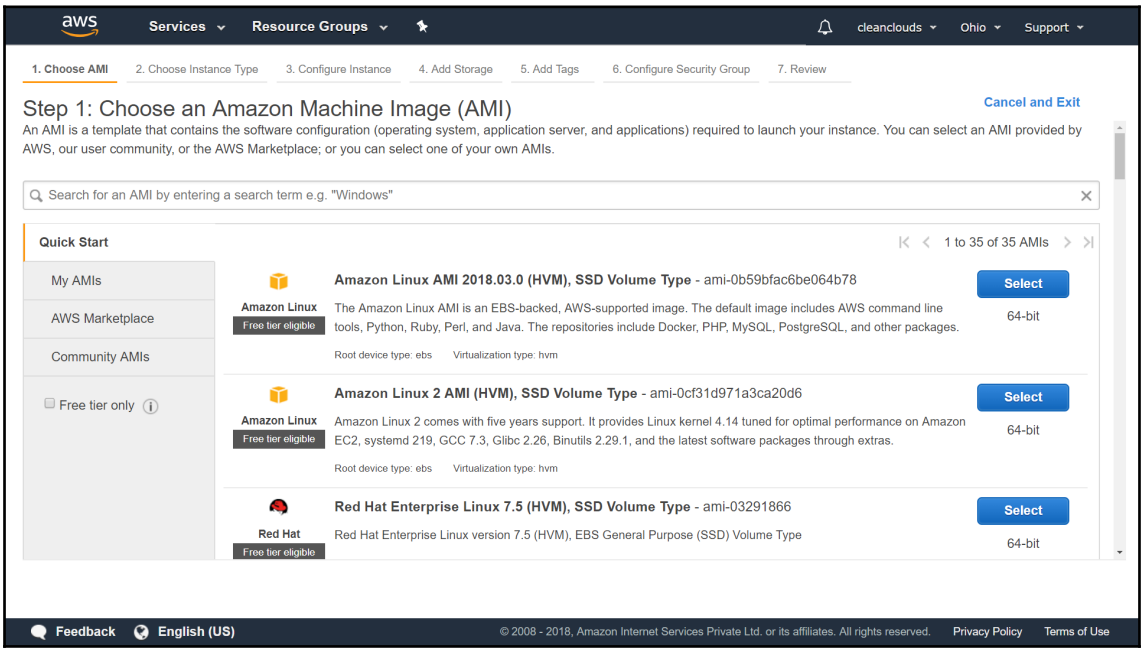
AMIs

An AMI is an image of the filesystem for your boot volume. A base AMI contains an operating system. However, you can launch AMIs that have applications already installed and pre-configured.

These are available from the AWS marketplace or the AWS community, or you can launch custom AMIs that you configure for yourself.

Quick start

When you click on **Quick Start**, you will get the following screen:

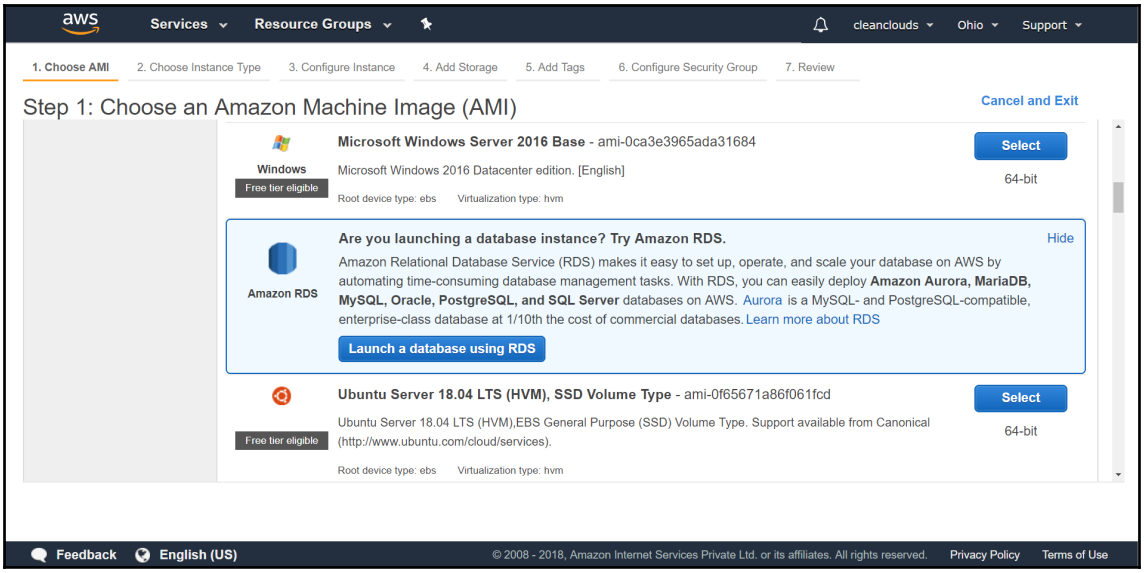


You can select the **Free tier only** checkbox to filter the available AMIs.

These include Linux varieties such as Red Hat, Ubuntu, SUSE, and Amazon's own Linux variety, which is optimized to the AWS environment and most closely resembles Red Hat. You can also select the Windows Server operating system, some with SQL Server already installed.

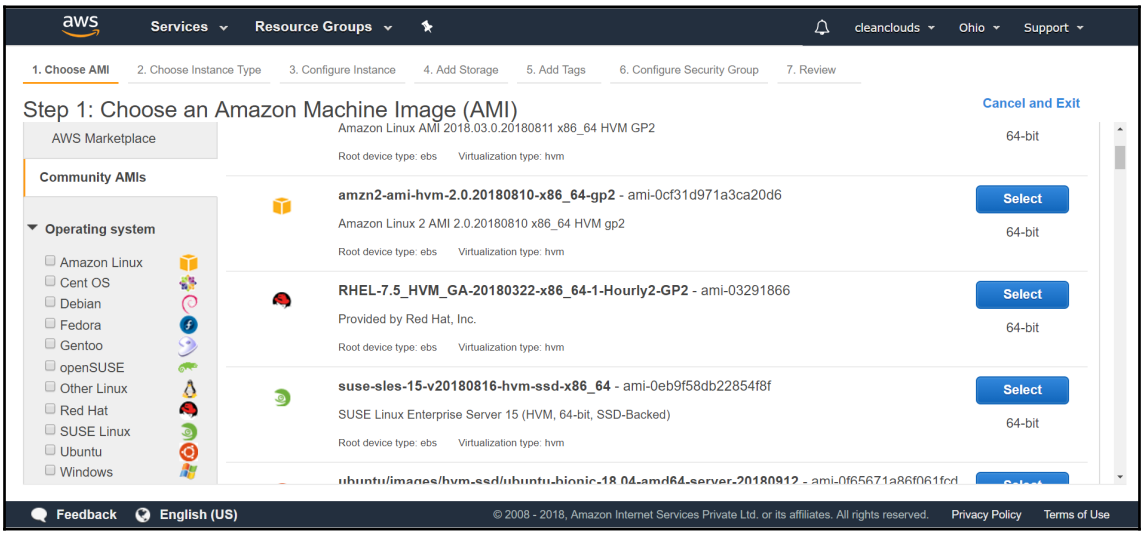
If you choose one of these, or Linux varieties such as Red Hat which have a software license fee, the licensing will be added to the hourly charge for the instance.

When selecting an AMI, some of the features to note are the architecture, 32- or 64-bit, the root volume type, EBS or instance store, and the virtualization type, HVM or para virtual:

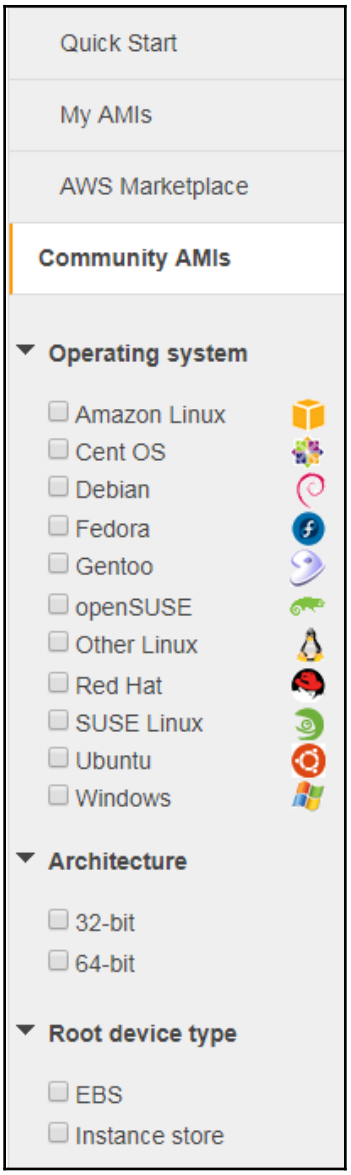


Community AMIs

To get an idea of all the options, click on the **Community AMIs** tab:



To get the maximum access to the instance memory, you should use the 64-bit AMIs, unless you have some software library that doesn't run well on 64-bit architectures:



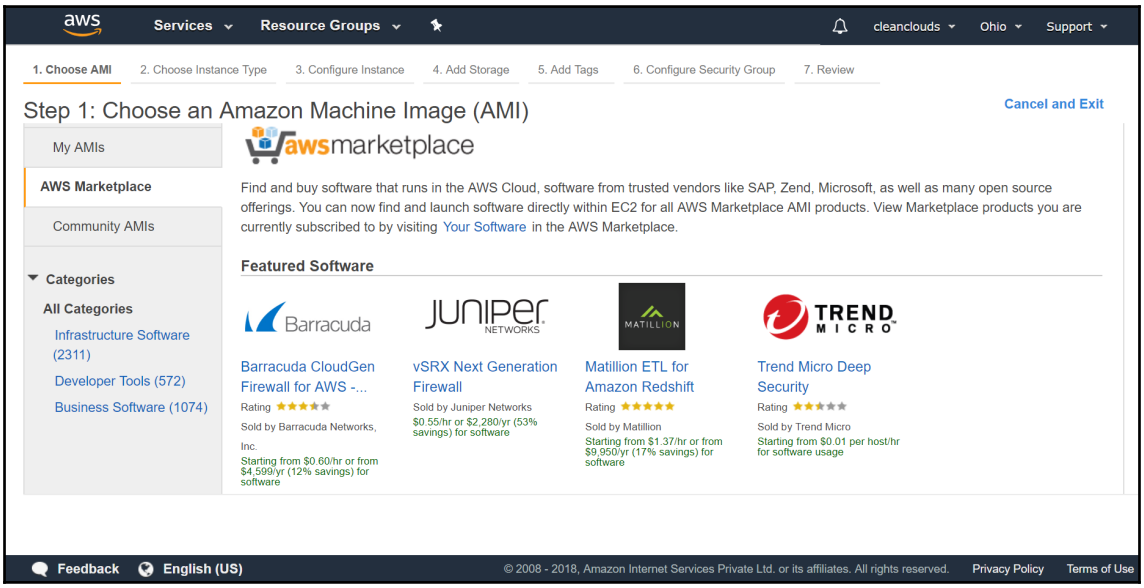
The choice of storage for the root volume depends upon the required durability of the data you put on it. If you don't plan on writing any critical data to the boot volume, then you could use **Instance store**, which is much faster than **EBS** volumes.

However, if you stop your instance for any reason, you will need to relaunch it from the AMI, because instance storage is wiped when the instance stops, so you are effectively terminating it. EBS storage persists with a separate lifecycle from the instance, so you can freely stop and start EBS-backed instances, and all of your data is retained. The ability to stop and start your instance is a really nice feature, because an instance that is impaired can usually be fixed by just stopping and starting it. So, my recommendation is to generally choose **EBS** for your root device type for most purposes, unless you need a very fast disk I/O to your root volume.

In the early days of the cloud, **paravirtual virtualization**, or **PV**, was believed to perform faster than a **hardware virtual machine** (**HVM**). However, this is no longer the case, and PV virtualization cannot make use of hardware extensions, such as enhanced networking. So, choose HVM whenever possible. Be cautious before you choose an **AMI from the community AMIs** tab. For safety, choose one that says **Provided by Amazon**, or another source that you trust.

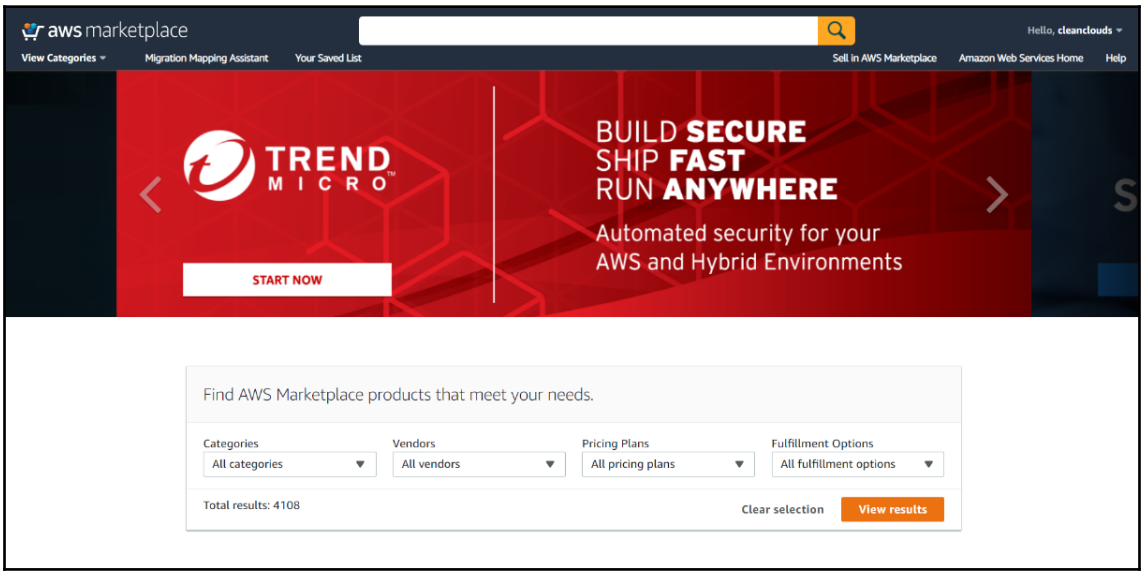
AWS marketplace

Third-party software vendors make their software available as pre-built AMIs. You can find these under the **AWS Marketplace** tab:



There are thousands to choose from, from well-known and respected software companies such as SAP, Barracuda, and Oracle.

For the complete selection, visit the marketplace at `https://aws.amazon.com/marketplace`:



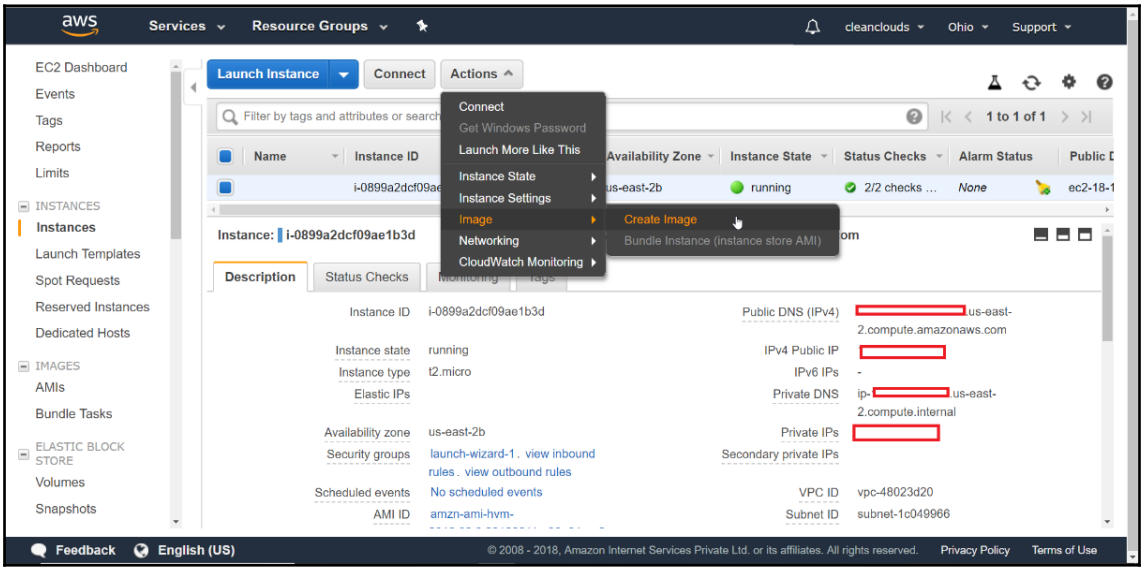
Some of these add the software license to the hourly rate, while others use a bring-your-own license model, in which you need to purchase the license separately.

My AMIs

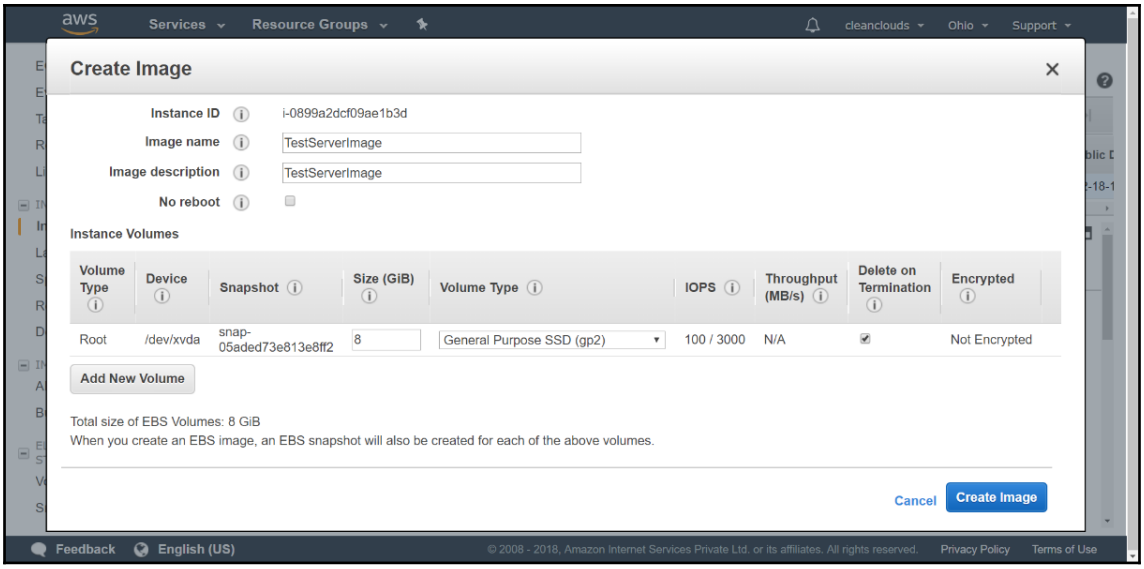
Finally, you can launch any AMI that you create yourself. These are listed under the **My AMIs** tab.

Creating an AMI is a very simple process. It involves launching an EC2 instance from a base AMI; logging in to the operating system; and then adding your own configuration, downloading and installing software, and creating an image from the running instance:

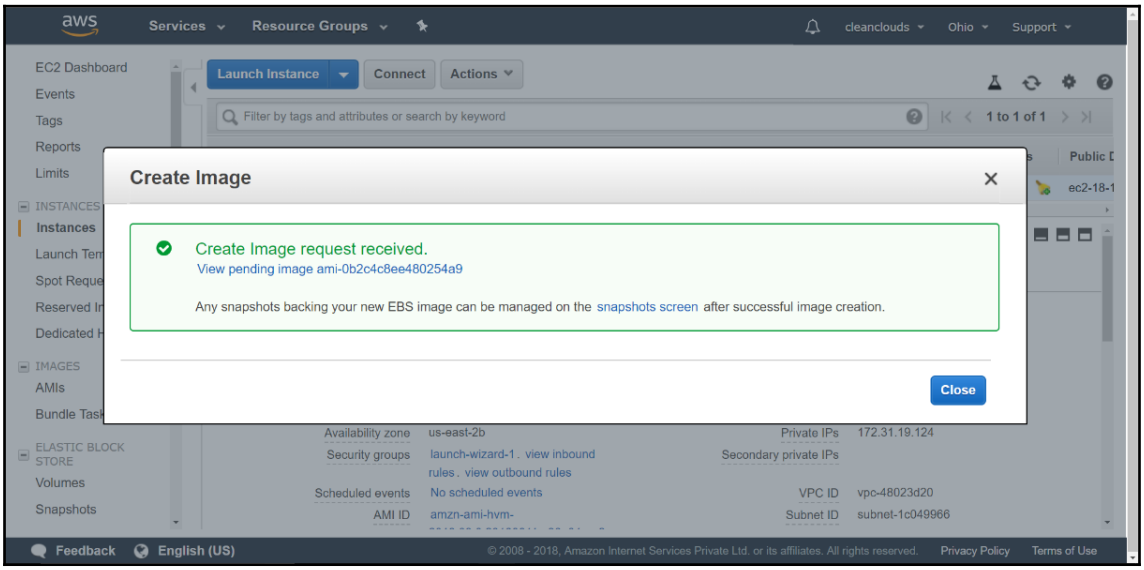
1. Click on **Actions** | **Image** | **Create Image**:



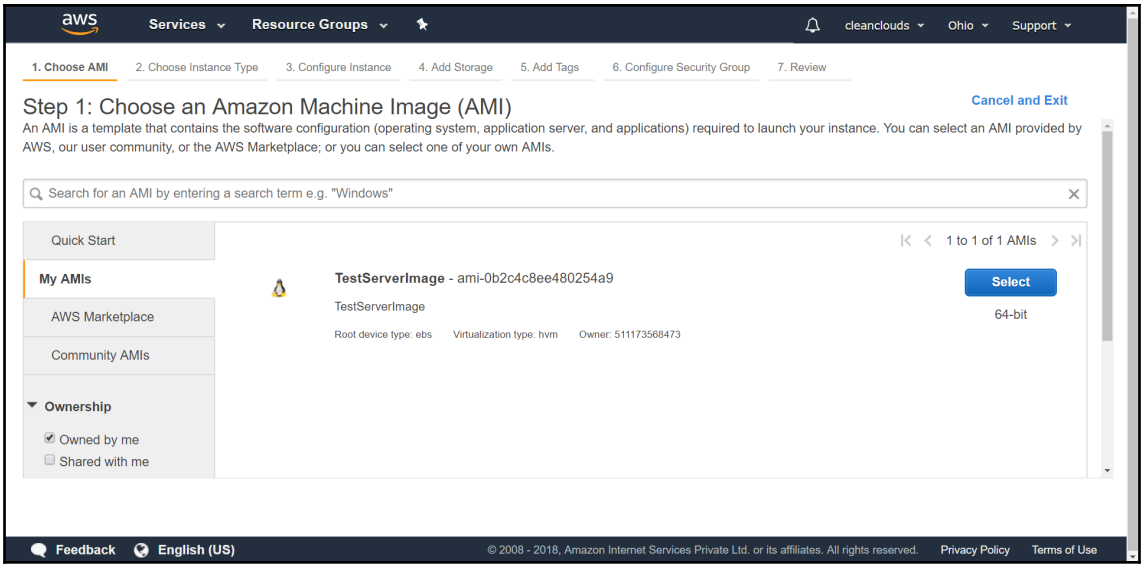
2. Review **Instance Volume**:



3. Click **Close**:



4. Go to the **My AMI** section and verify our newly created image:



Launch an instance from your image and verify the previously installed settings.

Summary

In this chapter, we learned about AMIs and where to locate them. We learned about the five general categories of instance types, their features, and suitable workloads for each. We discussed the storage options for EC2 instances, and showed how to attach these volumes when launching an instance.

In Chapter 3, *Logging in to EC2 Instances*, we're going to learn more about the key pairs and logging in to EC2 instances, and the difference between Linux instances and Windows.

3
Logging in to EC2 Instances

In `Chapter 2`, *Launching an EC2 Instance*, we covered the process of launching EC2 instances. In this chapter, we will cover how to connect to them.

The topics that we will cover are as follows:

- Key pairs
- Logging in to Linux instances
- Adding Linux users
- Logging in to Windows instances

Key pairs consist of a public key and a private key, and are used to securely connect to your instances. The process is different for Linux and Windows instances. We will discuss ways to generate the keys, and then use them to log in to your instances.

Key pairs

We will first cover key pairs. You must have a key pair before you launch an instance. We're going to generate a key pair and associate it with an instance.

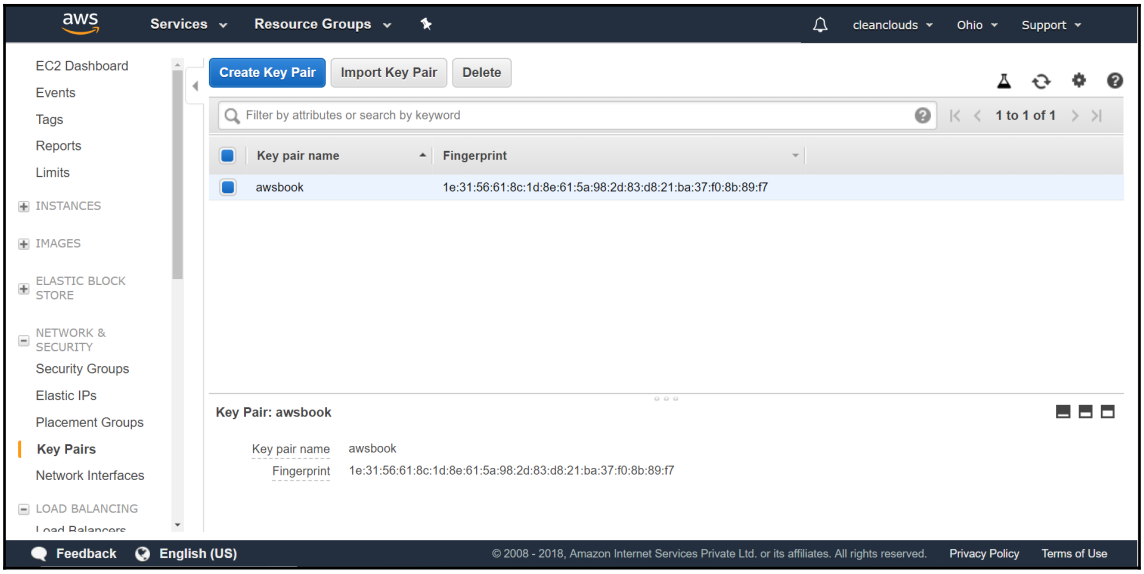
AWS uses RSA asymmetric public-key cryptography to secure the login information for your instance. Each pair consists of a public key, used to encrypt data, and a private key, used to decrypt data.

When you launch a Windows instance, a random administrator password is automatically created and then encrypted with the public key. You must present the private part of the key to decrypt the password. Then, use a remote desktop client to connect to the instance, and log in as administrator using the password. For Linux instances, AWS will create a Linux user named `EC2-user`, or on Ubuntu, a user called `Ubuntu`. The public key is copied automatically into a file called `authorized_keys`, in the `SSH` directory, in the user's home: `~/.ssh/authorized_keys` .When the user attempts to log in, the server uses the public key to encrypt a challenge message, and sends it to the user's SSH client.

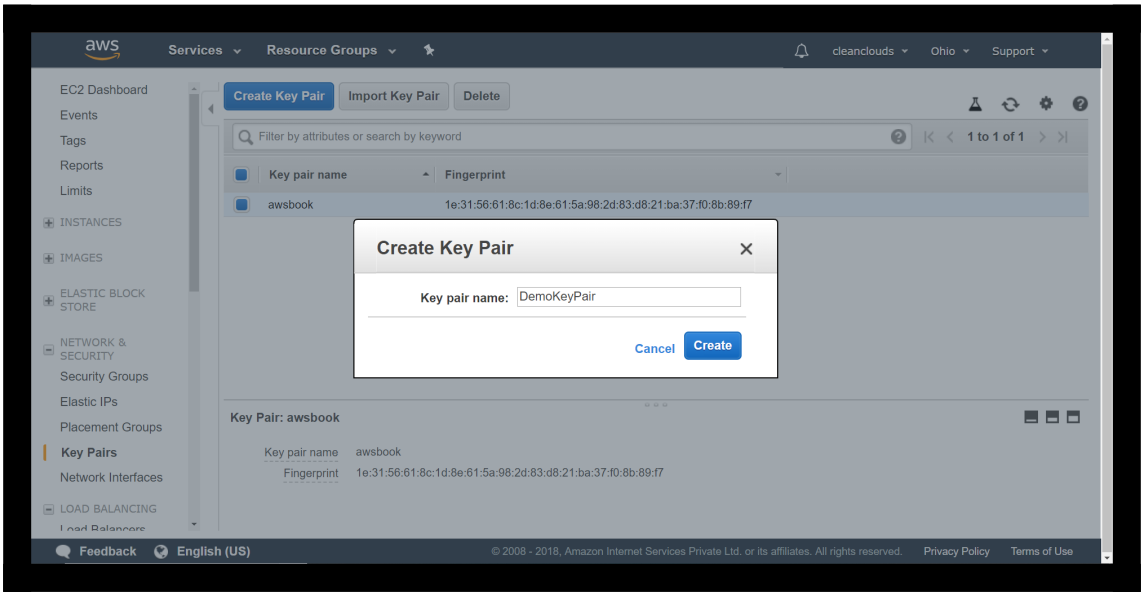
If the client is able to decrypt this message with the private key, then the user is authenticated and can log in. It's important to note that AWS never stores the private part of the key pair. This must be safely stored on the client machine.

Key pairs are very easy to create in the management console. Just go to **EC2**.

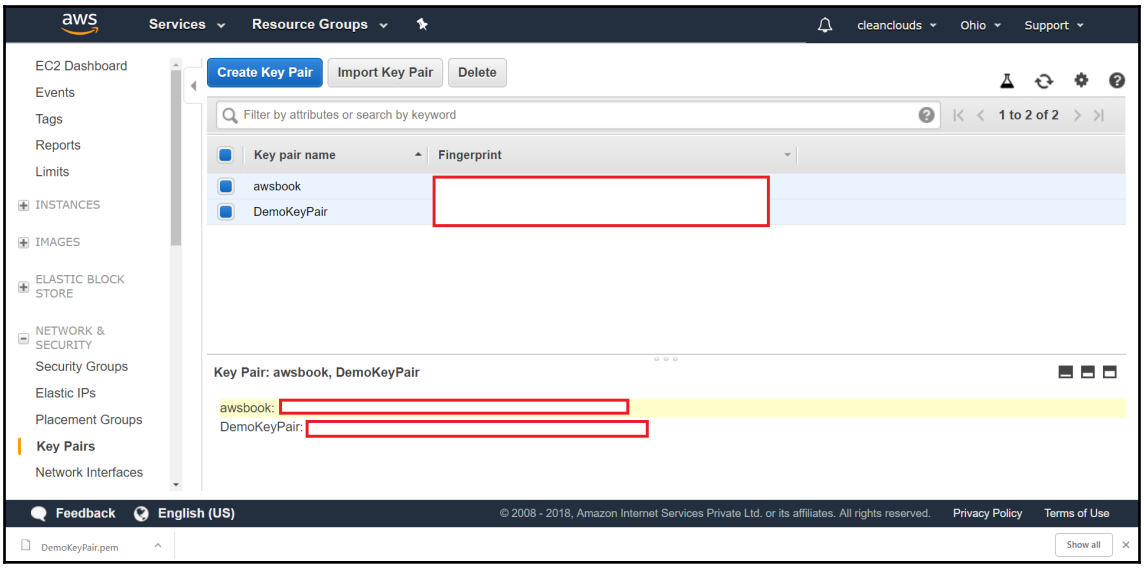
Then, in the left navigation section, go down to **Key Pairs**:



Click the **Create Key Pair** button, and give it a name:



Click on **Create** and it will download the key:

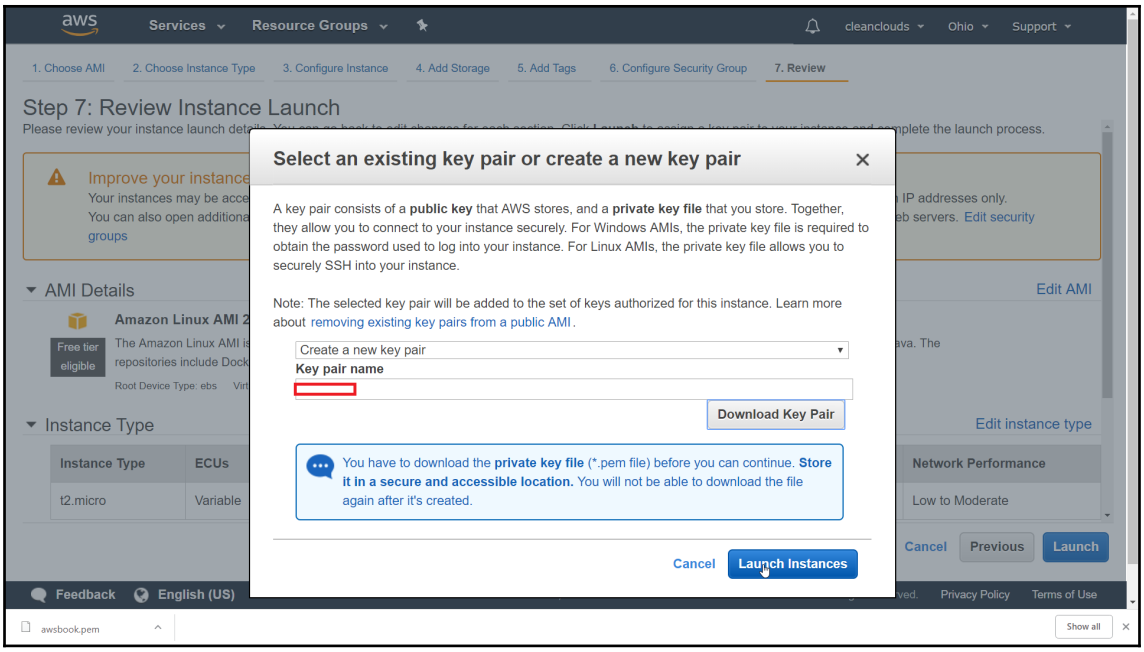


Here, we have added `DemoKeyPair`:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAjjlayH7kkjHpc2qxCpT9lAJ45ROrxdxCTLzHE84LGIjP0QXNkmEACbLxiJLS
HfTBksww9rQLY6lWsqefe77rJJMhl+bGPsRZwIVjvD7/2CU9dDcgRzvyAo2k5YR3dxHA9e95LGiA
PL2Xf7BZSkbsgW0719solQTCMRpm5HkBUB0mCVpCh/ytFBm2DfwRqPD4Hyjew+zDZ2/yn2jjdHMT
kS7l9rfnudglgHN/dBCtWS7Ac0JIYqEevGxOkoTxs8OjY+8ZcEZntXpho+XQFnv2HFxXpRxlAzEr
v+prOiA6vSYFnZ2BzJyCl1l2vTiB1i1XC2jqGVxuoxetjM+dk5MJxwIDAQABAoIBABzdbRxmDJPT
R3MiQz5HkNOaUNkFTgaBNn+GjMzrM3tOQQdGKzu7LhvJ0pWKXtSvmR3nDUT0s58OThtCQ+8Eh4rL
xbSjpLaWeTTGncBV9Oz0We2dRpLOisaT4tSmNuzWkJgZezlVGDm7rOTAcM7PaC64cb3Uf/8GkTcQ
CbPzDx4oDtzF/YlQRyBJlhE/Cjp+iQilZkVC+MBfxTx5QBE7Tn4Yt746YsMV7D0mKV6Iw7AHqOgN
K7c7lDdLudJ0nc2rLWrEhPnrvXYL0NiAvle3lm95SJU1LXSDSz3MJypuyvToSBkzhGA31iQ8AKRH
HFIc2ltYe0twR/F1dd64HBCJXVECgYEAv/HM38SlbLSkDq6t+2VVr82Yi2mXd6hGICnOyNlpo7zK
GqVviKqi2uGb4g9zHl5dWzSAzFAA3i5eaPjTKRranLdFm3z2V+0WvDQZiTUqb2+DvEKqTYmvx2W3
apEsWjDhQKqJ6DTEk8XpgMkNQHVrNHGCtY+0UsNfrEHyZjywFfkCgYEAva/VtXXMr6pfiFSj6wGs
CUNVBzg0RLIfkjWUAvYfyZuN45HBruyNHNFqMAAz940HOsrjq9XR8D2uaW/ESzz9D2nbrEm3bZlK
ZY0FtkB0w4gvxikTgWuYTN6gxF485DrUlKBSwaaLOpspNYUorx5B+1WJxTdsfp1FgBEa8cT9Db8C
gYEAiORWjTCeGk0qxB3oj2bSGbOzm5oPIJOuj7nBdJU6NkCqkx5x0TKBO0Gz/yGtpHcAdc+YHB7i
v5KqWos7bhFs4GAFEMOqnULF+CvgGm8EeL07YsTY2LvdlYPBsAshaqF5xLjb6fPYTtvTkLnS7anQ
b3KVGhAHNw+H6D6QNkIKA2ECgYAJjGdimgpysU3by/7wXoYtdTE7is2+w4SpVH8d4wvYlzb+RKEO
UCQOVlBVGD2FcZjykbICPWIzCLfV4OVtsCVA+GAxfQh5wRg0onOuJ/J4wUs+W2T699KTlAUdvWyt
RHTgOy3PR7XsflEFu60/m3NQ1o532gFhIkOUXcgQcY0XewKBgQCa5+I9rqheo3HXNrYY5O93n613
d9k7BgPTmX3yYRifIgVImWnjgXTrscpoudHjOBKNANegjhtDqi7E5cLXFxxEiKHXvg1MtNcCwcNH
zQr+d/Oi1UtXvzJg3pJtz+L8Ob0giEgxrAth3fNzGPttJxMZxOy7mvNjUpScMqM9hYjJqQ==
-----END RSA PRIVATE KEY-----
```

The private part of the key pair will be automatically downloaded to your desktop, and this is what it looks like. The public part of the key is saved by Amazon.

You'll recall from the previous `Chapter 2`, *Launching an EC2 Instance*, that the last step when launching an instance is choosing a key pair:



As you can see, we have a popup that allows us to choose an existing key pair, and we can choose from the ones that we've already created. You have to acknowledge that you have access to this key by checking the box shown in the previous screenshot. Now, you can launch your instance by clicking the **Launch Instances** button.

Note that you can use this same key to launch multiple instances, or even all of your instances if you like.

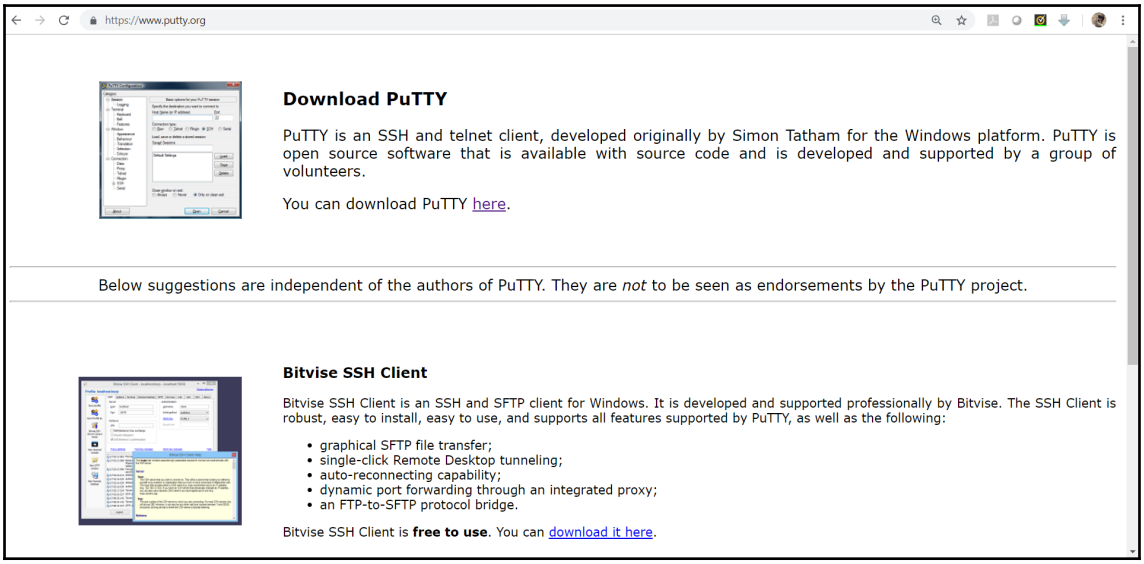
In the next section, we will cover how to log in to Linux EC2 instances.

Logging in to Linux instances

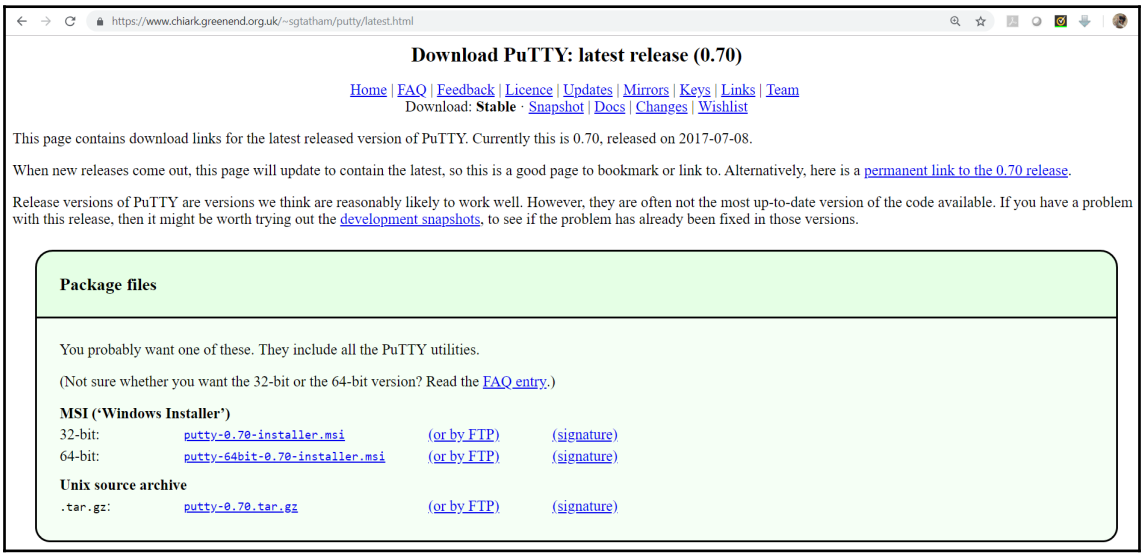
In the previous section, we looked at EC2 key pairs, and associating them with instances. In this section, we're going to use a key pair to log in to a Linux instance from Windows. Then, we will add more Linux users and generate key pairs for them to use.

To connect and log in to Linux instances, we use the SSH2 protocol. We don't use a password, because we provide the private key for authentication. If you are connecting from a Windows laptop or desktop, you will need an SSH client. There have been rumors that Microsoft is planning on adding an SSH client to PowerShell, but that hasn't happened yet. There are many good free SSH clients you can use.

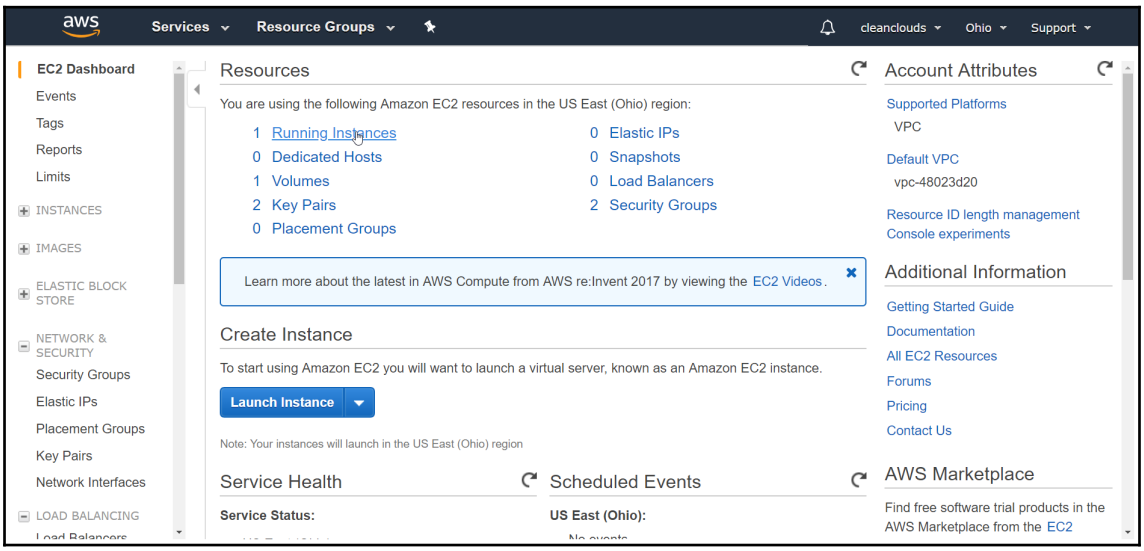
For this example, we're going to use PuTTY. You can download PuTTY by going to `https://putty.org/` and then clicking the link on the download page:



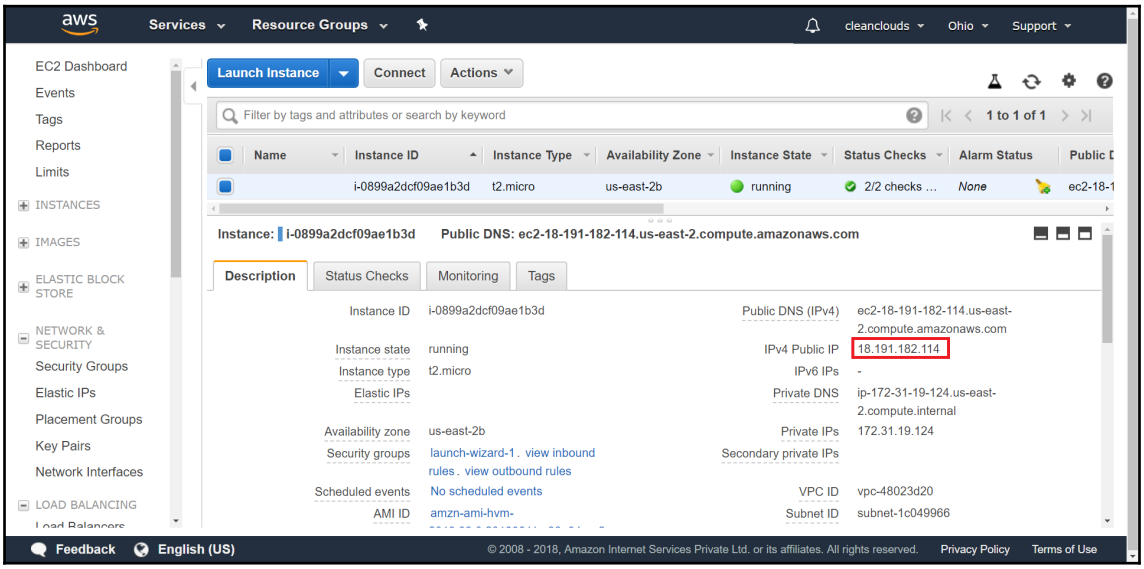
You're going to need `putty.exe` to SSH, but if you're going to be working with Linux instances from Windows, you'll probably also need `puttygen.exe`, `pageant.exe`, and `psftp.exe`. So, you should probably just download the ZIP file containing all of the binaries:



First, you will need the public IP address of the instance. So from the dashboard, click on **EC2** and then **Running Instances**:

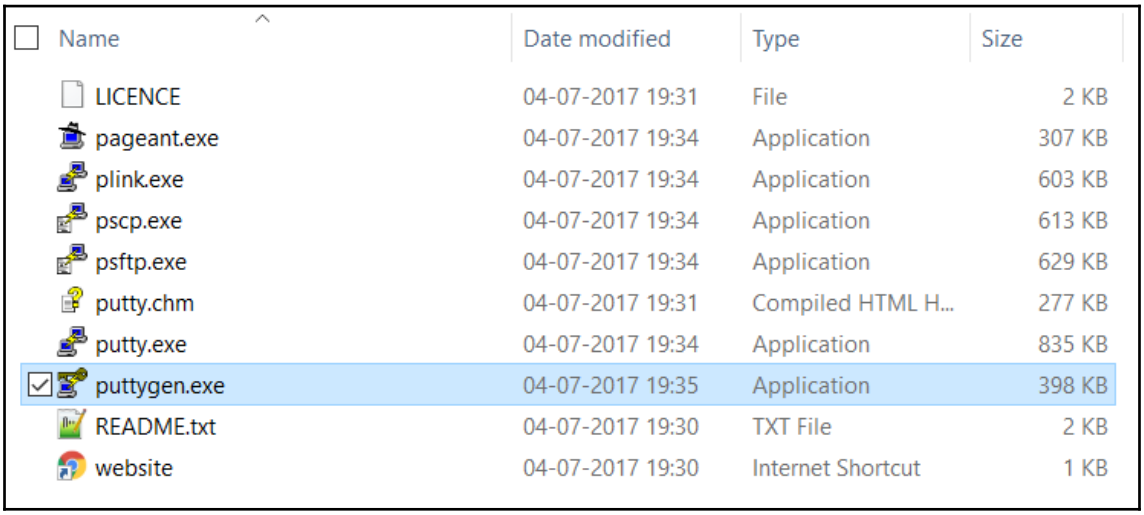


Then, make sure your instance is selected, and then go down to the bottom and copy **IPv4 Public IP**, under the **Description** tab:



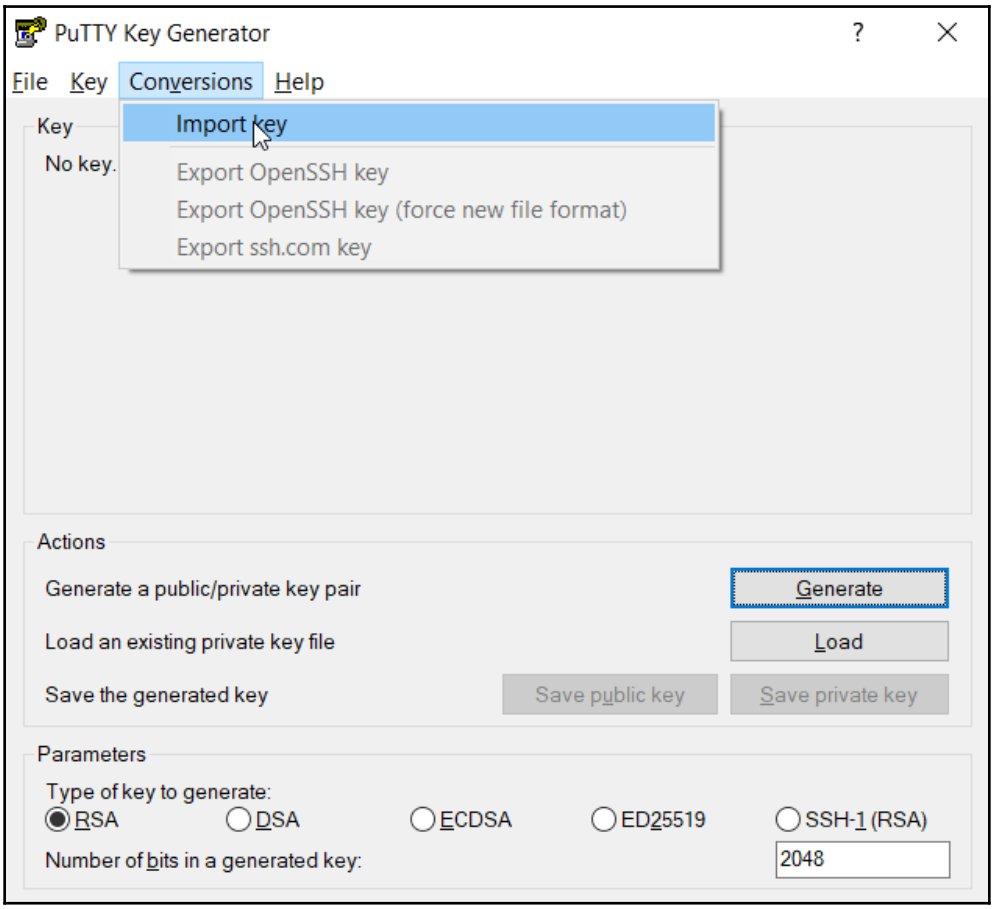
Copy that into the clipboard, and now you have the public IP address for the instance. The last thing we need to do before connecting is to convert our private key file into a format for PuTTY to use.

So, go to your `putty` folder and launch `PUTTYGEN`:



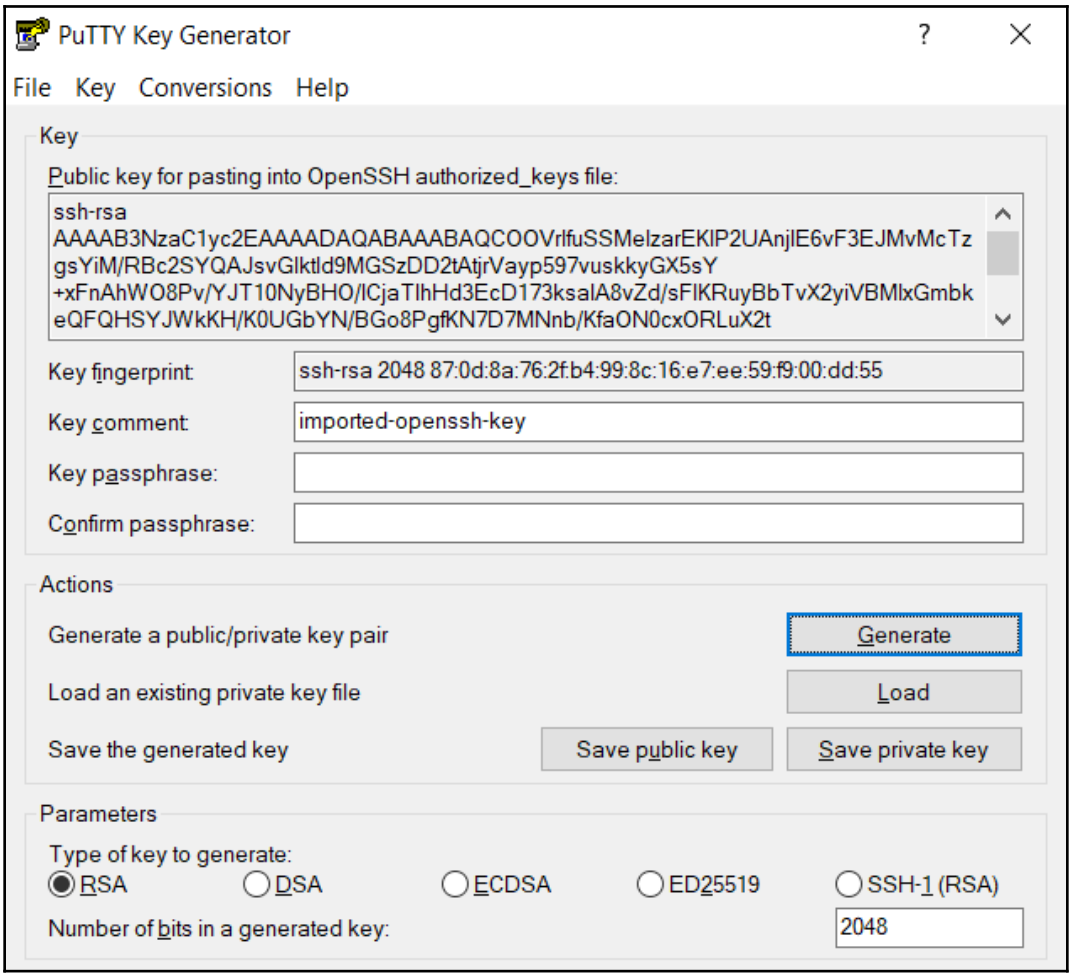
Name	Date modified	Type	Size
LICENCE	04-07-2017 19:31	File	2 KB
pageant.exe	04-07-2017 19:34	Application	307 KB
plink.exe	04-07-2017 19:34	Application	603 KB
pscp.exe	04-07-2017 19:34	Application	613 KB
psftp.exe	04-07-2017 19:34	Application	629 KB
putty.chm	04-07-2017 19:31	Compiled HTML H...	277 KB
putty.exe	04-07-2017 19:34	Application	835 KB
puttygen.exe	04-07-2017 19:35	Application	398 KB
README.txt	04-07-2017 19:30	TXT File	2 KB
website	04-07-2017 19:30	Internet Shortcut	1 KB

Go to the **Conversions** menu and choose **Import key**:



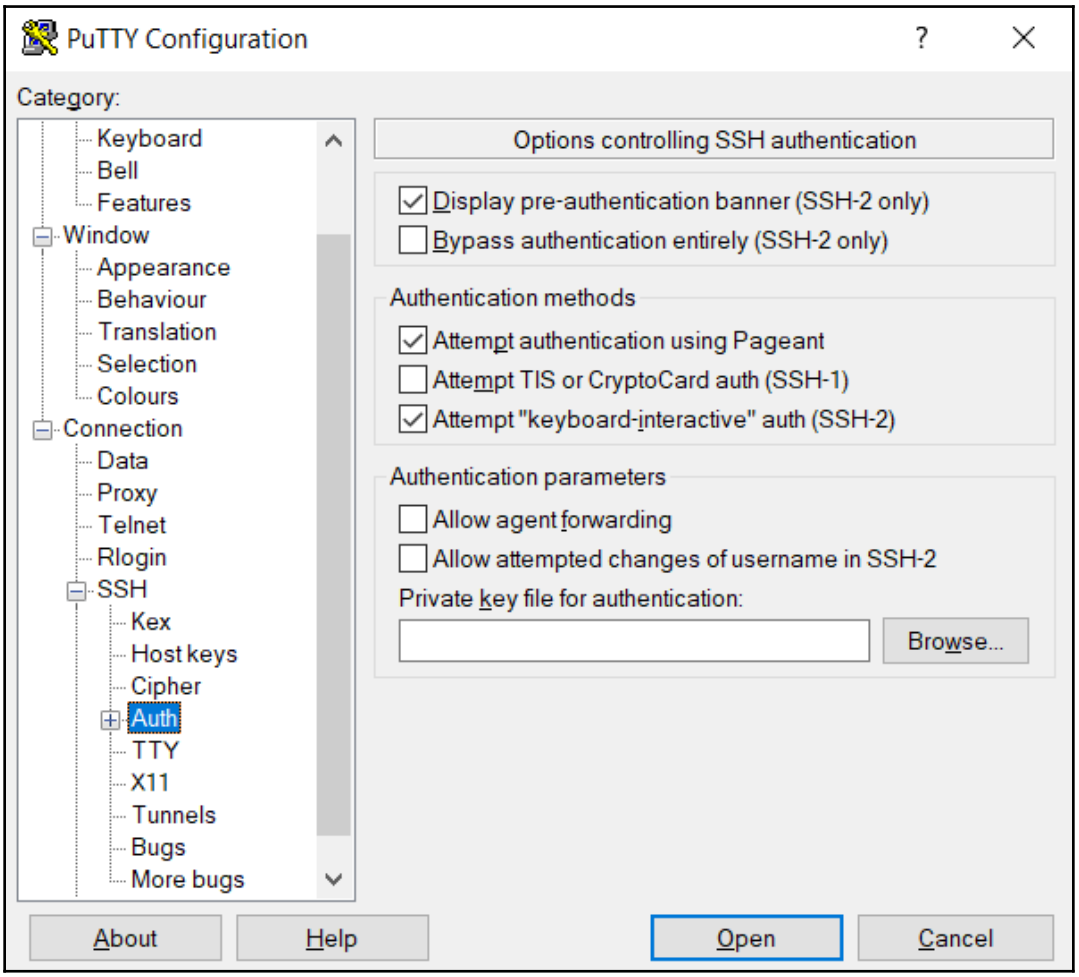
Now, select the key pair file that we downloaded previously and choose **Open**.

This will convert the key into a format that PuTTY can use:



Click on **Save private key**, and save it without a passphrase. Make sure you save it as a `.ppk` file. Now, we are ready to launch PuTTY and SSH into our instance.

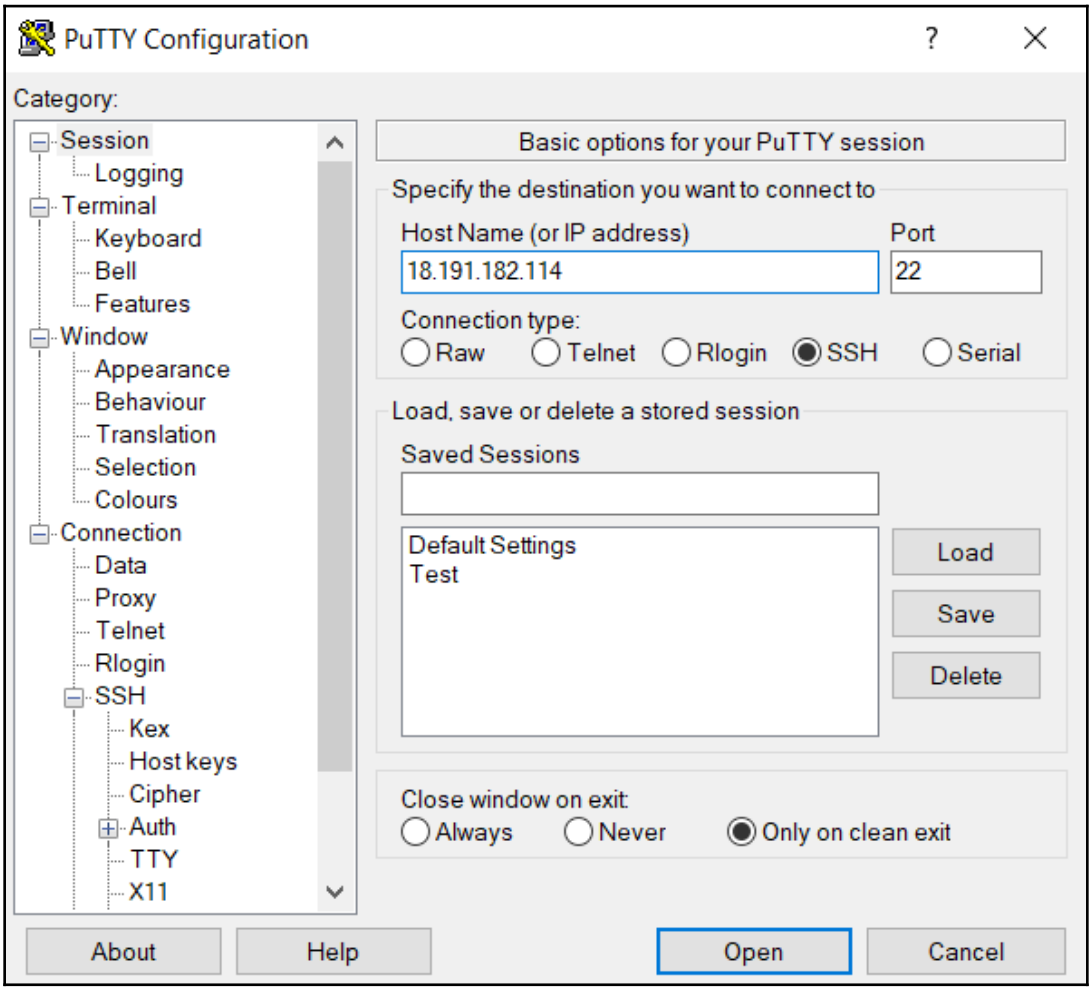
Open PuTTY, and under the **Connection** category, choose **SSH** and then **Auth**:



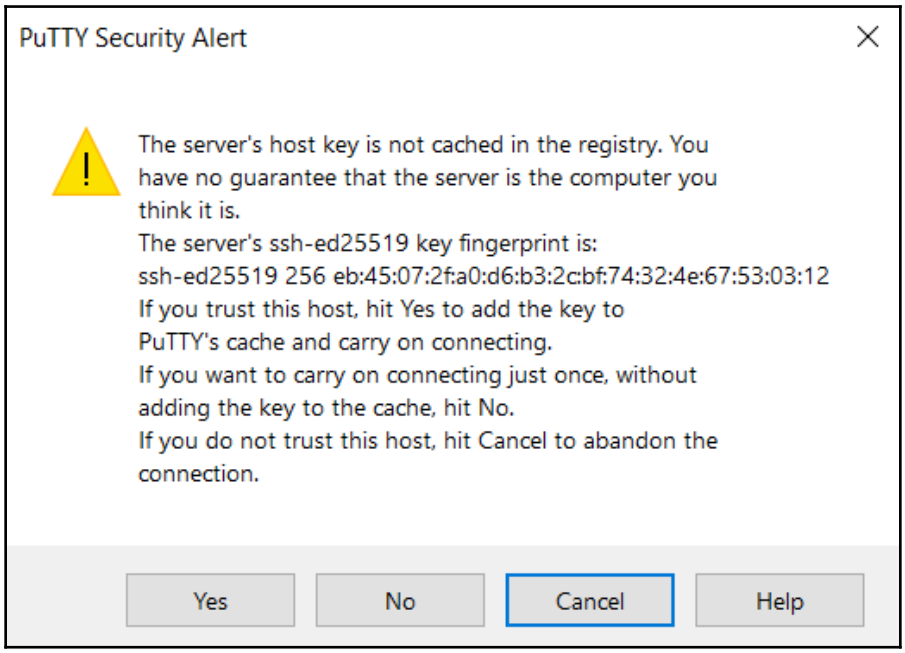
Browse... to the `.ppk` file and then click **Open**

Now, go back up to the top and choose **Session**.

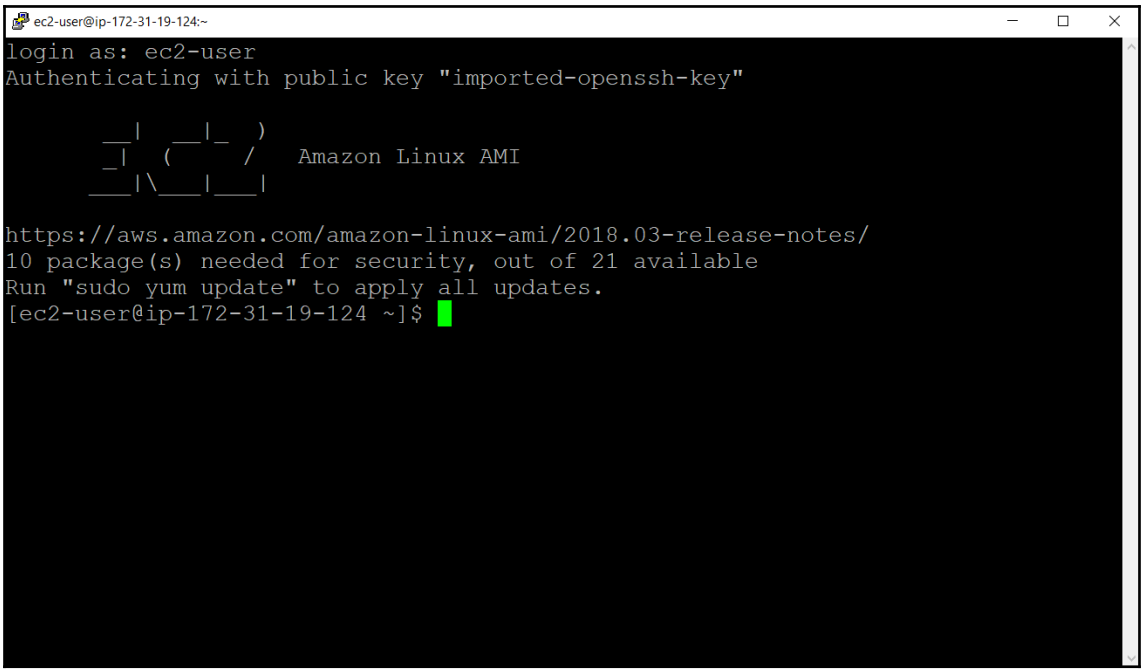
In the **Host Name (or IP address)** field, enter `ec2-user@` and the IP address of the EC2 instance that we saved before. If this was an Ubuntu instance, you would enter `ubuntu` instead of `ec2-user`:



Finally, click the **Open** button to begin the SSH session. Choose **Yes** at the dialog box:



Now, we are logged in to our SSH session:



This user has `sudo` privileges, so just to test that, we can enter the following command:

```
$ sudo yum update
```

This will update the Linux packages:

```
ec2-user@ip-172-31-19-124:~
 kernel              x86_64  4.14.70-67.55.amzn1            amzn-updates   21 M
Updating:
 amazon-ssm-agent    x86_64  2.3.68.0-1.amzn1               amzn-updates   17 M
 aws-cli             noarch  1.15.83-1.49.amzn1             amzn-updates  1.2 M
 e2fsprogs           x86_64  1.43.5-2.42.amzn1              amzn-updates  1.5 M
 e2fsprogs-libs      x86_64  1.43.5-2.42.amzn1              amzn-updates  208 k
 gnupg2              x86_64  2.0.28-2.33.amzn1              amzn-updates  2.6 M
 java-1.7.0-openjdk  x86_64  1:1.7.0.191-2.6.15.4.82.amzn1  amzn-updates   32 M
 kernel-tools        x86_64  4.14.70-67.55.amzn1            amzn-updates  126 k
 krb5-libs           x86_64  1.15.1-19.43.amzn1             amzn-updates  861 k
 libcom_err          x86_64  1.43.5-2.42.amzn1              amzn-updates   47 k
 libss               x86_64  1.43.5-2.42.amzn1              amzn-updates   52 k
 libxml2             x86_64  2.9.1-6.3.52.amzn1             amzn-updates  723 k
 libxml2-python27    x86_64  2.9.1-6.3.52.amzn1             amzn-updates  331 k
 ntp                 x86_64  4.2.8p12-1.39.amzn1            amzn-updates  1.0 M
 ntpdate             x86_64  4.2.8p12-1.39.amzn1            amzn-updates   89 k
 openssh             x86_64  7.4p1-16.71.amzn1              amzn-updates  639 k
 openssh-clients     x86_64  7.4p1-16.71.amzn1              amzn-updates  1.1 M
 openssh-server      x86_64  7.4p1-16.71.amzn1              amzn-updates  511 k
 openssl             x86_64  1:1.0.2k-12.110.amzn1          amzn-updates  1.8 M
 procps              x86_64  3.2.8-45.16.amzn1              amzn-updates  250 k
 python27-botocore   noarch  1.10.82-1.67.amzn1             amzn-updates  4.5 M
Installing for dependencies:
 copy-jdk-configs    noarch  3.3-10.3.amzn1                 amzn-updates   21 k
 fuse-libs           x86_64  2.9.4-1.17.amzn1               amzn-main      98 k

Transaction Summary
================================================================================
Install   1 Package  (+2 Dependent packages)
Upgrade  20 Packages

Total download size: 87 M
Is this ok [y/d/N]:
```

Input y to update packages.

Logging in to Windows instances

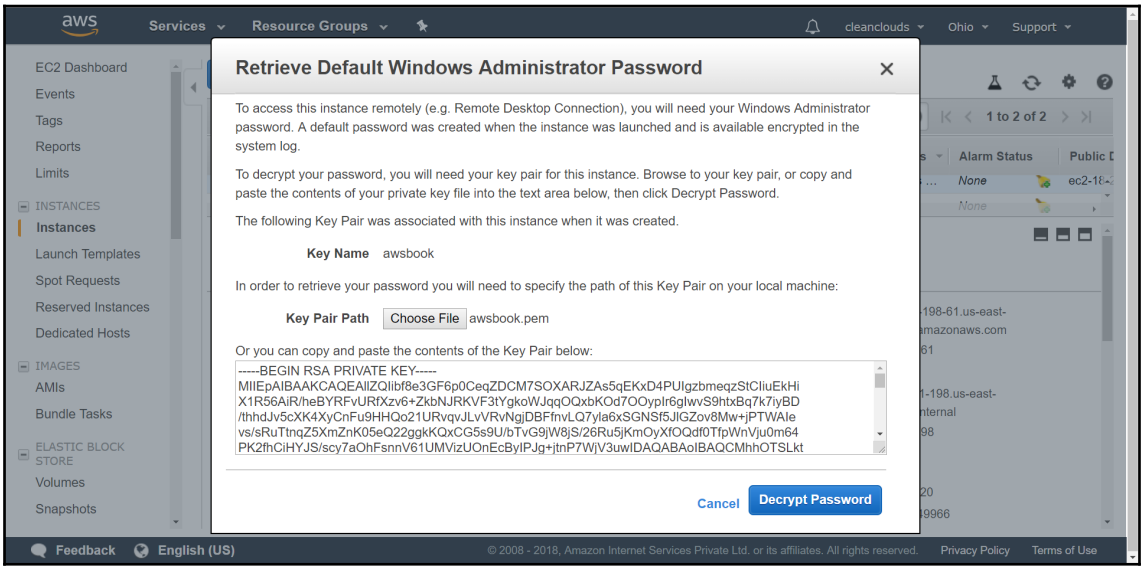
In the previous section, we saw how to connect to Linux instances using a private key for authentication. In this section, we're going to see how the process works for Windows instances.

For Windows, we use the remote desktop protocol, and log in as administrator. We will obtain the administrator password, then we will log in to the instance from Windows. When AWS launches in a Windows instance, it creates a random password for the administrator user. The public key associated with the instance is used to encrypt the password. The encrypted password can only be decrypted with the private key portion of the key pair.

You can decrypt the password easily in the management console.

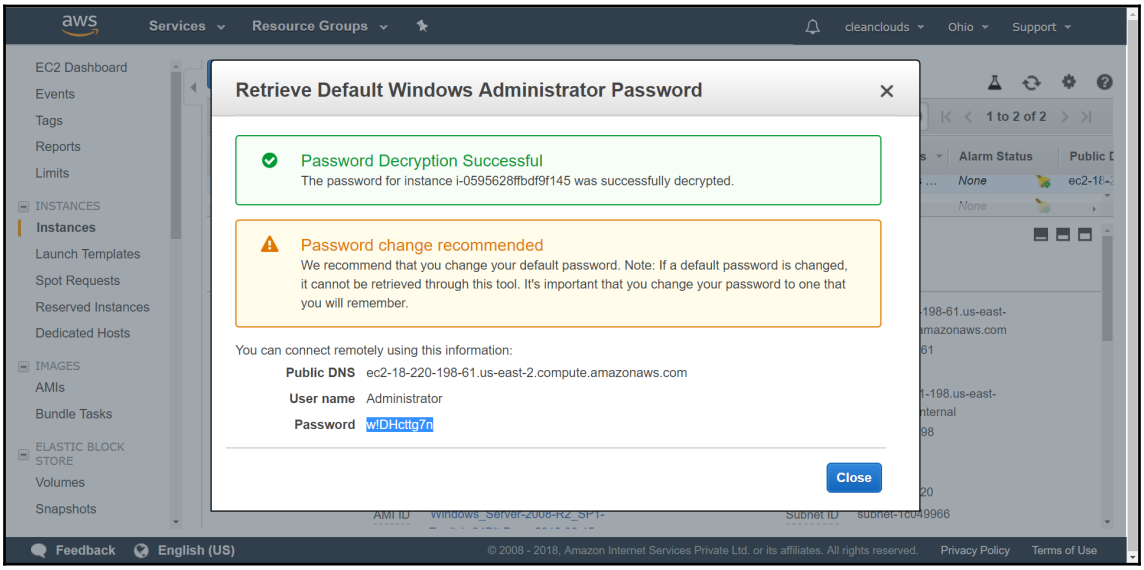
Go to **EC2**, go to **Running Instances**, select the instance, and under the **Actions** menu choose **Get Windows Password**.

Notice that it gives the name of the key pair that was used when the instance was launched:



In this case, it was `awsbook`. So, we need to select the private key.

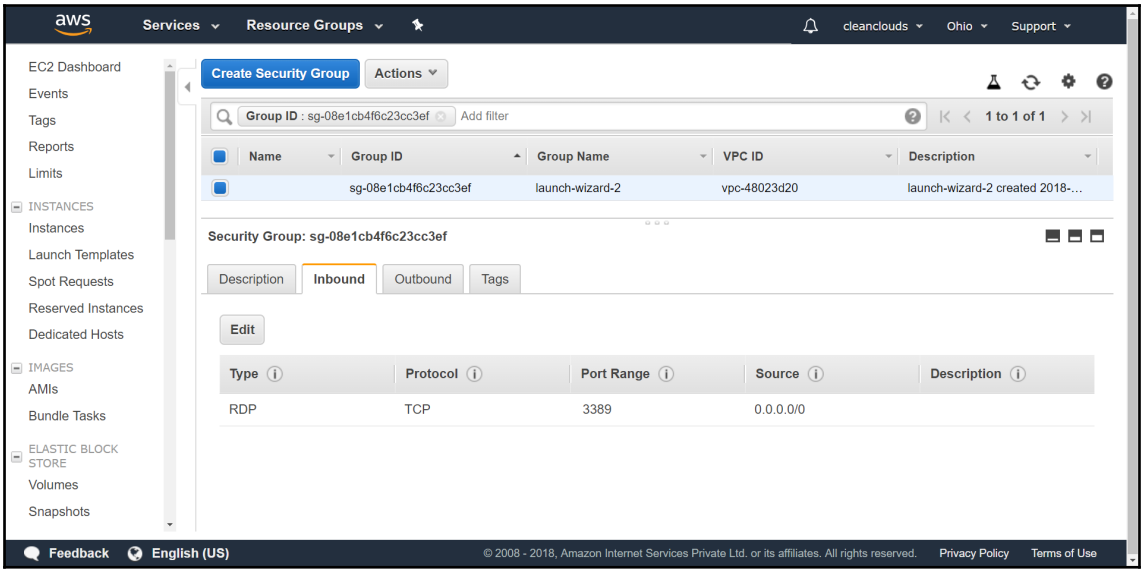
Then, click **Decrypt Password**:



Now, we have the password for the administrator login. Let's make a note of the **Public DNS** and **Password**. Copy that and place it on the notepad.

Note that, in order to connect, there has to be a rule in our security group that allows remote desktop. So, let's check the security group and add that rule.

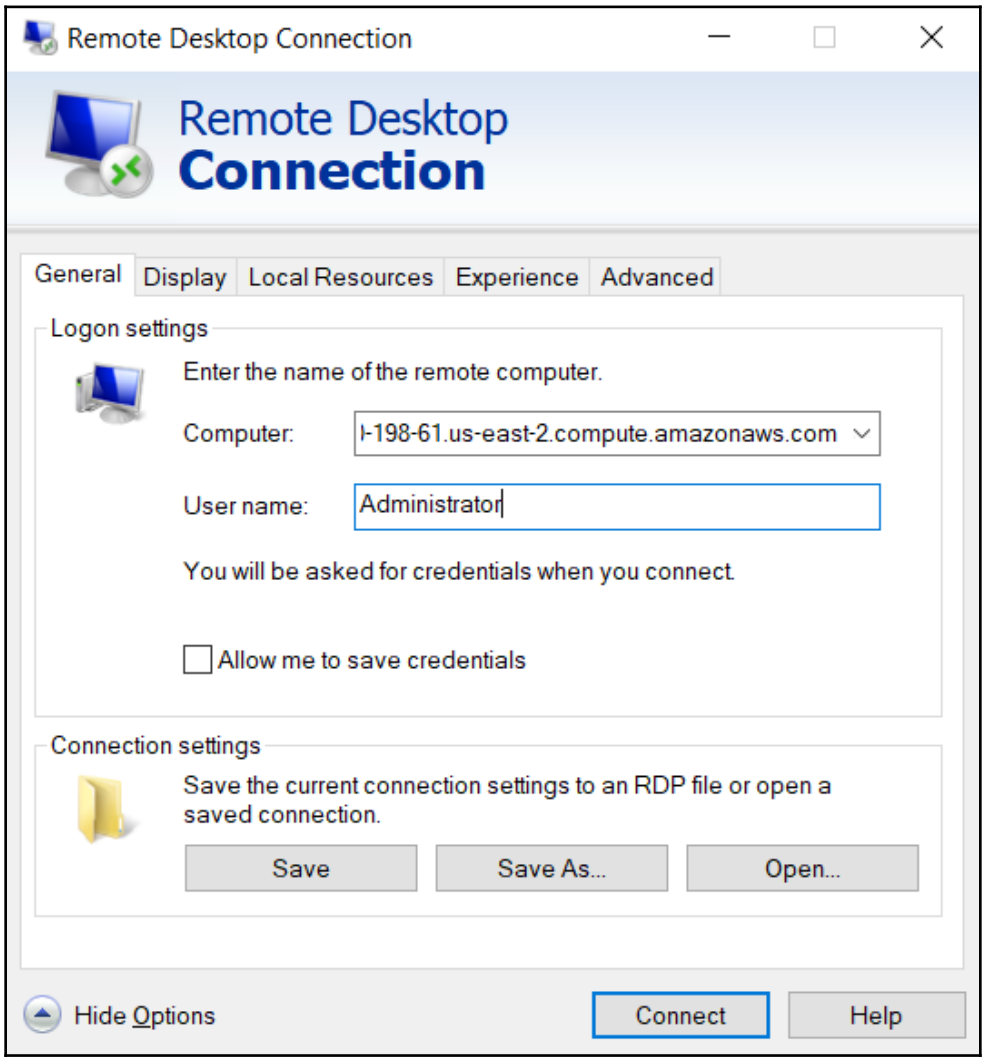
Select **RDP**, which will already fill in the **Port Range 3389**:



Select your IP address and click on **Save**.

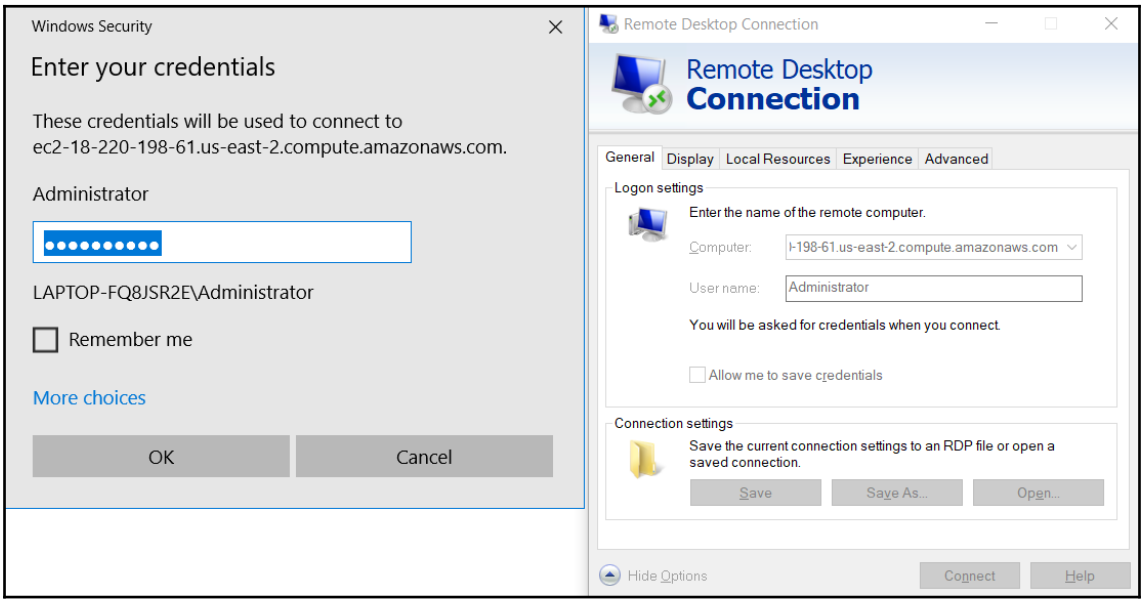
Now, we should be ready to RDP into the instance.

For Windows, we'll just search for the remote desktop client, and enter our connection information. The **User name** should be `Administrator`:

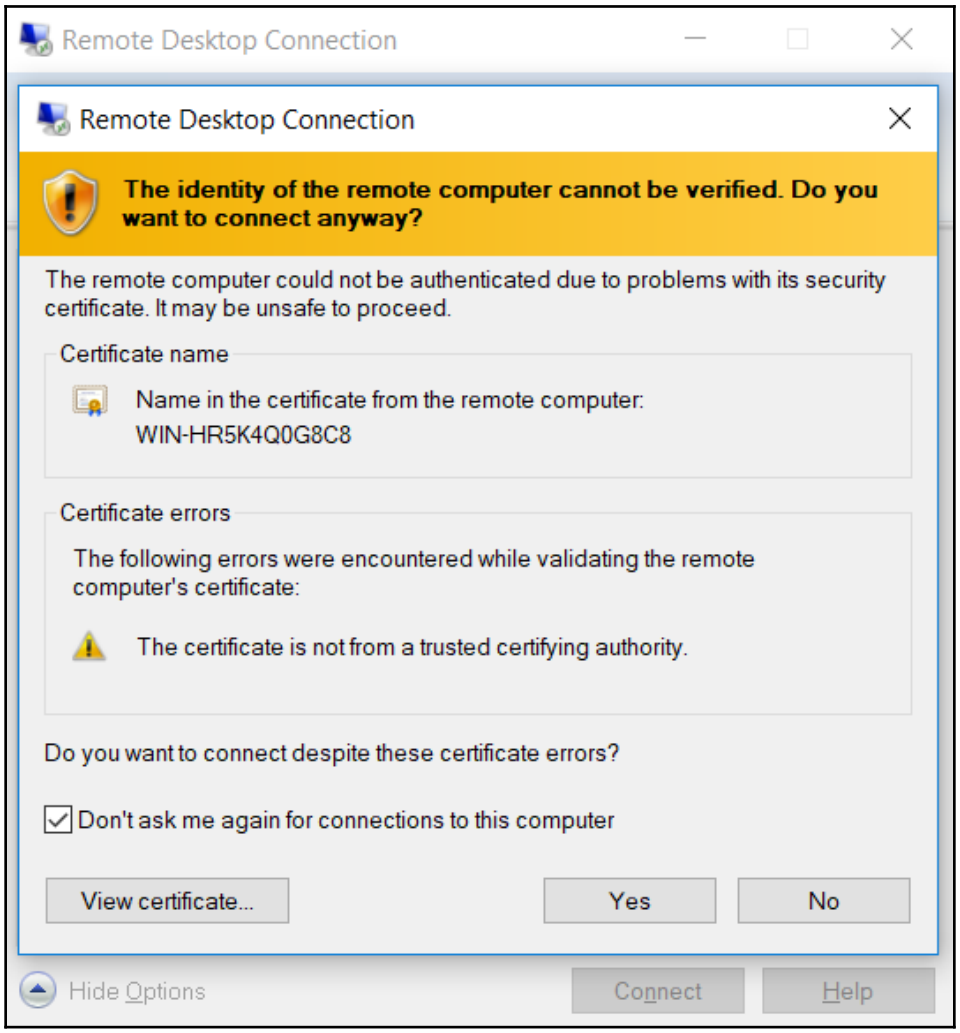


Copy the previously saved password into the clipboard and choose **Connect**.

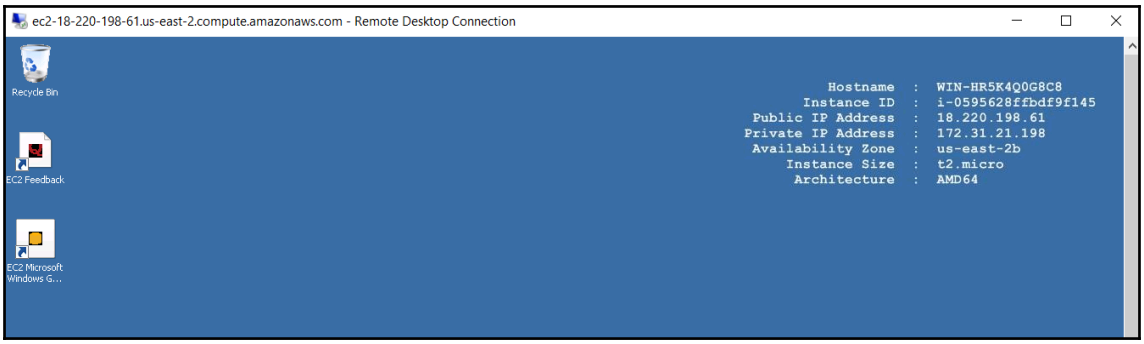
Enter the password and click **OK**:



Verify the identity of the instance:



Now, we have a remote desktop connection to the EC2 instance:



Summary

In this chapter, we learned about key pairs, how to generate them, and how to associate them with an EC2 instance. We have logged in to a Linux instance, from Windows. We saw how to connect to a Windows instance.

In `Chapter 4`, *Networking on AWS*, we will cover networking on AWS, including IP addressing, subnets, and routing tables.

4
Networking on AWS

In this chapter, we will explain how to define the IP address block for your virtual private cloud, also known as a VPC. We will start by explaining the **Classless Inter-Domain Routing** (**CIDR**) notation, which IP addresses are available for you to use for your VPC, and then the three types of IP address on AWS: public, private, and elastic. Next, we will discuss dividing your VPC into one or more subnets, and finally defining routes for your network traffic in route tables.

The main topics that we will cover are as follows:

- CIDR notation
- Private, public, and Elastic IP addresses
- Subnets
- Route tables

Now, let's move on to the first section in this chapter, CIDR .

CIDR

When you work with VPCs, subnets, security groups, and network access control lists, you will often need to specify IP address ranges. CIDR provides a succinct method, known as CIDR notation, for defining IP address blocks.

In this section, we're going to briefly explain CIDR notation and then review the valid IP address ranges you can use when defining your VPC and subnets. We'll also discuss IP addresses that are reserved by AWS, and why this is important when you're sizing your subnets.

IPv4

VPCs on AWS use version 4 of the Internet Protocol, known as IPv4. We can assign the IPv6 CIDR block to our VPC, and assign IPv6 CIDR blocks to our subnets. An IPv4 address is defined using 32 bits, broken up into four parts (4x8 bits). Each part is 8 bits, and so their values range from 0 to 255. So, an IP address can be anywhere from `0.0.0.0` to `255.255.255.255`. This means that the total number of IP addresses available globally is 2 to the 32 power, or 4,294,967,296, so basically just under 4.3 billion. CIDR notation makes it easy to specify a range of IP addresses by appending a slash and the number of fixed bits, from 0 to 32, after the address. It is easiest to explain this by an example (`/24`, `/28`).

By default, Amazon EC2 and Amazon VPC use the IPv4 addressing protocol

For a single IP address, we need to provide all 32 bits, so an example of a single IP would be `192.168.55.31/32`. If instead we wanted to define a range of IP addresses, we would use a number less than 32 for the prefix length. When we use `/24`, for example, we are saying that the first 24 bits are fixed, and the remaining 8 bits are unspecified. This would give us an IP address block consisting of 256 addresses:

`192.168.55.0/24` - `192.168.55.0` → `192.168.55.255` (256 addresses)

For `/16`, there would be 32 minus 16, which equals 16 unspecified bits, and so this represents 65,536 addresses:

`192.168.0.0/16` - `192.168.0.0` → `192.168.255.255` (65,536 addresses)

As a final example, `/20` would leave 12 bits unspecified, or 4,096 addresses:

`192.168.96.0/20` - `192.168.96.0` → `192.168.111.255` (4,096 addresses)

the following is an example of the available range for IP addresses, Subnet Masks, and IP Quantity:

CIDR Block	IP Range	Subnet Mask	IP Quantity
10.0.0.0/32	10.0.0.0 - 10.0.0.0	255.255.255.255	1
10.0.0.0/31	10.0.0.0 - 10.0.0.1	255.255.255.254	2
10.0.0.0/30	10.0.0.0 - 10.0.0.3	255.255.255.252	4
10.0.0.0/29	10.0.0.0 - 10.0.0.7	255.255.255.248	8
10.0.0.0/28	10.0.0.0 - 10.0.0.15	255.255.255.240	16
10.0.0.0/27	10.0.0.0 - 10.0.0.31	255.255.255.224	32
10.0.0.0/26	10.0.0.0 - 10.0.0.63	255.255.255.192	64
10.0.0.0/25	10.0.0.0 - 10.0.0.127	255.255.255.128	128
10.0.0.0/24	10.0.0.0 - 10.0.0.255	255.255.255.0	256

Here is a link to one of the many online calculators that will convert CIDR notation to starting and ending IP addresses: `http://www.ipaddressguide.com/cidr`.

Valid private IP address ranges

For VPCs, we need to specify private IP address blocks. The Internet Assigned Numbers Authority (**IANA**) has reserved certain IP address ranges to be used for private IPs.

The following are few examples:

```
10.0.0.0        -  10.255.255.255     (16,777,216 addresses)
172.16.0.0      -  172.31.255.255     (1,048,576 addresses)
192.168.0.0          -  192.168.255.255 (65,536 addresses)
```

So, the CIDR blocks for your VPCs need to fit into these ranges.

You can find some of them defined by RFC1918, as follows:

`https://tools.ietf.org/html/rfc1918`

On AWS, VPCs can be made as small as `/28`, which is 16 addresses, all the way up to `/16`, which is `65536` addresses.

However, think twice before you create a VPC as small as `/28`. You will have to create at least one subnet in your VPC, and for every subnet in your VPC, five addresses are reserved by AWS. These are the first four and last one in the subnet block. So, that leaves only 11 IP addresses for your `/28` VPC:

IP Addresses	Reserved for
10.0.0.0	Network address
10.0.0.1	Reserved by AWS for the VPC router
10.0.0.2	Reserved by AWS
10.0.0.3	Reserved by AWS for future use
10.0.0.255	Network broadcast address

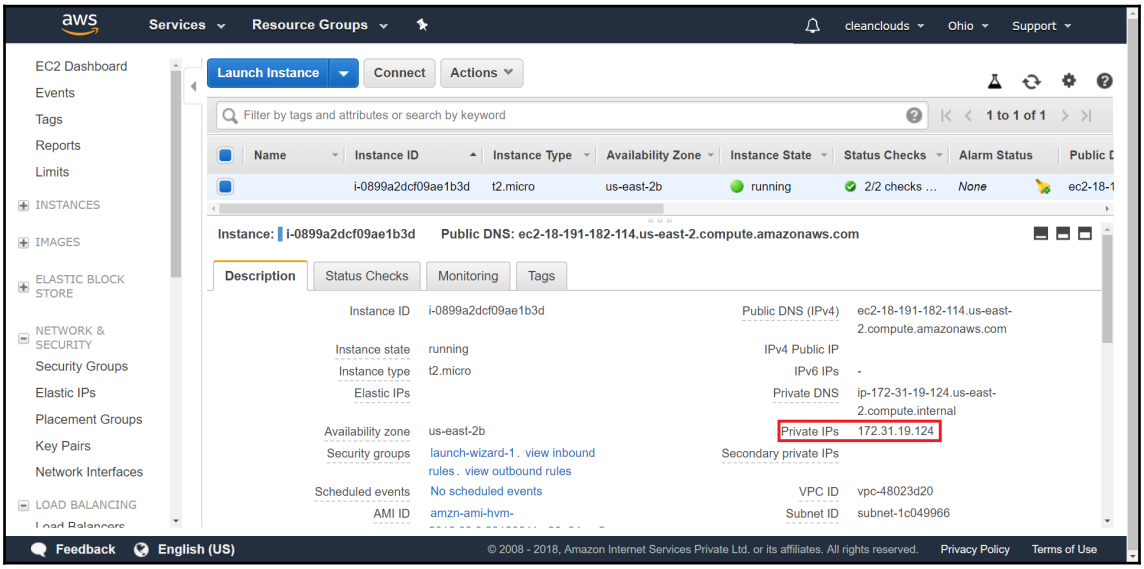
This completes our discussion of CIDR notation. In the next section, we will cover the types of IP address you can assign to your EC2 instances, and we will introduce Elastic Network Interface.

EC2 IP addressing

In this section, we're going to continue our discussion of private IP address and the two types of public IP address you can use for your EC2 instances. We are also going to discuss Elastic Network Interface, which are the virtual network cards attached to your instance.

Private IP addresses

Every instance in your VPC must be placed inside a subnet. A random available private IP address from the subnet range is automatically assigned to each instance. Private IP addresses are not routable from the internet:

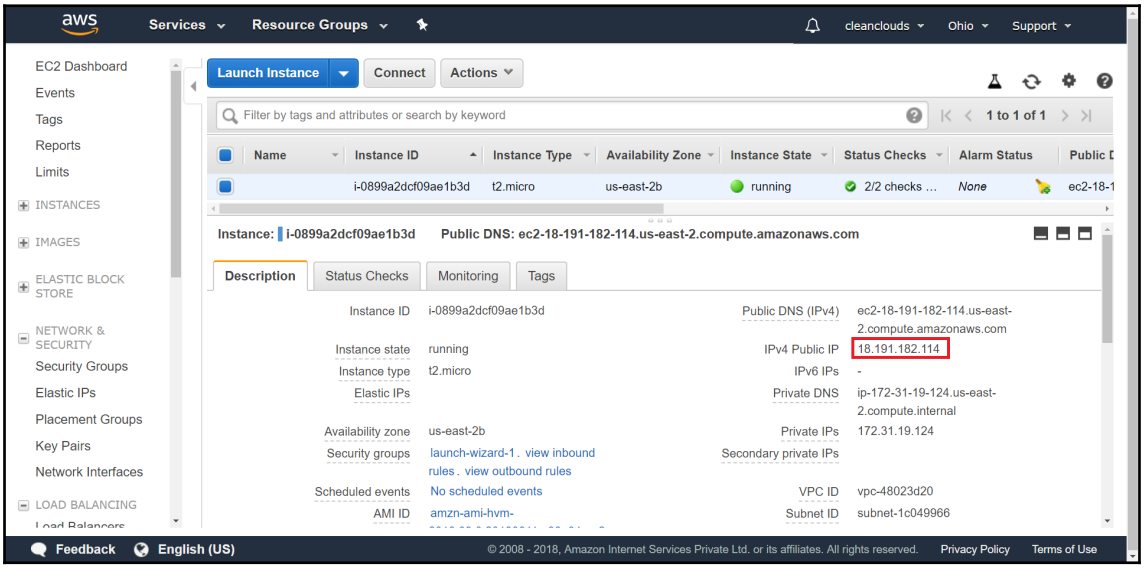


If you want your instance to be reachable from the internet, then you will need a publicly routable IP address assigned to the instance's Elastic Network Interface. There are two types of public IP address that can be assigned to your instance.

Public IP addresses

AWS owns a large but finite number of public IP addresses. When you launch an instance, you can choose to have a public IP randomly assigned to it. You only need a public IP address if you want the instance to be reachable from the internet, so it is optional. When you create a subnet, which we will do later, you can set the subnet so that instances that launch in it will get a public IP automatically. However, when launching an instance, you can choose to override that and not receive a public IP.

There is no cost for assigning public IPs to your instance; however, to follow best security practices, you shouldn't get a public IP for your instance if you don't need it. One of the nice features of EC2s is that you can stop them whenever you want to stop the billing. However, when you stop your instance, AWS will take back your public IP address. When you start your instance up again, it will be assigned a new public IP address, different from the original. This will cause a problem if applications that connect to your instance have hardcoded the old address. Also, you would have to update any DNS records that point to the old address. Even if you don't plan on stopping your instance, sometimes you have to if the instance becomes impaired:



To avoid this problem, AWS has a second type of public IP address, known as an Elastic IP.

Elastic IP addresses

A public IP is given to an instance when you launch or start it, but an Elastic IP is independent of the instance lifecycle.

You can, at any time, ask for an Elastic IP address. It is given to your account, not a specific instance. One of the great things about an Elastic IP is that you can associate it with an instance that already exists, even if it is already running. And if you stop the instance, it does not become dissociated from the instance. So, you can freely stop and start EC2 instances with Elastic IPs, and they will retain the same public address. By default, you can request up to five Elastic IPs per region. If this isn't enough, you can request more by contacting AWS support. One Elastic IP is free per instance.

However, if your instance is not in the running state, you'll be charged an hourly fee. So that means that, when you stop an instance, you'll be charged a small fee per hour for the Elastic IP that is associated with it. Also, if you have Elastic IPs in your account that are not associated with a running instance, you will be charged.

Elastic network interface (ENI)

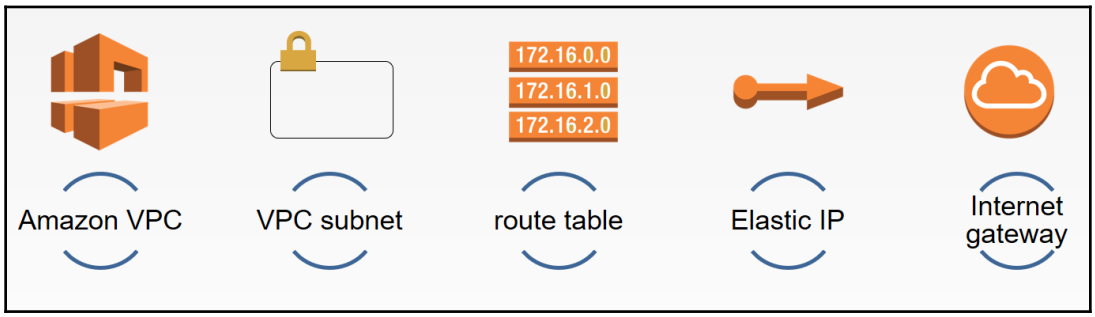
Each EC2 instance has one or more Elastic Network Interface, or ENIs, attached, and the IP addresses are actually assigned to these. You can think of them as virtual network cards.

Each instance has one ENI, designated `eth0`, attached by default, known as the primary ENI. The IPs given to your instance at launch are assigned to the primary ENI. The primary ENI cannot be detached from the instance. However, you can create additional ENIs, and attach them as secondary network interfaces to an instance. This doesn't increase the overall network capacity of the instance, but you can detach a secondary ENI and move it to another instance. This will actually move the IP addresses with it. So if you need to replace an impaired instance, or just want to replace it with a new type, you can launch a new instance and move the secondary ENI over from the old instance. This will move the associated IP addresses, so you won't have to update DNS or application config files. The ENI will have one primary private IP address, and you can add one or more secondary private IP addresses. All of these would of course come from your subnet's CIDR block. You can choose to associate an Elastic IP for each of the private IPs.

However, only one Elastic IP per instance is free. You will have to pay a small hourly rate for more than one Elastic IP per instance. The regular public IP address is also a property of the ENI, but if you associate an Elastic IP, the public IP will be returned to AWS. Security groups are also associated with the ENI, and are not directly associated with the instance. Each ENI will have a unique MAC address. This is useful if you have a software license tied to a specific MAC address. If you want to move the application to a different instance, you can move the ENI, and that will move the MAC address. Finally, ENIs have an attribute known as the source destination check flag. Enabled by default, this blocks network traffic that isn't destined for your instance. You can disable this check if you want your instance to do network address translation, routing, or serve as a firewall. In the next section, we will describe subnets in more detail and introduce route tables.

Subnets and route tables

In the previous section, we learned about private, public, and Elastic IP addresses, and how they are assigned to the ENIs attached to an instance. Following are a few components that are important in the Amazon VPC:



In this section, we're going to discuss subnets and route tables, and how a route in the route table can make a subnet public or private. Then, we'll talk about NAT instances and NAT gateways, to give instances and private subnets access to the internet.

Do you need to create a VPC the moment you create your account? The answer is, No. A default VPC is available on the Amazon VPC. If you delete the default VPC then you cannot restore it. You need to contact AWS Support.

What are subnets?

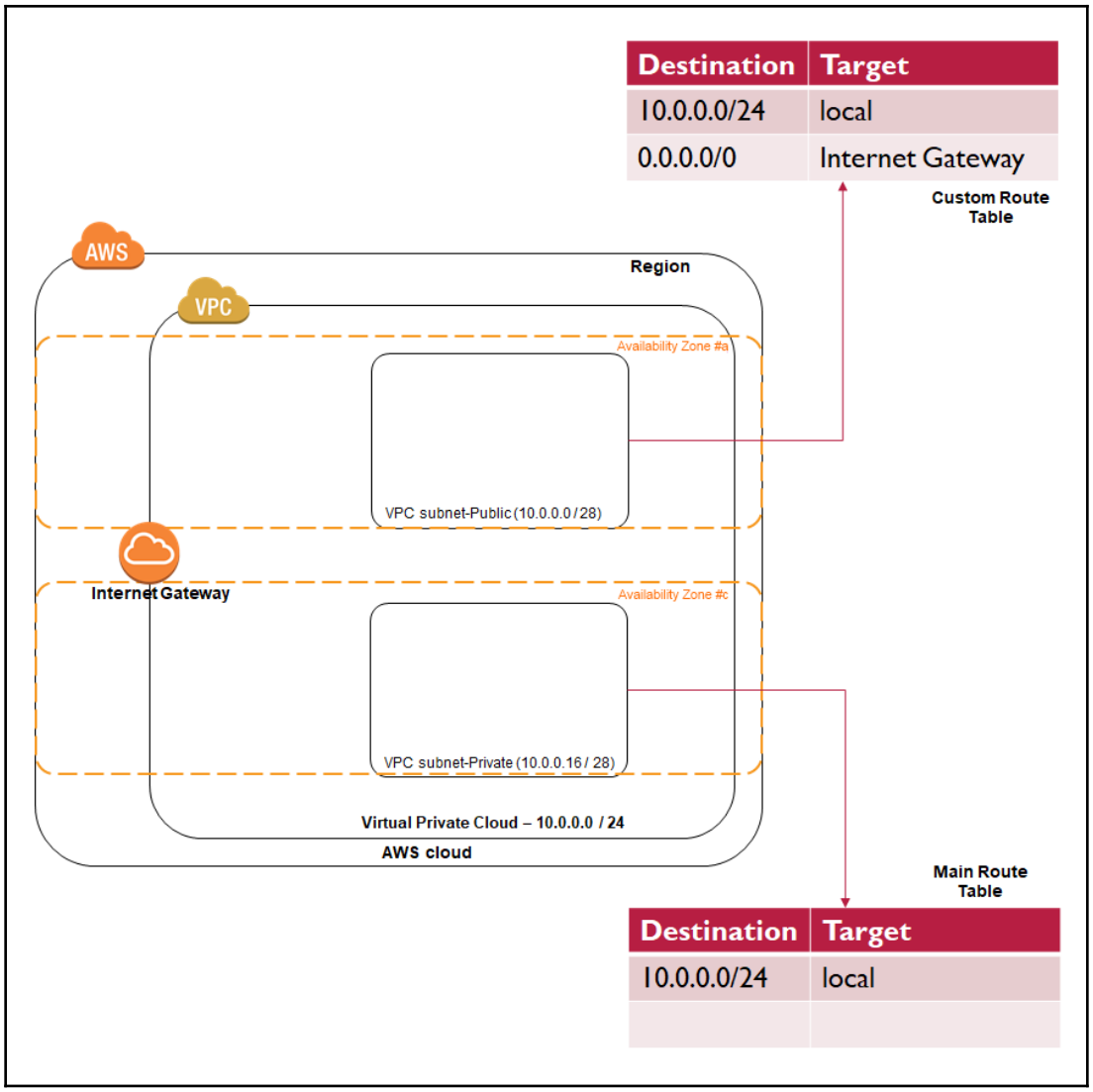
Subnets are separate portions of the VPC. EC2 instances must be launched into a subnet, so at least one subnet is required for a VPC. To create a subnet, you have to define a CIDR block for it, which is a subset of the VPC's CIDR block. Every subnet in a VPC has to have a unique range of addresses. There can't be any overlap, or else there would be a duplicate IP address in your network.

Route tables

Now, let's talk about route tables. For instances to communicate, there has to be a route that defines a path for the network traffic to take. A route consists of a destination, designated as a CIDR address block, and a target, which defines the path to take. An example of a route would be a local route that allows communications to all instances in the local network, in our case a VPC:

Destination	Target
10.0.0.0/16	Local
192.168.0.0/20	pcx-1234abcd
0.0.0.0/0	igw-1a2b3c4d

To communicate with instances in another VPC, you could define a VPC peer connection and define the peer connection ID as the target. To access the internet, you could define an internet gateway as a target. Every VPC will have a default route table with a single local route; this defines a route so instances inside a VPC can communicate with one another:



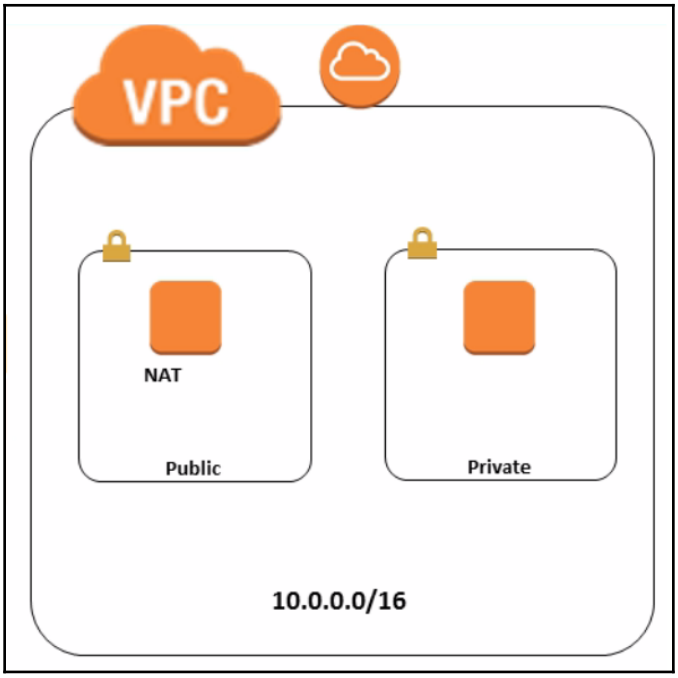
You cannot remove this route, but you can customize the table by adding additional routes. You can create additional route tables and associate them with your subnets. This allows you to create what are known as public and private subnets.

Each subnet can have only one route table. Multiple subnets can be assigned to a single route table.

Difference between public and private subnets

A public subnet will be used for instances that need a public IP to be accessible from the internet. The way we make it a public subnet is to associate a custom route table and add a route to the internet, with an internet gateway as a target. Notice that the internet is defined with CIDR notation as `0.0.0.0/0`. An internet gateway must first be attached to your VPC. If you don't have an internet gateway attached to your VPC, none of your instances will be able to reach the internet.

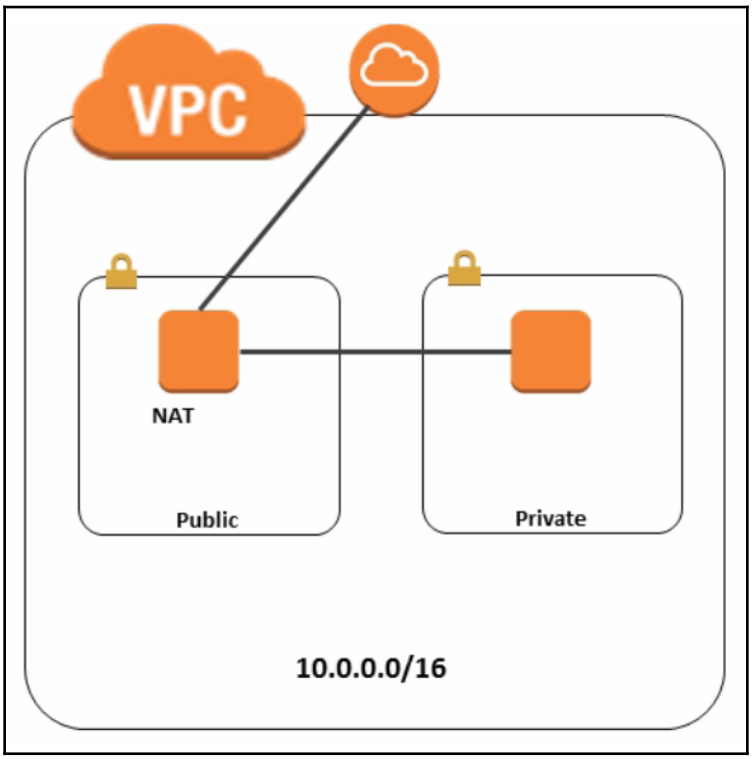
A private subnet is used for instances that do not need to be directly reachable from the internet. For the best security, it's important to keep backend instances and databases private, so those would go in a private subnet. A private subnet would be associated with a route table that does not have a route to the internet through the internet gateway. Even though private instances will not be directly reachable from the internet, we will probably need to provide a way for them to get out to the internet, to download software patches and to access external services such as AWS's DynamoDB, a NoSQL database. To do this, we can configure an instance in a public subnet to do Network Address Translation, or NAT:



NAT instance

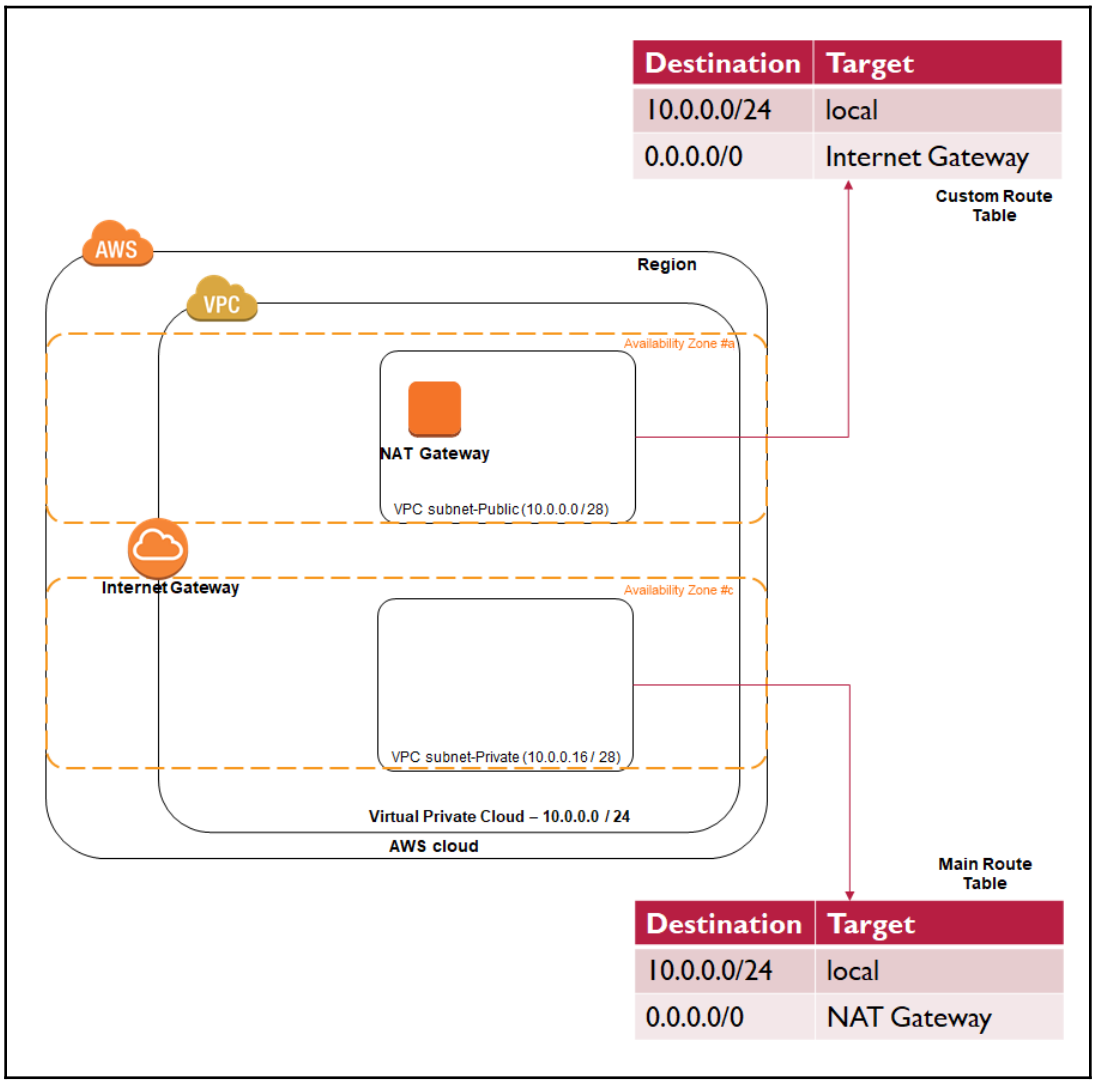
An NAT instance will forward outbound requests for private instances to the internet, and send the responses back to the correct instance.

You can easily launch an NAT instance by using an AMI with the correct network configuration:



You can find these on the community's AMI tab in the console when you're launching the instance. You will also need to add a route in the custom route table for the private subnet. For this route, you will specify `0.0.0.0/0` for the destination, and the NAT instance ID for the target. There are a few things to watch out for when you use an NAT instance. One is that it is a single point of failure, so you will need to use some automation to monitor the NAT instance, and recover it or failover to a second NAT. Another issue is that the network bandwidth will be limited by the instance type and size. A better option is to use AWS's NAT gateway service.

AWS will manage the availability of the NAT gateway so you don't have to, and it provides bursting network bandwidth up to 10 GB/s:



Summary

In this chapter on networking in AWS, we first explained CIDR notation, which we will use to define IP address ranges for our VPC and subnets. Then we covered public, private, and Elastic IP addresses. We explained how these are assigned to Elastic Network Interface, and how they could be moved from one instance to another. Next, we explained dividing our VPC into subnets and using route tables to make subnets public, or to give instances in private subnets a path to the internet through a NAT instance or NAT gateway.

In the next chapter, we will create a VPC from scratch, and add additional security for our subnets with network access control lists. Then, we will describe ways to connect to our VPC.

5
Creating a VPC

In `Chapter 4`, *Networking on AWS*, we discussed networking on AWS, which laid the foundation for being able to create our own VPCs. We discussed IP addressing, subnets, and route tables. In this chapter, we will learn several methods to build, secure, and connect to a VPC. First, we're going to look at classic EC2s, which are instances that are launched outside of a VPC. Then, we'll talk about the VPC that AWS already creates for you, the default VPC. Next, we'll demonstrate creating a VPC, using the VPC Wizard, and then creating one from scratch. After that, we'll talk about several ways to connect to the instances in your VPC, and then we'll make your VPCs more secure by introducing network access control lists and Bastion instances.

Finally, we'll discuss making your architectures highly available by leveraging multiple availability zones, load balancing, and auto scaling.

The main topics that we will cover are as follows:

- VPC EC2s versus classic EC2s
- The default VPC
- Creating a VPC
- Connecting to a VPC
- Securing your VPC
- High availability

Getting started with VPCs

In this section, we will begin with a little history lesson by talking about classic EC2s, and comparing them with EC2s that are launched in a VPC.

Classic EC2s

EC2s were first introduced by AWS back in 2006. Back then, there was only one big public network in which to launch your instances. Every instance was automatically assigned a public and private IP address, controlled by AWS. If you stopped your instance for any reason, AWS took back your IPs, and when you started it up again, you got new ones.

Since every instance had a public IP address, they were all essentially public. So you had to rely on security groups to restrict access to your databases and other instances that you wanted to keep private, and the security groups only allowed you to specify inbound rules. All outbound traffic was always allowed. In 2009, AWS launched VPCs and encouraged customers to launch instances in these virtual private networks, instead of in the big public network. EC2s that are not launched in a VPC, are today called classic EC2s. These include RDS instances, which also had to be launched in the public network. In 2013 AWS declared that all accounts created after December 4th 2013, would have to launch their instances in VPCs. However, AWS accounts created before that date are grandfathered, and can still launch classic EC2 and RDS instances today.

EC2s in a VPC

If you, or one of your customers, have one of these older AWS accounts, migrating the classic instances to a VPC is highly recommended. Some of the advantages are the ability to specify your own private IP address ranges, and keep the private IPs associated, when you stop and start your instances. You can choose not to get a public IP address, and also put your instance in a private subnet with no route to the internet. This is a powerful extra layer of security, in addition to security groups.

Security groups in a VPC allow you to create rules for outbound as well as inbound traffic, and by dividing your VPC into subnets, you can control traffic in to and out of your subnet with NACLs and custom route tables.

The default VPC

VPCs cannot span regions, so AWS creates a default VPC in each region.

AWS Account supports EC2 instances in a VPC only. The default VPC is available in the Amazon VPC. If you delete the default VPC, then you cannot restore it. You need to contact AWS Support.

Click on **Services** | Go to **Networking & Content Delivery** section | Click on **VPC** | Click on **VPC Dashboard**:

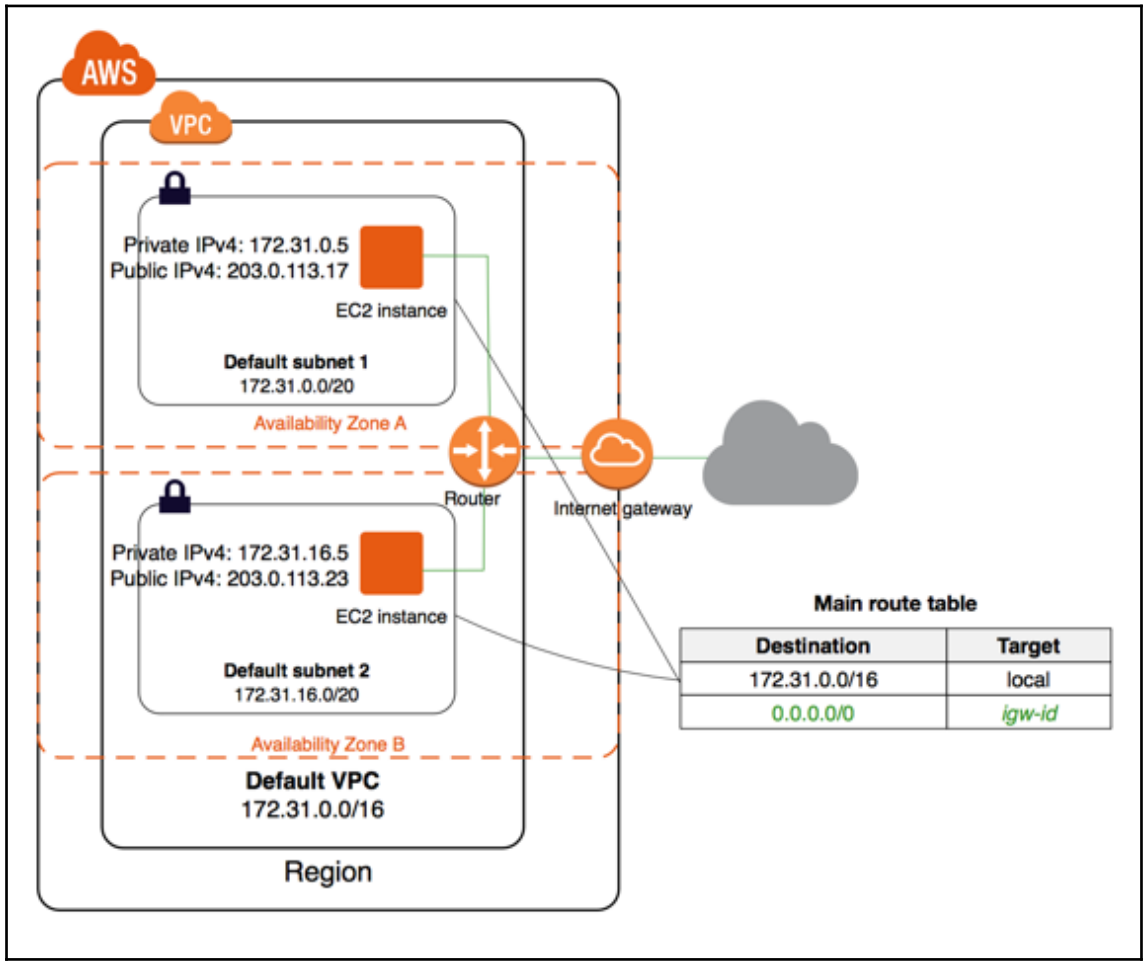
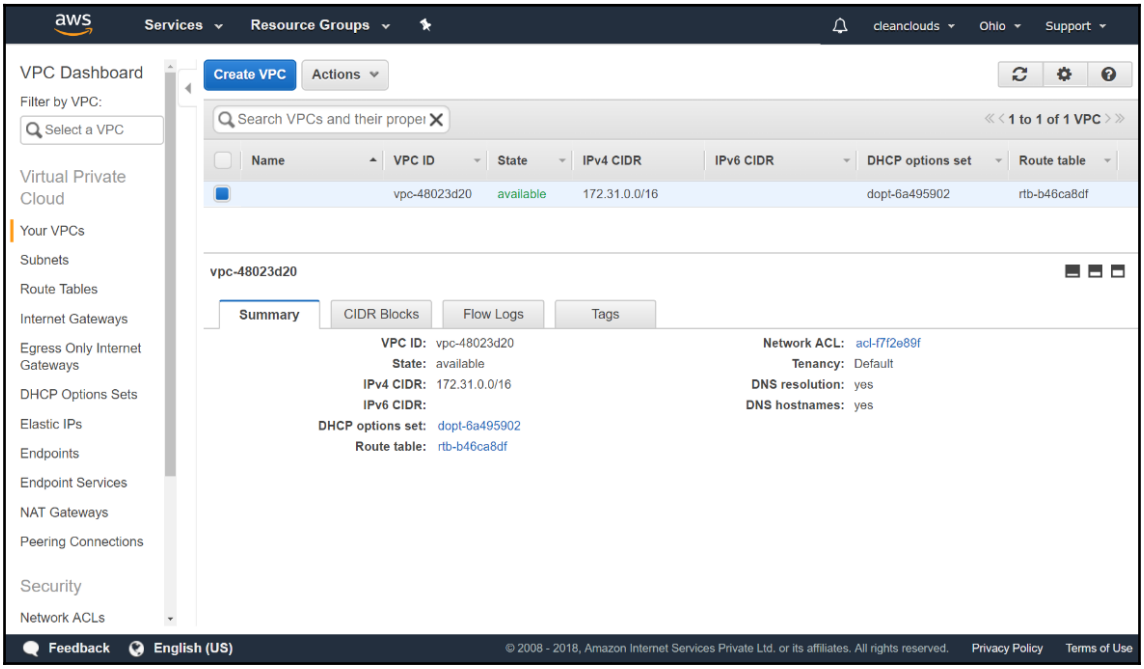


Figure reference :http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/default-vpc.html

The default VPC contains following:

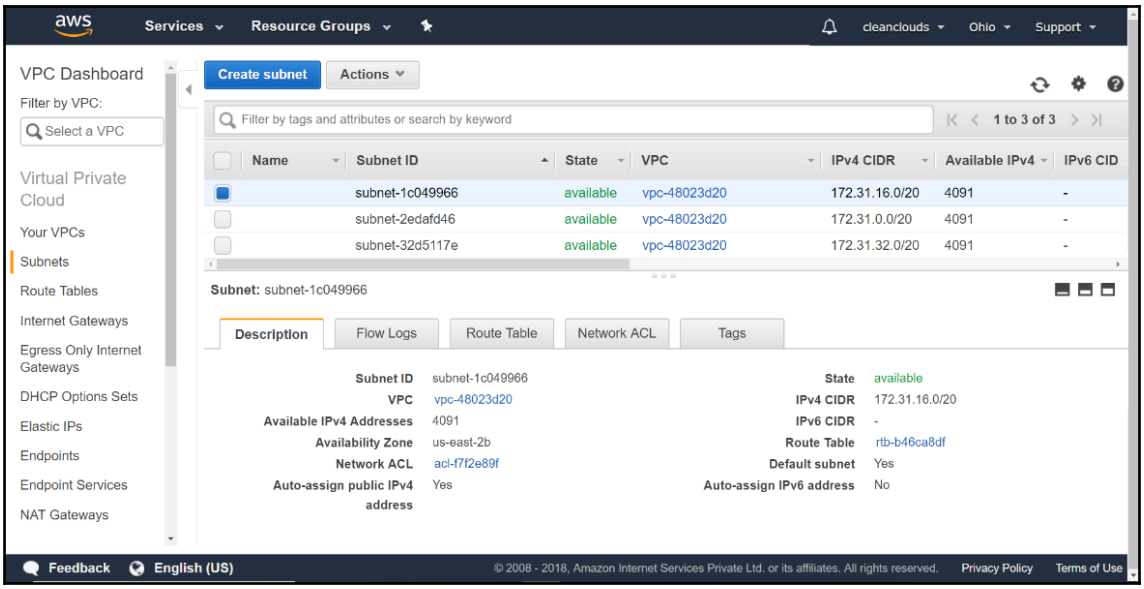
- VPC with a size /16 IPv4 CIDR block (172.31.0.0/16); this means 65,536 private IPv4 addresses
- Default subnet /20 in each Availability Zone; it means 4,096 addresses per subnet
- One internet gateway
- A main route table for default VPC
- Default security group associated with your default VPC
- Default network access control list (ACL)



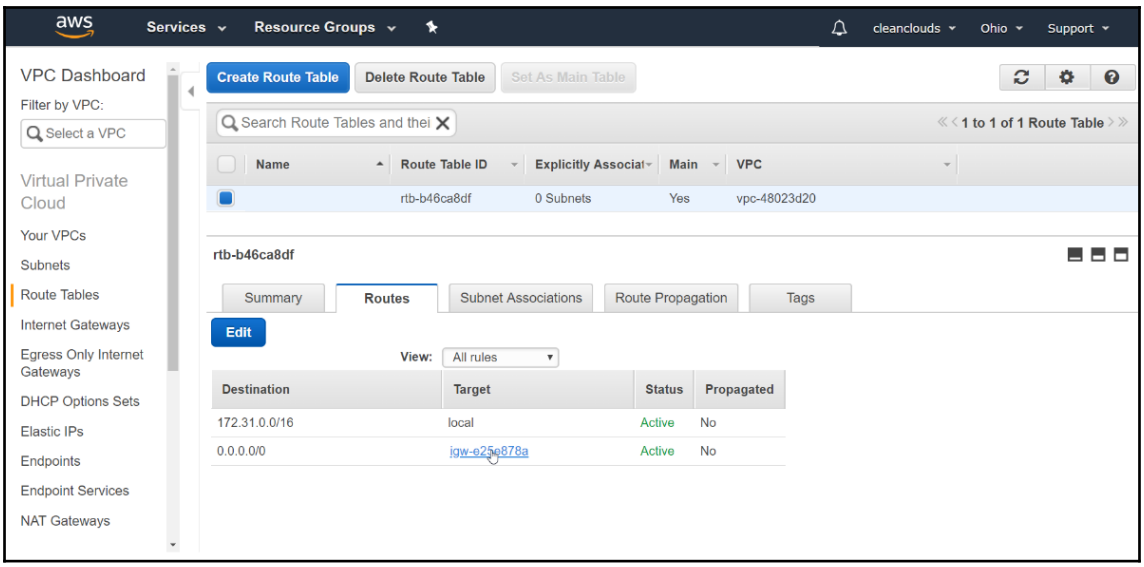
Verify the VPC ID, State, IPv4 CIDR, Route Table, ACL and so on.

A subnet can be defined as a section of a VPC's IP address range where you can place groups of isolated compute resources.

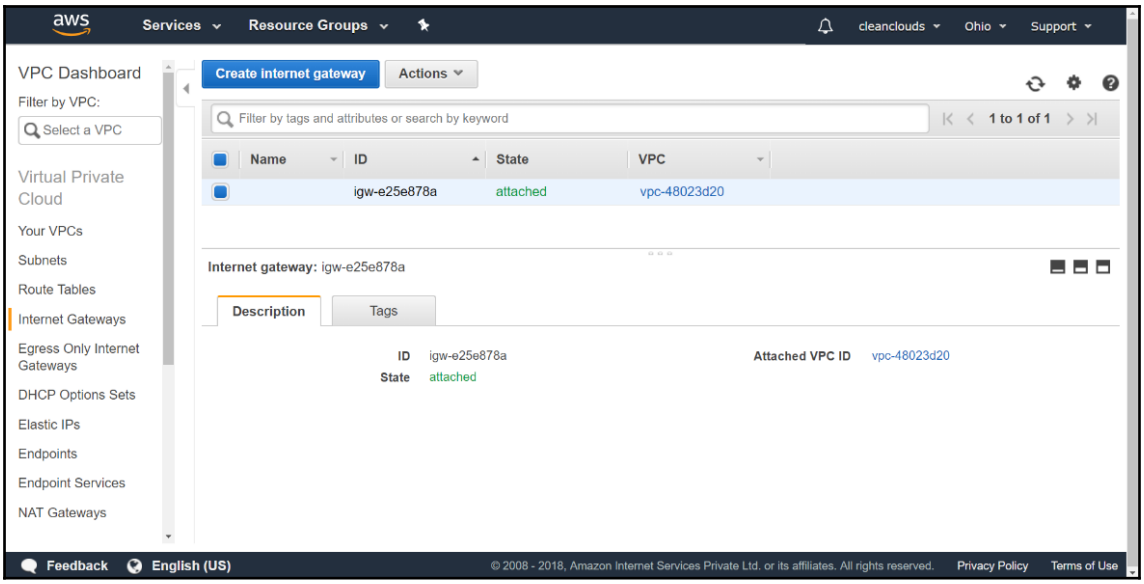
Click on the Subnet link in the left sidebar in the VPC Dashboard:



Route Tables helps to define subnets that need to be routed to the Internet gateway, the virtual private gateway, or other instances:



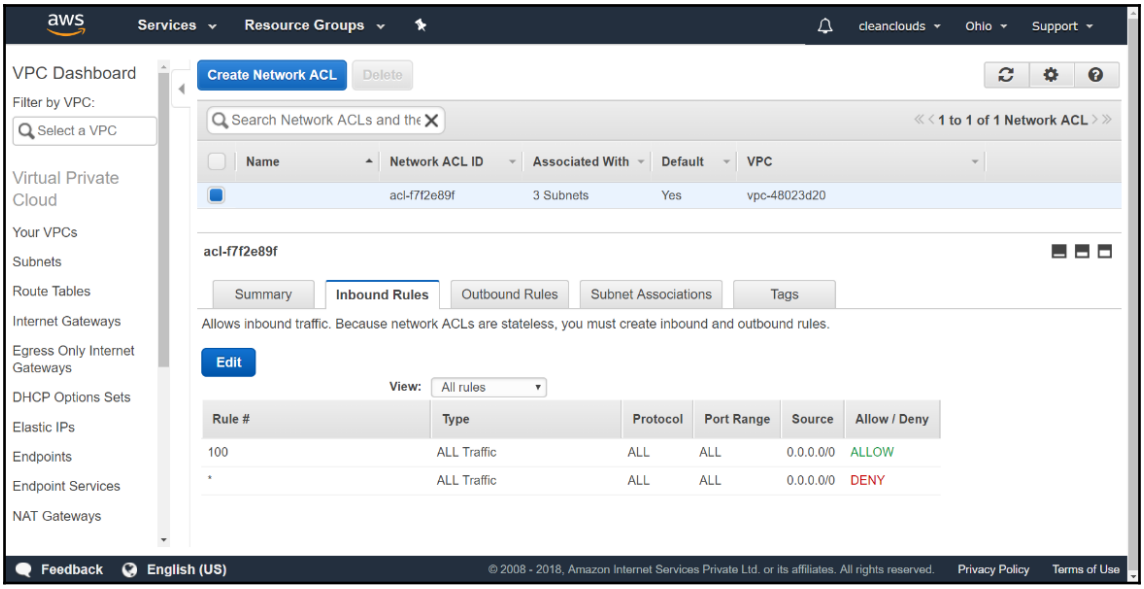
Internet Gateway allows connection to the public Internet from an Amazon VPC:



NAT Gateway represents a highly available and managed **Network Address Translation** (**NAT**) service for resources in a private subnet to access the Internet. NAT Gateway is created in Public subnet.

An Elastic IP address is a public static IPv4 address so you can access the resource.

One Network ACL is created and associated with the default VPC:



The default VPC will have a minimum of two, but depending on the region, could have up to five public subnets, one per availability zone. We'll discuss availability zones in a later section on high availability, but for now what you need to know, is that AWS builds its data centers in small clusters, known as **availability zones** (**AZs**) for short. Each AZ is separated by tens of miles, so if a natural or other disaster should strike, at most only one AZ should be affected.

Every region has at least two availability zones, but the largest regions have up to five. Subnets cannot span availability zones, so your default VPC will have a subnet in each AZ. These will be public subnets, and have the size of `/20`. The VPC will have an internet gateway attached, and a route in the route table to the internet through the gateway. As you can see, this is a very basic VPC, and it doesn't provide private subnets. However, you can customize it and add those if you want.

In the next section, we will demonstrate how to create your own VPC.

Creating a VPC demo

In the previous section, we learned about the difference between classic EC2s and EC2s in a VPC. In this section, we will demonstrate two methods to create your own VPC. The first method will be using the VPC Wizard, which can give us a VPC with private subnets, and even an NAT instance.

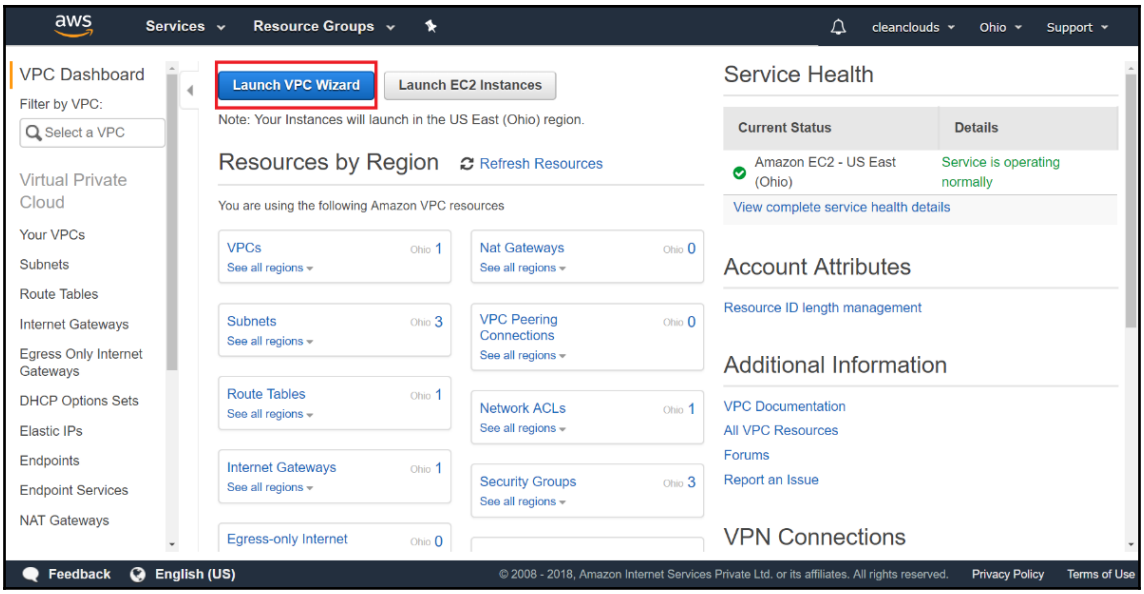
We can create an Amazon VPC in two ways:

1. A VPC the with Wizard:
    - Single Public Subnet
    - Public and Private Subnets
    - Public and Private Subnets and Hardware VPN Access
    - Private Subnet Only and Hardware VPN Access
2. A custom VPC without using the Wizard

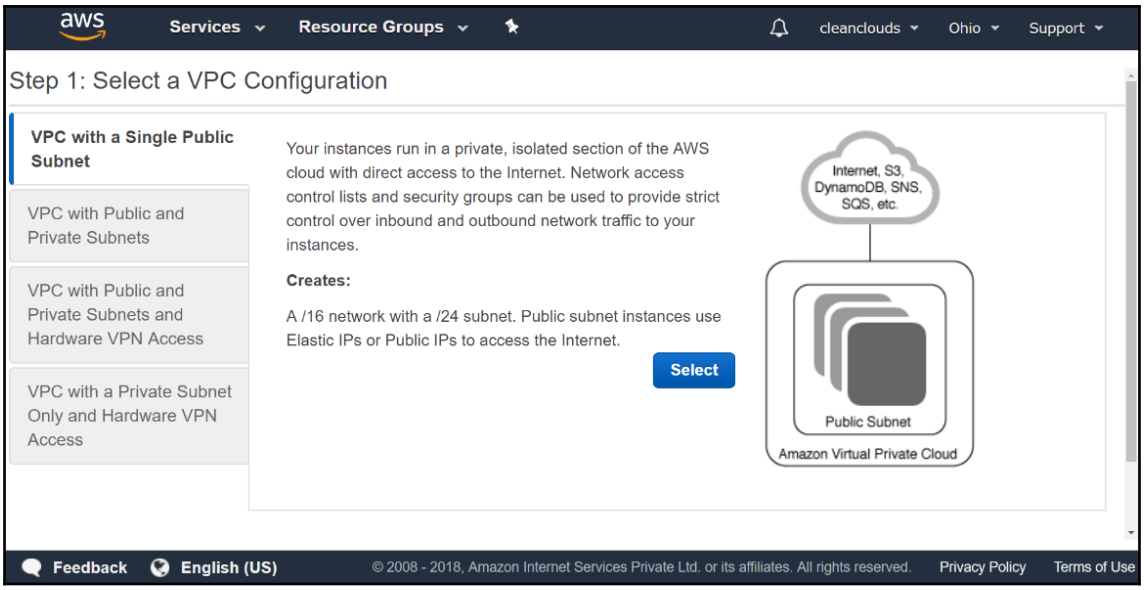
Create VPC using Wizard

Creating a VPC using Wizard is the easiest way to create VPC.

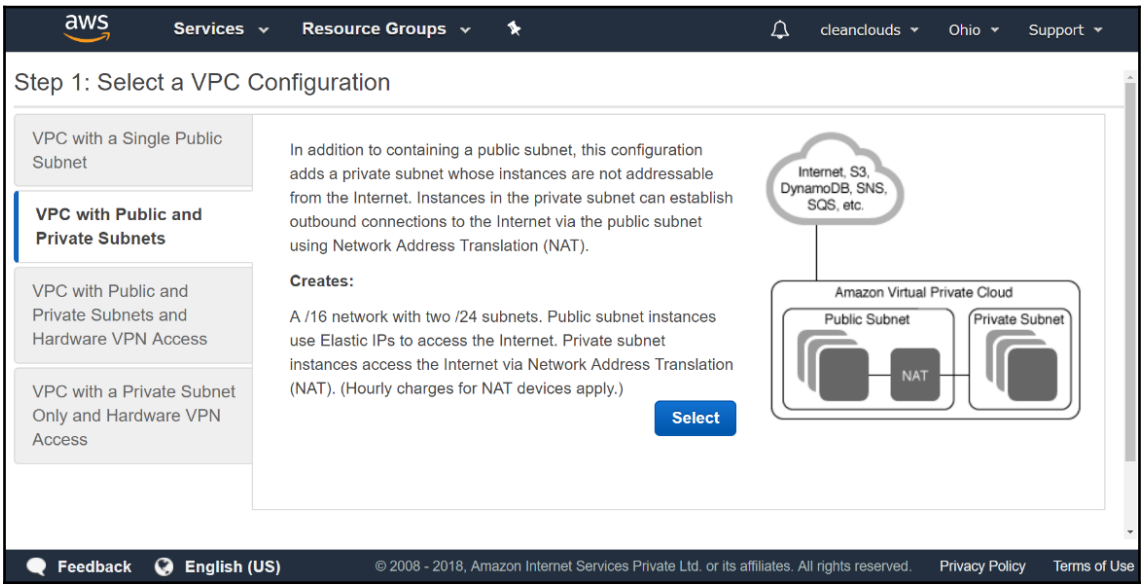
Click on **Services** | Go to **Networking & Content Delivery** section | Click on **VPC** | Click on **Start VPC Wizard** on VPC Dashboard.
To create a new VPC, click on Launch VPC Wizard:



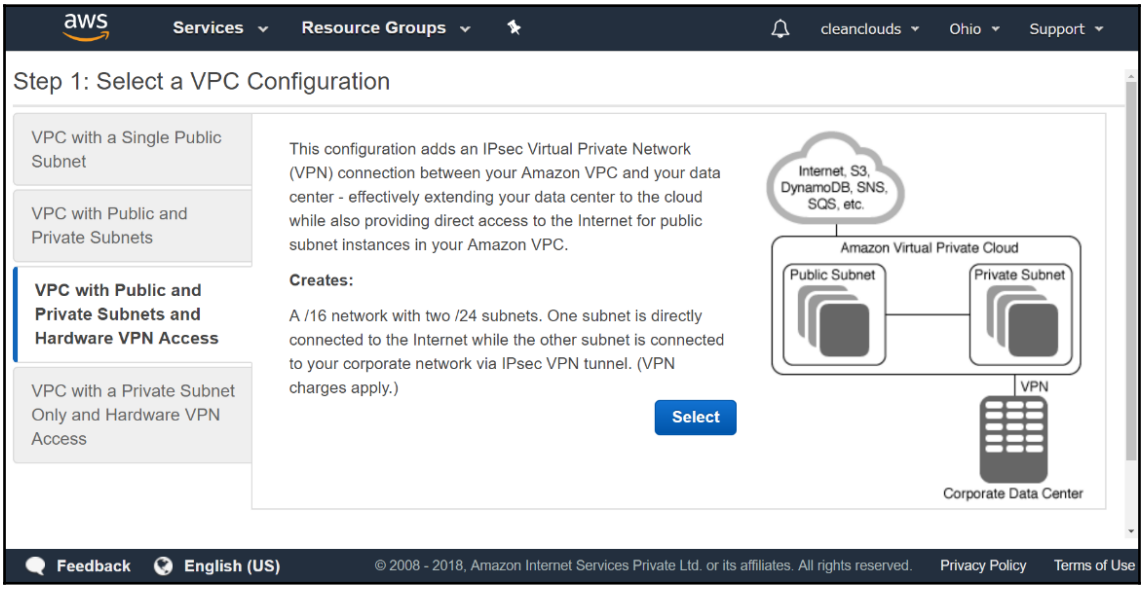
There are four possible options. The first one is a very basic VPC, with a single public subnet:



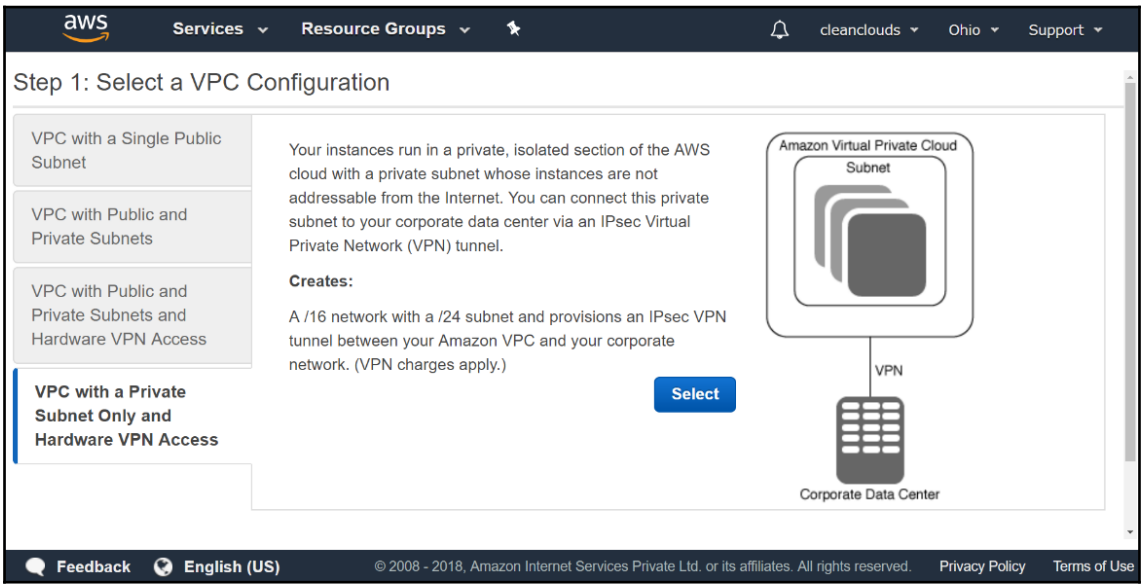
The second one, is a VPC with a public instance, containing an NAT instance or NAT gateway, and a private subnet:



The third one adds a virtual private gateway, which allows you to make a hardware VPN connection, back to your data center:

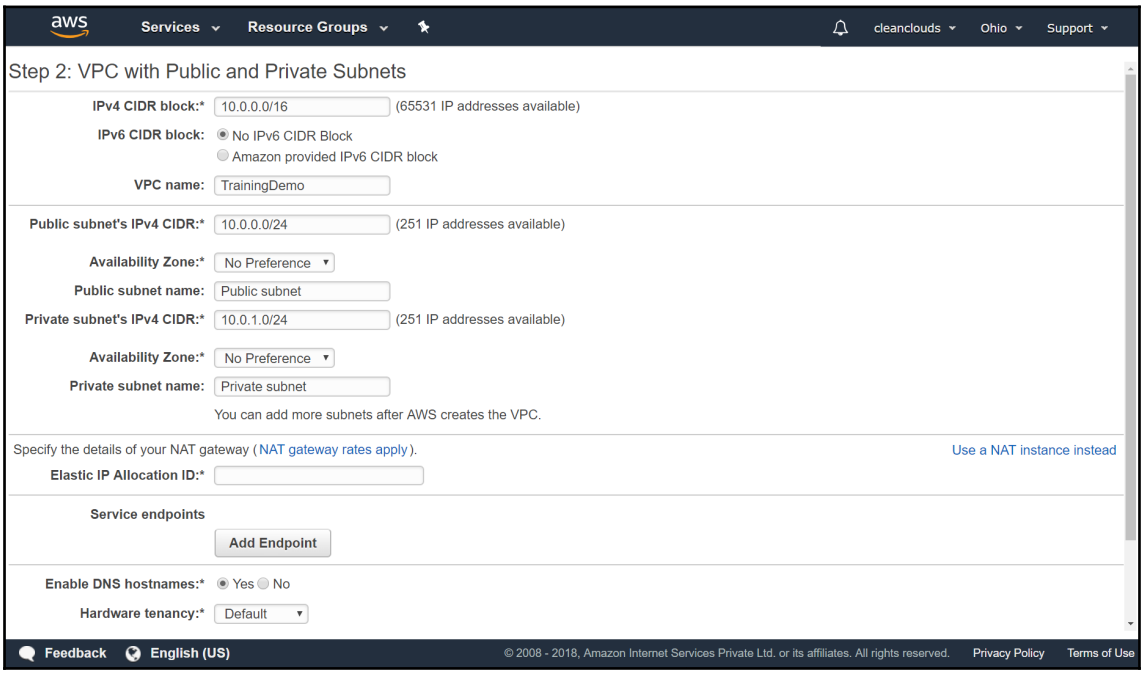


The fourth one takes away the public subnet, and just leaves the private subnet with a hardware VPN connection to your data center:



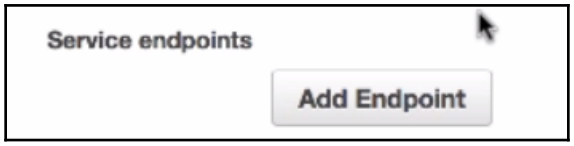
Let's pick the second one for now, which will create a public subnet, a private subnet, and an NAT. For the VPC **IPv4 CIDR block**, let's just leave it as `10.0.0.0/16`, and let's name the VPC `TrainingDemo`.

Let's also leave the public subnet CIDR block as it is, `10.0.0.0/24`. For now, let's not choose an availability zone, and one will be randomly assigned. Let's call the public subnet Public subnet, for the private subnet we will leave it `10.0.1.0/24`, and will call the private subnet Private subnet:

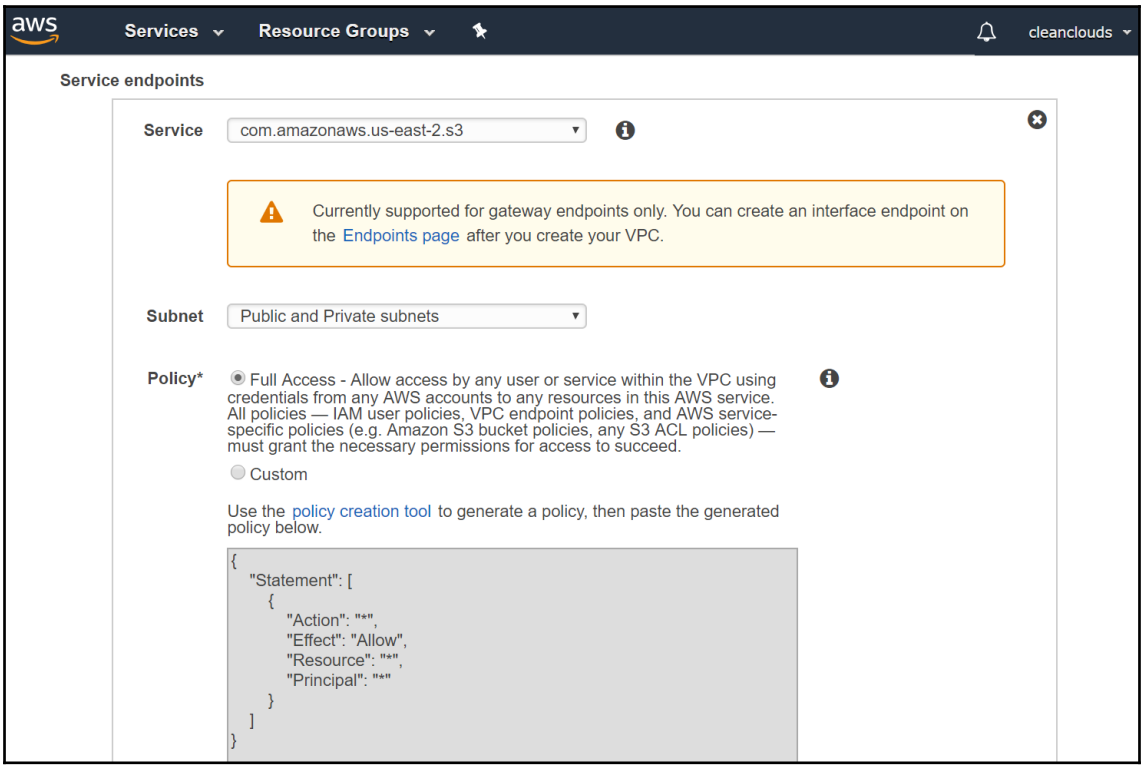


In the next section, we can choose to launch an NAT gateway, or an NAT instance in our public subnet. For this demo, let's launch an NAT instance by clicking on Use an NAT instance instead:

Pick the instance type you would like to launch. Remember, the larger the instance, the higher the network bandwidth. Then, enter the private key pair associated with your account. The next section allows us to add a VPC endpoint for S3:



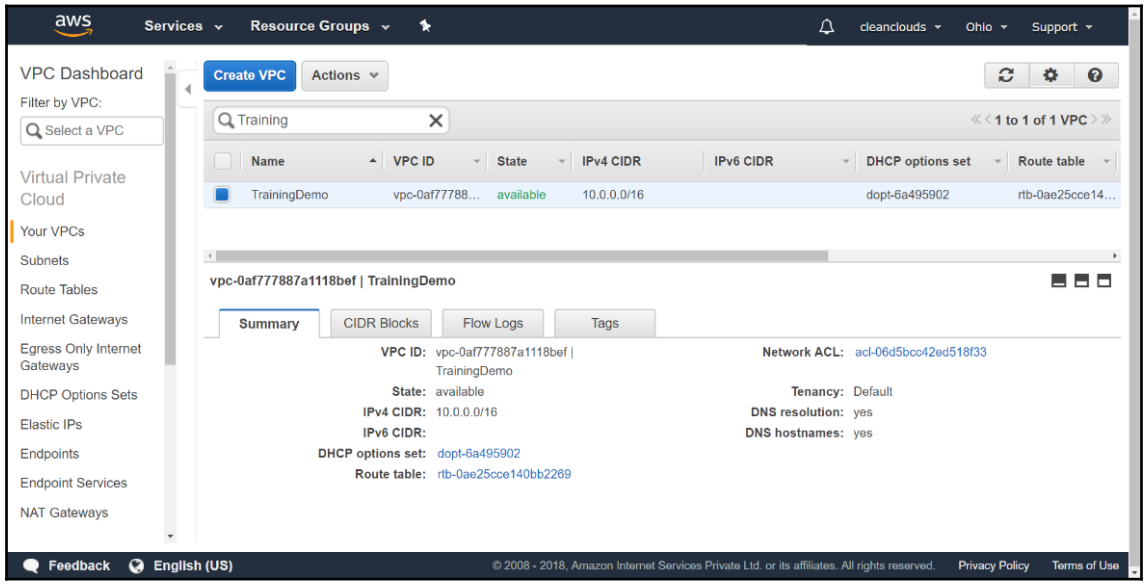
We could leave the default settings after clicking on **Add Endpoint**:



Let's also leave **Enable DNS hostname** on:

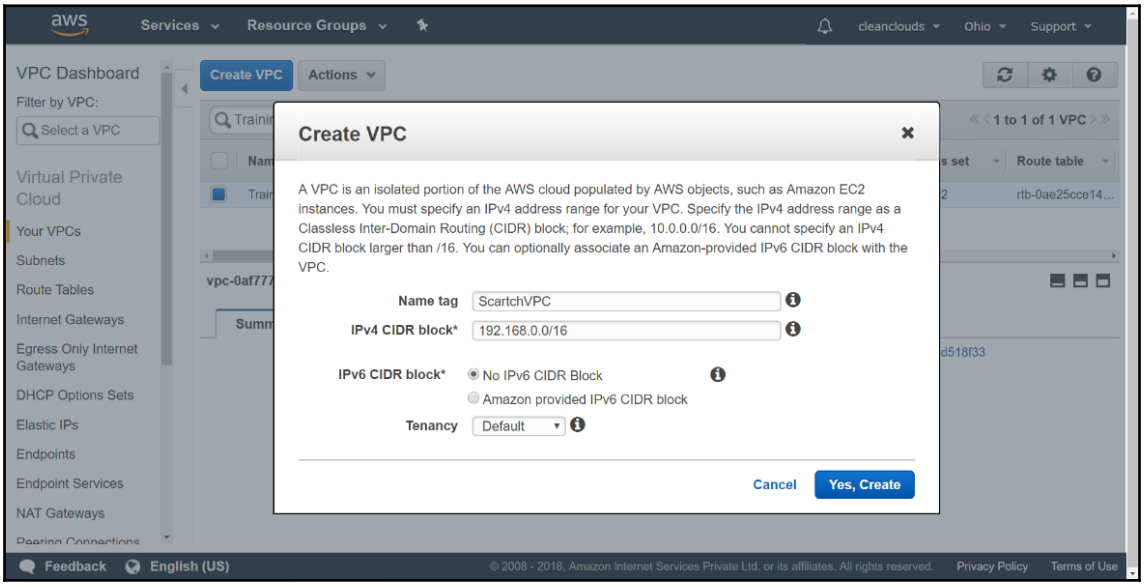
This means our instances with private IP addresses will have a DNS hostname as well. Leave **Hardware tenancy** as **Default**, and then click on **Create VPC**. After a few minutes, your VPC will have been created:

Click on **OK** to see your new VPC:



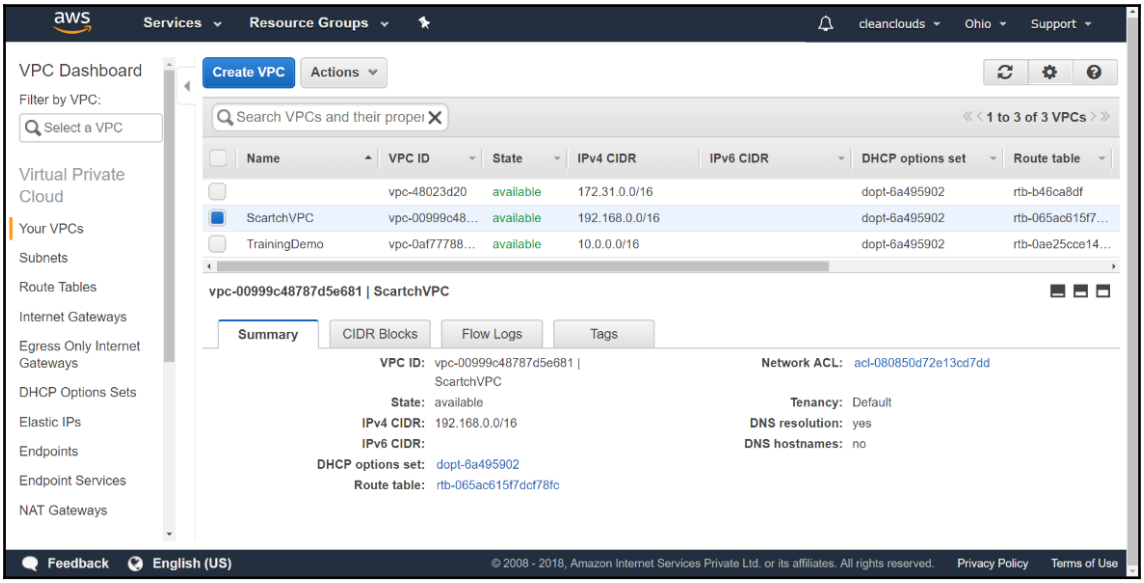
However, to get the most control, you can create a VPC from scratch, without the VPC Wizard.

Click on **Create VPC**. In the popup, give your VPC a name, and enter an IP address block using CIDR notation. `/16` is a typical size, and this gives your VPC `65536` private IP addresses it can use:



You can leave the **Tenancy** option as **Default**, but just so you know what this is, the other option, **Dedicated** tenancy, makes it so that any instance you launch in this VPC will be launched in a host server on which you are the only customer. In other words, with the **Default** tenancy, your EC2 instances get created on host machines that are shared by other AWS customers. But with **Dedicated** tenancy, only your dedicated instances will run on that host machine; no other customers will share it. This is sometimes required for security compliance, or strict software licenses that are not cloud-friendly. Of course, additional hourly charges apply.

Click **Yes, Create**, and you should see your new VPC appear:



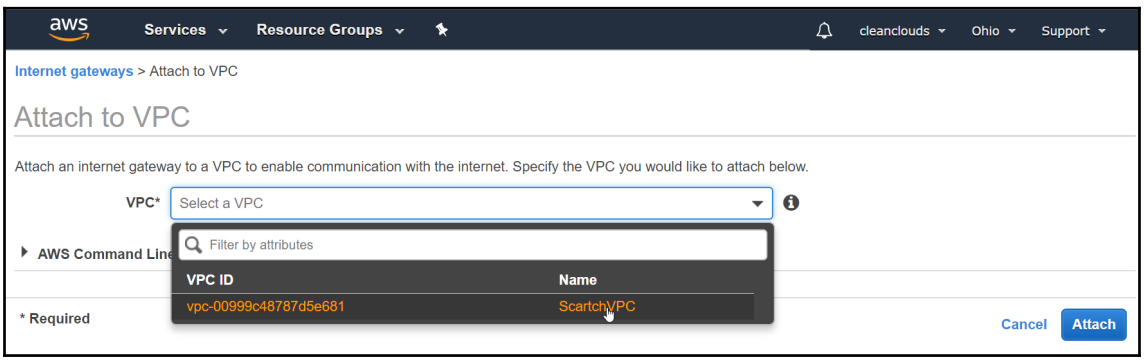
If you want your VPC to have internet access, you need to attach an internet gateway. In the navigation, click **Internet Gateways**, and then **Create Internet Gateway**.

Give it a name, and then click **Yes, Create**:

Click on Close.

The next step is to select the internet gateway, and then click **Actions** | **Attach to VPC**.

Click the VPC you just created, and then click **Attach**:

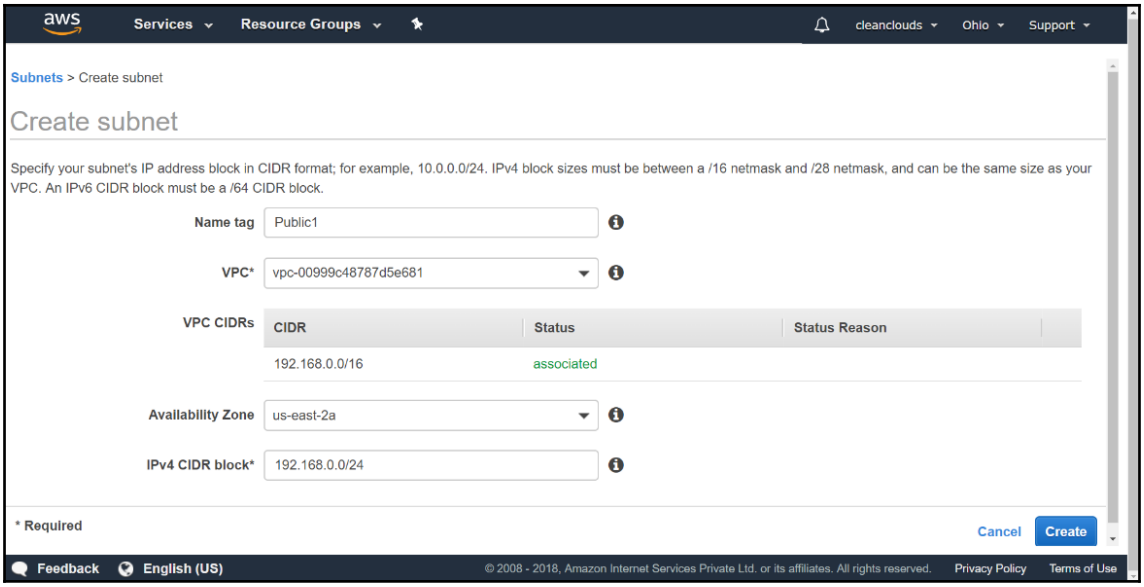


Verify that the newly created Internet Gateway is attached to the VPC we created.

Now, let's create some subnets. In the navigation, click **Subnets**, and then **Create Subnet**.

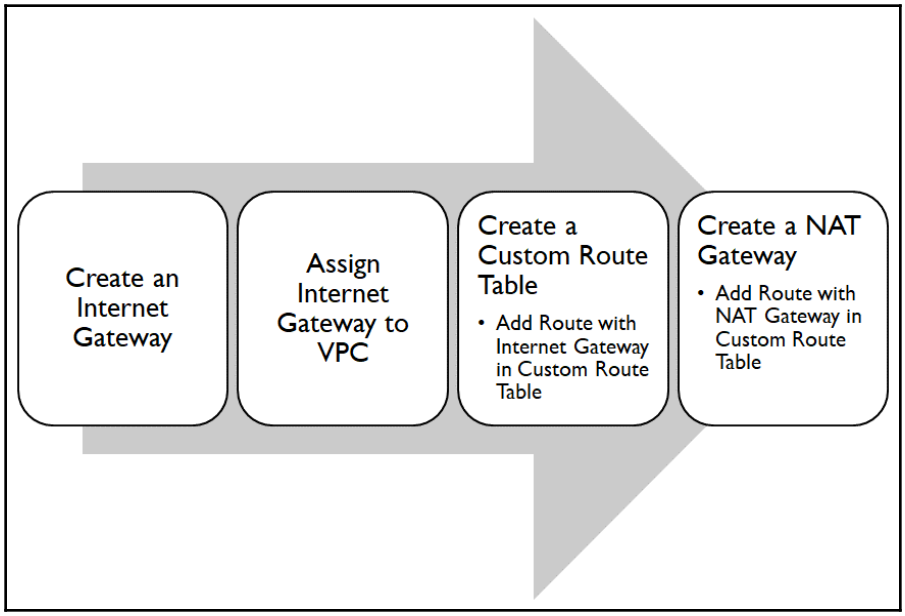
Give it a name. We're going to make this a public subnet, so let's call it `Public1`.

Click the VPC we just created, and select any **Availability Zone**. Enter a **CIDR block**, but remember this must be a subset of the IP address range you gave your VPC. So, let's make this a `/24`, which will give it `251` available addresses:



Remember that five addresses are taken by AWS.

We will configure the following things in the next section to achieve internet access for our instances in the public subnet:

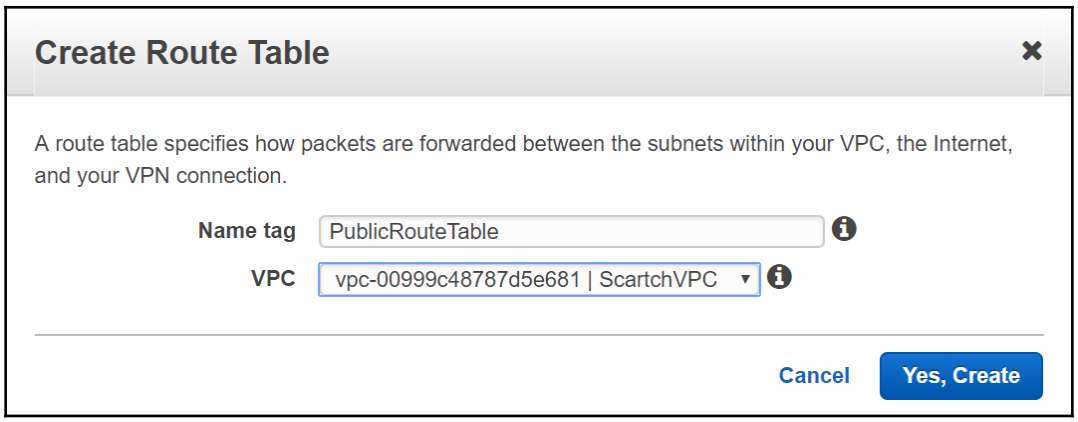


How can we know whether a subnet is Public or Private?
If an Internet Gateway is assigned to the subnet, then it is Public.

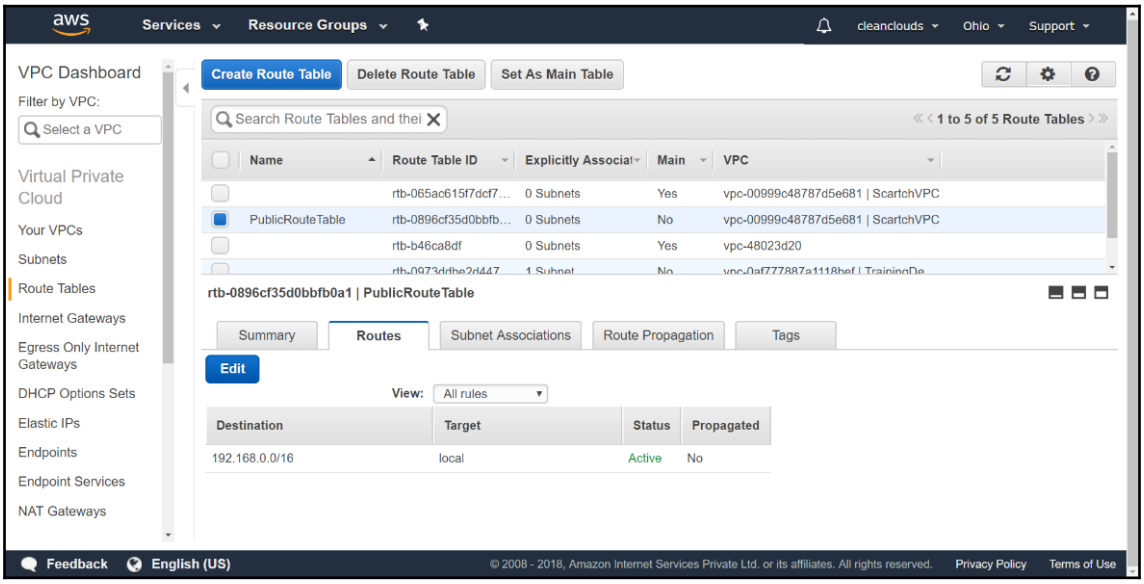
And since we want this to be a public subnet, we have to give our subnet a route to the internet, through the internet gateway.

In the navigation, click **Route Tables** and then **Create Route Table**.

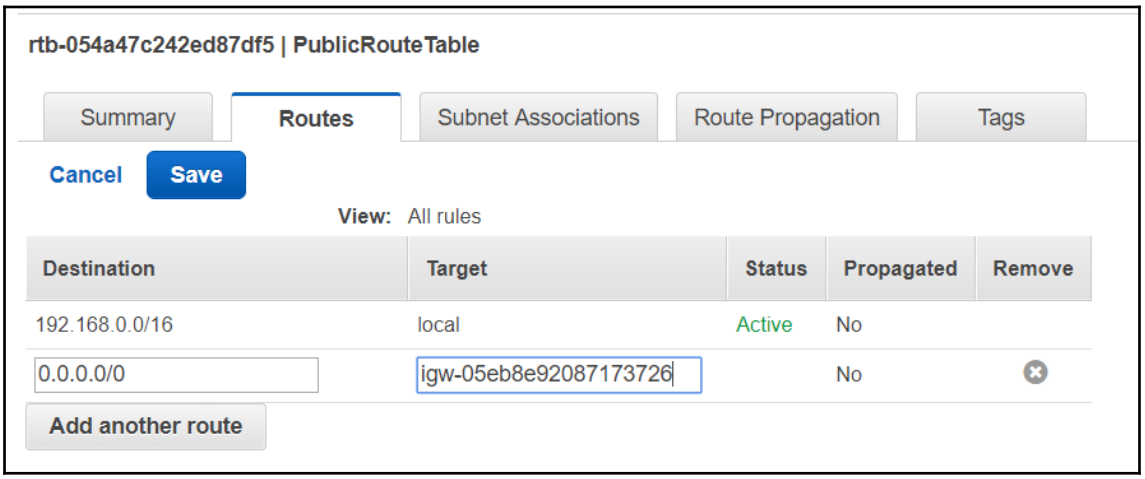
Give it a name, such as`PublicRouteTable`, and select your **VPC**:



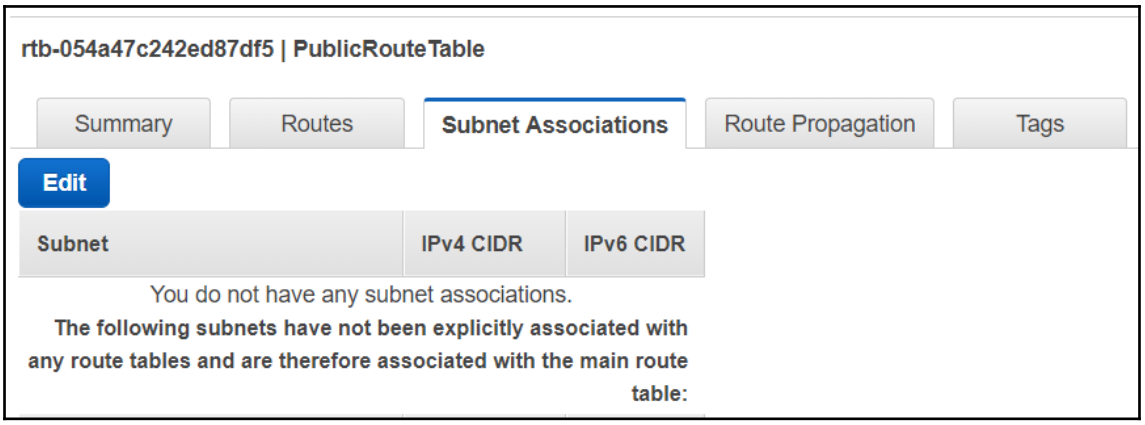
Click **Yes, Create**. Select a new route table, and then on the **Routes** tab, click **Edit** and **Add another route**:



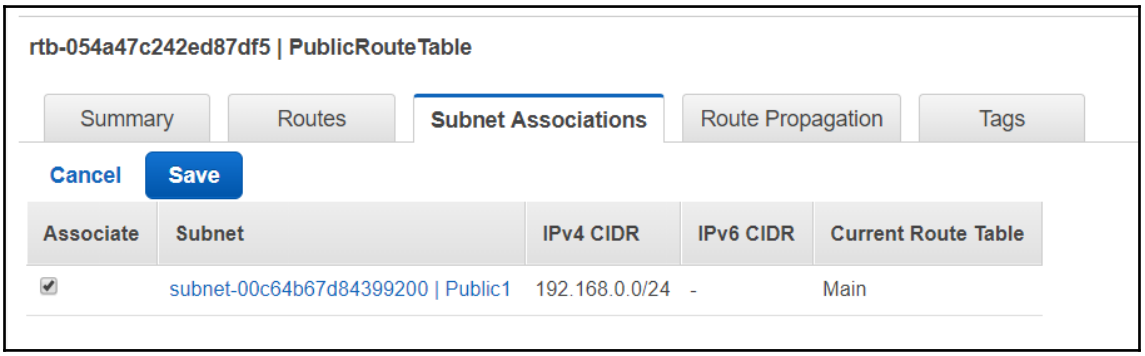
For the **Destination,** type `0.0.0.0/0`, which is the way we specify the entire internet. Click in the **Target** box, and then click the internet gateway we just created and click **Save**:



Now that we've created the route table, let's associate it with our subnet. Click the **Subnet Associations** tab, and then click **Edit**:



Select our public subnet, and click **Save**:



Here, we are talking about accessing the internet from an instance launched in a private subnet.

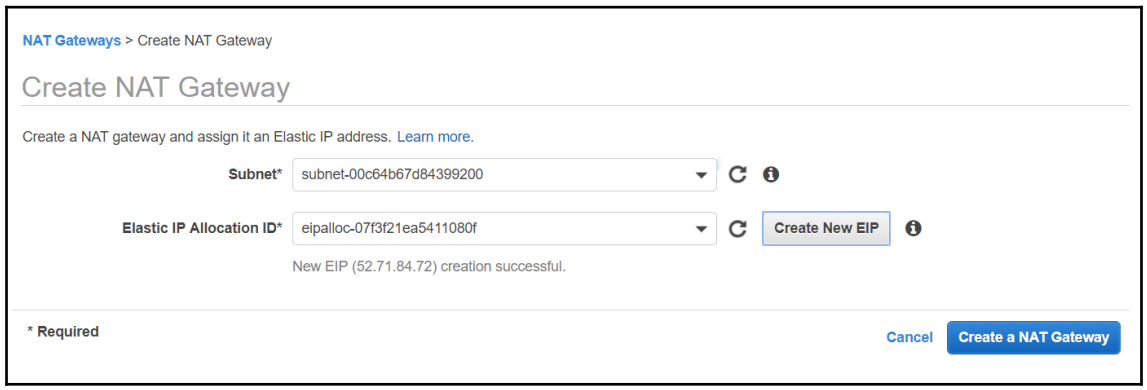
We can achieve this using NAT Devices. There are two ways to achieve this in AWS. One is by creating an NAT instance and another is by creating an NAT Gateway.

Add NAT Device in the Public Subnet. Why? Only then it will be able to access the internet, right?

We will use an NAT Gateway here to demonstrate the configuration of internet access for instances available in the Private subnet.

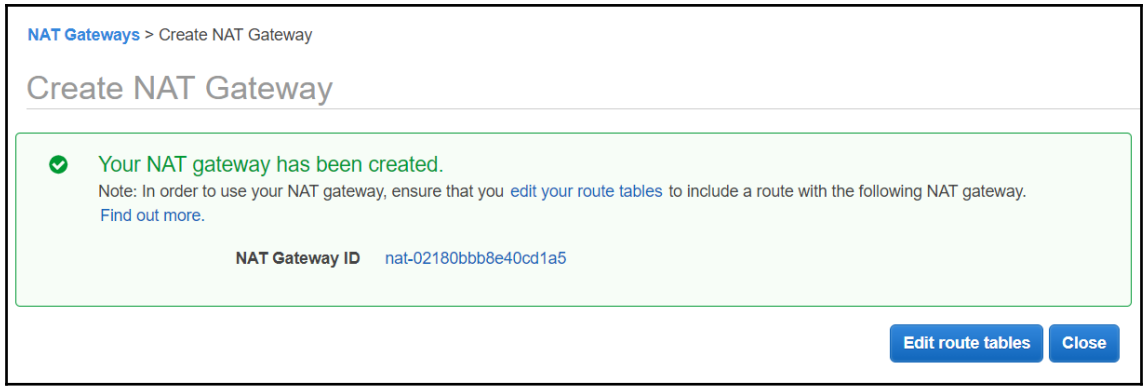
Now, let's put an NAT gateway in our new subnet. In the navigation menu, click on **NAT Gateways**, and then **Create an NAT Gateway**.

Click in the **Subnet** box, and then select our public subnet. Click on **Create New EIP**, so our gateway will have an elastic IP, and can access the internet:

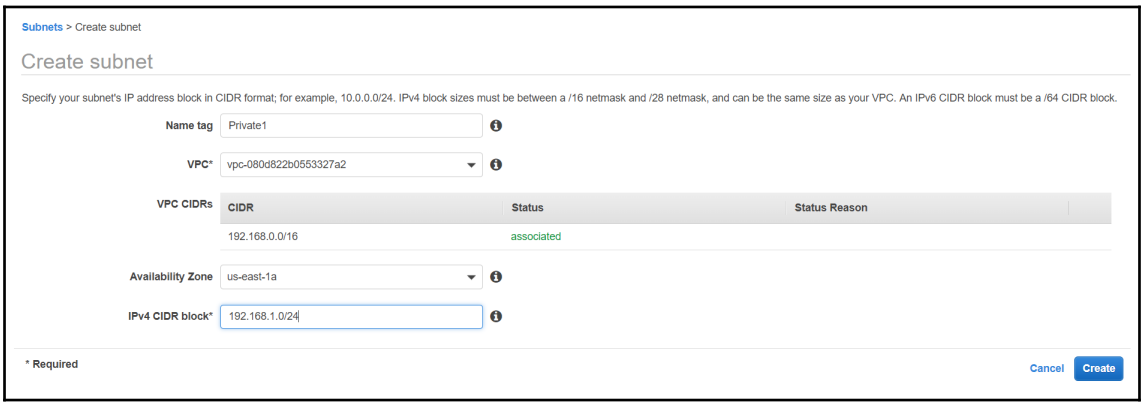


Finally, click **Create an NAT Gateway**.

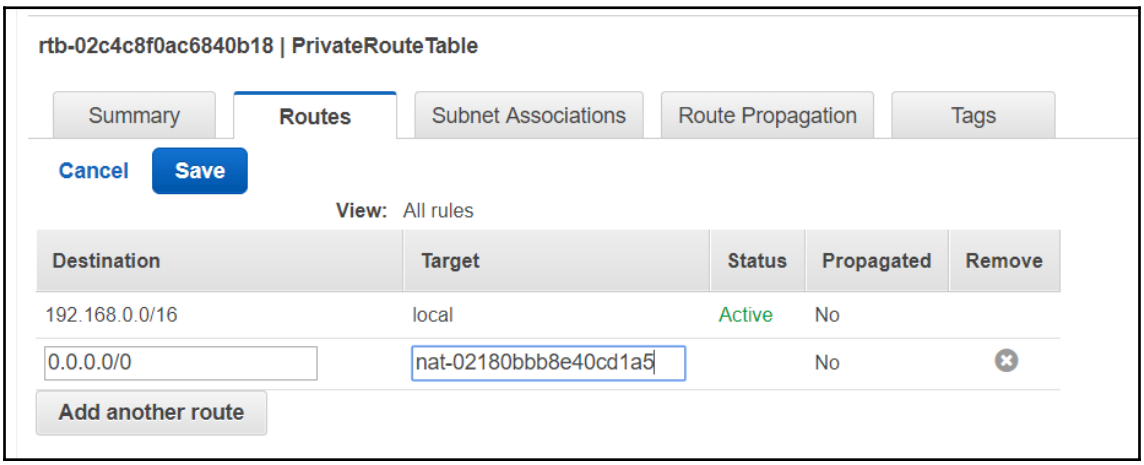
You will get the following pop-up window:



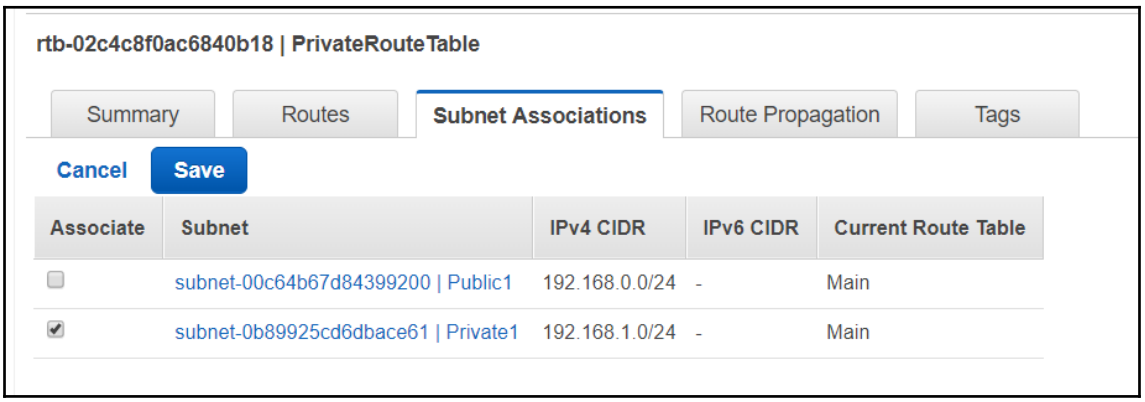
Now, let's create a private subnet, and then give it a route through the NAT gateway to access the internet and public AWS services. Back in the navigation, click **Subnets** and then **Create subnet**. Give it a name, let's call it `Private1`, click the **VPC** we just created, and select any **Availability Zone**. Enter an **IPv4 CIDR block** that won't overlap with our public subnet, and click **Yes, Create**:



Now, let's create a route table for our private subnet. In the navigation menu, click **Route Tables** and then **Create Route Table**. Give it a name, this time `PrivateRouteTable`, and select your **VPC**. Click **Yes, Create**. Select the new route table, then the **Routes** tab, and then click **Edit | Add another route**. For the **Destination,** type `0.0.0.0/0`, and in the **Target** box, click the NAT gateway we just created and click **Save**:



Finally, let's associate that route table with our private subnet. So, click on the **Subnet Associations** tab and then **Edit**. Select our private subnet, and then click **Save**:



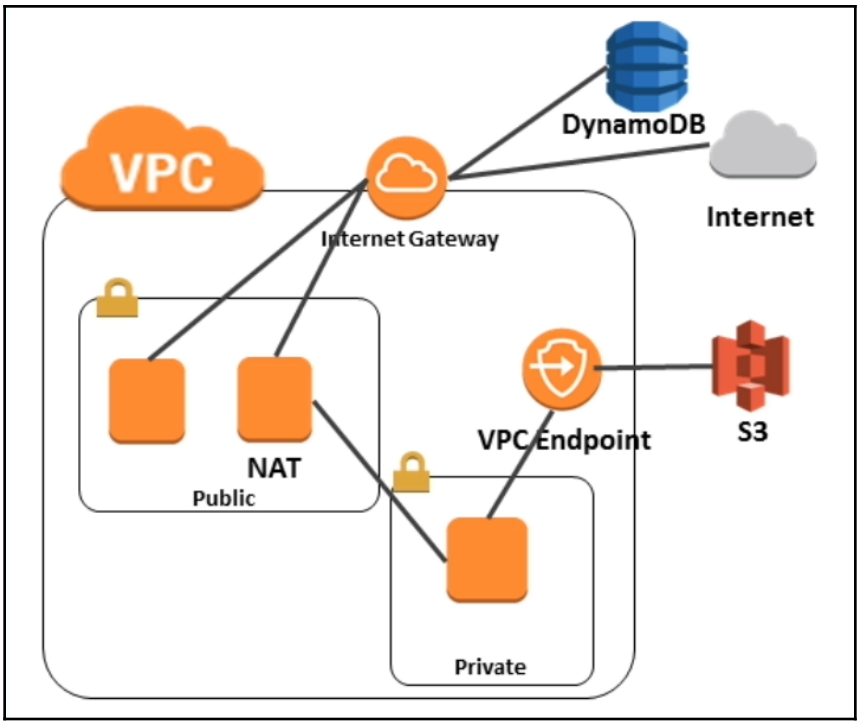
You can keep creating subnets if you like. In a later section, we will show you how to take advantage of multiple availability zones to make your application environment highly available. In the next section, we will discuss several ways to connect to your VPCs, including VPNs and direct connect.

Connecting to a VPC

In the previous section, we demonstrated methods for creating a VPC. In this section, we're going to discuss what you need to do to securely connect to your VPC. We're going to look at the two types of gateways, the internet gateway and a virtual private gateway. Then, we're going to discuss making a hardware VPN connection from your data center to a virtual private gateway, or using a traditional software VPN. Next, we'll briefly discuss connecting over a private dedicated line using direct connect. Finally, we'll talk about connecting two VPCs together using peering. In our VPC example shown earlier, we created an internet gateway and attached it to our VPC.

Internet gateway

An internet gateway is a service managed by AWS that connects the internet to your VPC. It is highly available, and scales to meet your traffic requirements:

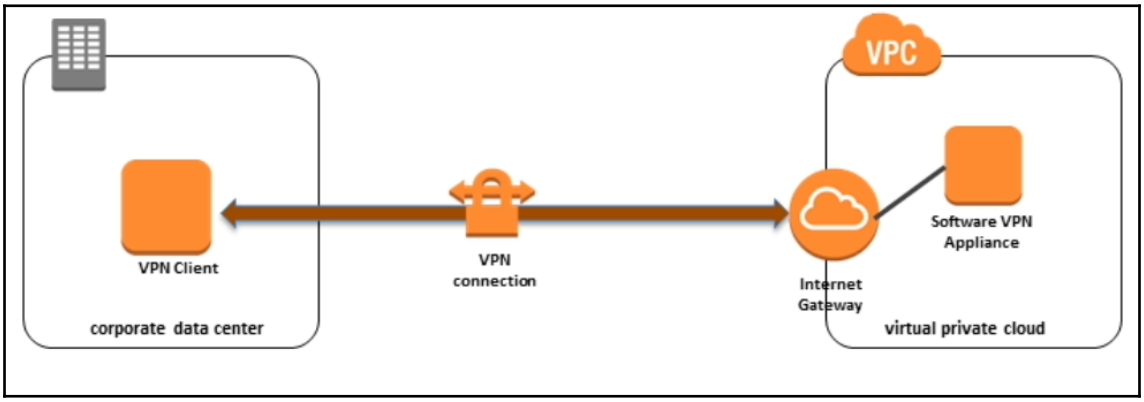


For an instance to be directly reachable from the internet, it needs to have a public or elastic IP, and it needs to be launched in a subnet that has a route to the internet, through the internet gateway. You may be surprised to learn that many AWS managed services, such as DynamoDB, Kinesis, SQS, and SNS, are not reachable from private subnets in your VPC. Only instances in a public subnet, with public or elastic IPs, can reach these services, because communication with them is over the public network, which means it has to pass the internet gateway. Instances in a private subnet can only reach these services by routing the requests through an NAT instance or NAT gateway.

Since NAT instances can sometimes become a bottleneck, AWS has started introducing VPC endpoints for some of their public services. You add a VPC endpoint inside of your VPC, and then you can communicate with the service as if it is inside your VPC. So, private instances can communicate directly with the service without an NAT. The first service to support VPC endpoints is S3, an object storage service that we will cover in a later section.

Software VPN

We can communicate with our instances from the outside by using their public IP addresses. However, if you wanted a secure way to extend your private network, you could use a software-based virtual private network (VPN) appliance. This gives you a secure encrypted connection to your VPC:

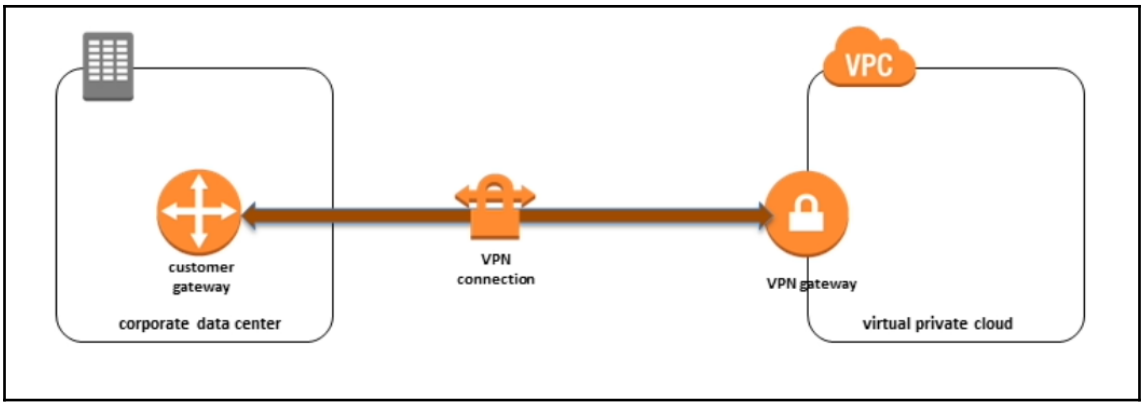


You install the appliance, by launching an AMI from the AWS marketplace, which is provided by third-party vendor such as OpenVPN.

Virtual gateway

AWS directly supports another type of VPN connection, known as a hardware VPN.

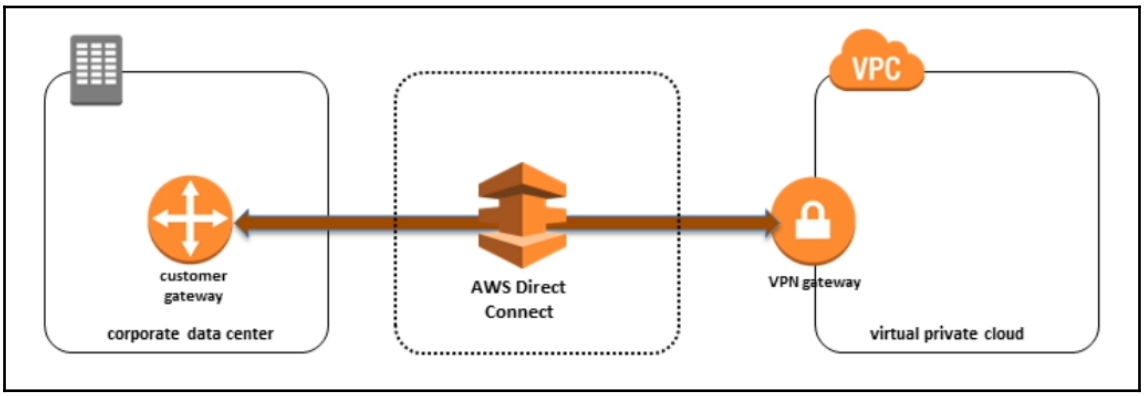
For this, you attach a virtual private gateway to your VPC and install a supported customer gateway in your data center:



Like the software VPN, traffic is encrypted with IPsec tunneling. Both hardware and software VPNs use the public internet for the transport layer.

Direct connect

A third option, direct connect, does not use the public internet. Instead, you use a dedicated fiber optic connection between your data center and a direct connect facility. You work with a service partner, such as Equinix or AT&T, to make the fiber connection and then make it cross connect to the AWS network in the direct connect location:

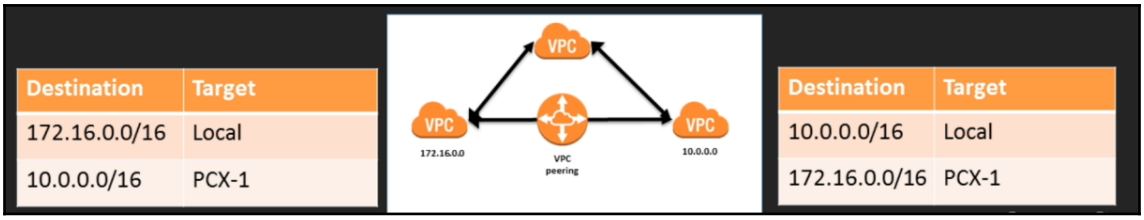


A direct connect location is associated with a particular AWS region.

The benefits of direct connect include speeds available up to 10 GBps, improved performance, and enhanced security and compliance.

VPC peering

If you wanted to connect two VPCs, AWS offers VPC peering, which allows instances in two VPCs to communicate, as if they're in the same network:



Both VPCs must be in the same AWS region, but they do not have to share the same AWS account. For security purposes, peer VPCs cannot share other connections, such as gateways or other peer connections. If two VPCs are peered with the same VPC, they still cannot communicate with one another, unless you peer them directly. In the next section, we will secure our VPC using network access control lists, and Bastion instances.

Securing your VPC

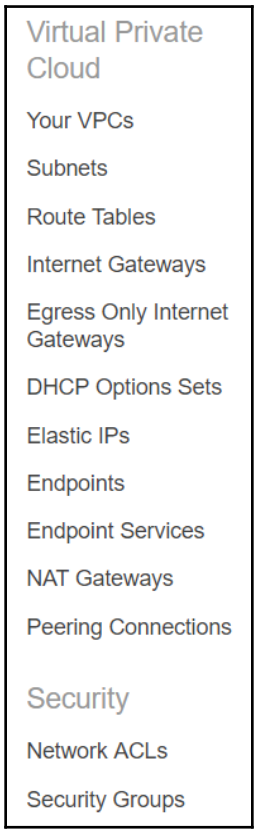
In the previous section, we looked at ways to connect to your VPC, which included gateways, VPN connections, direct connect, and peering. In this section, we're going to add some additional security to your VPC, by adding network access control lists to our subnets. We're also going to talk more about private subnets, and how administrators can still connect to private instances, by using Bastion instances. In an earlier chapter, we talked about security groups, and how these are like firewalls that protect our instances. An additional type of firewall we can use is network access control lists, or just network ACLs.

NACLs

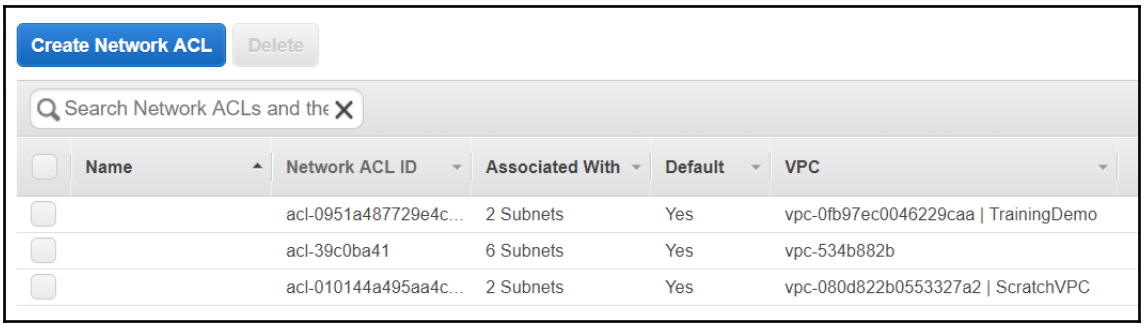
While security groups surround our instances, network ACLs allow and deny traffic at the subnet boundary, both inbound and outbound.

Since we already have security groups, it may seem that network ACLs are a bit redundant. However, best practice is to back up critical firewall rules, by including them in both security groups and network ACLs. By default, every subnet already has a network ACL, but they're configured with just one rule, allow all traffic. So technically, you could consider adding any other rules to them, to deny traffic as optional. Or you can just rely on your security group rules. While this might be okay for low-security environments, consider what would happen if someone misconfigures a security group. It opens up access to your database from the internet. If you have a deny rule in the network ACL, then no harm would be done, as this would block that traffic.

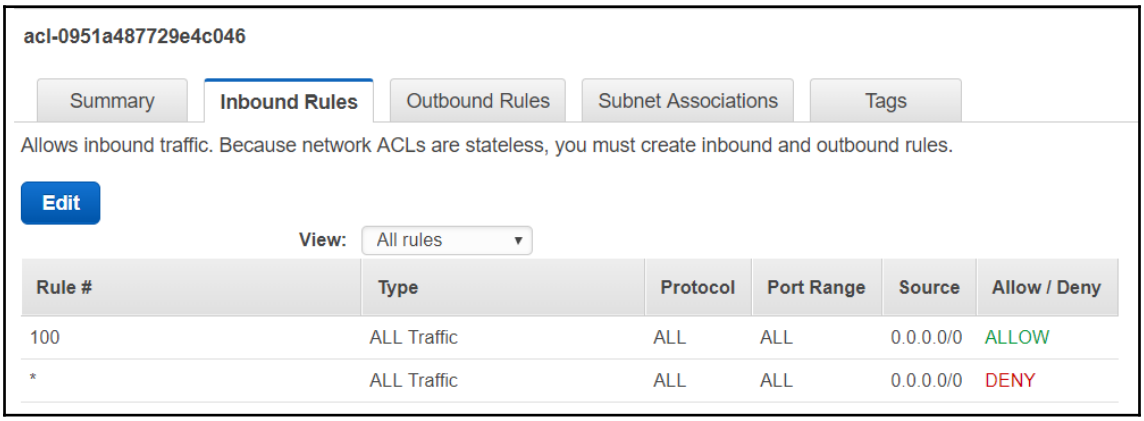
So let's go to the management console and take a look at some network ACLs. Click on **VPC**, and then **Network ACLs**:



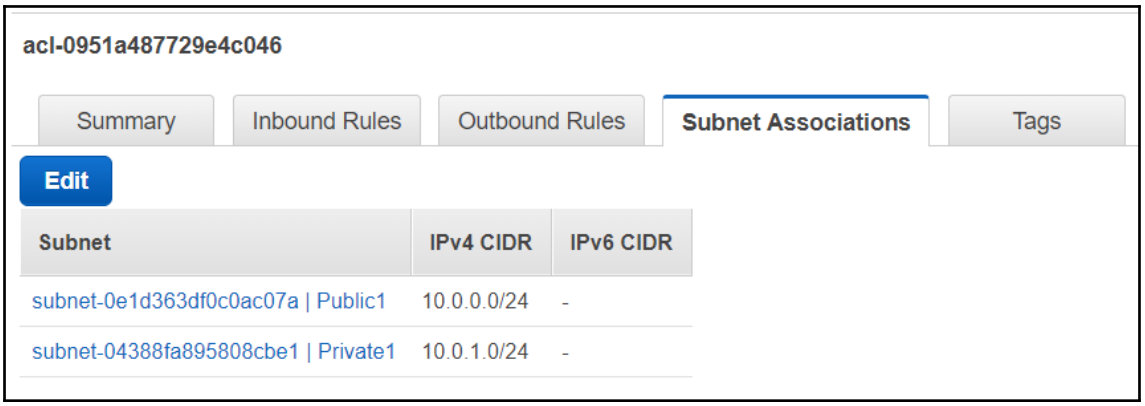
In the list, you will see the ones that are already created for our subnets:



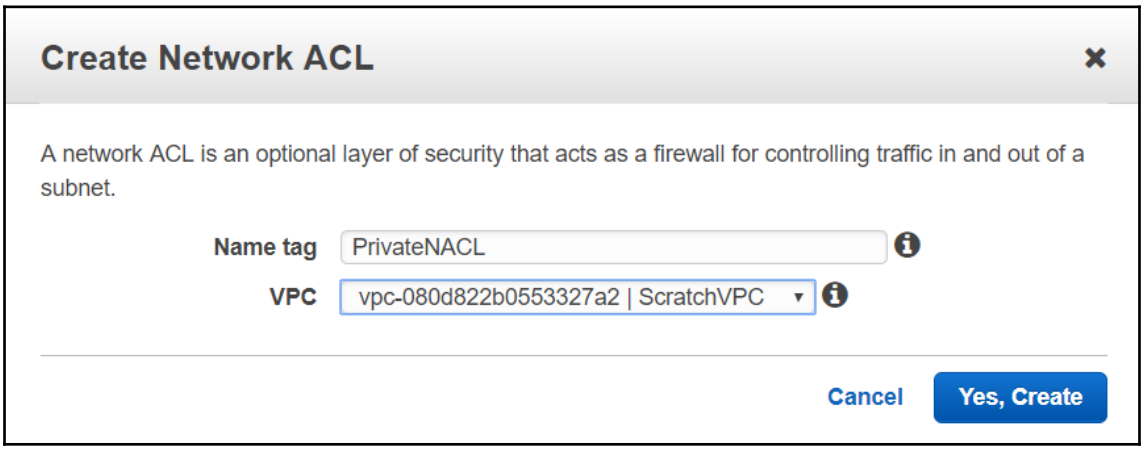
Click on the one for our VPC. Take a look here at the **Inbound Rules** and **Outbound Rules**. You will notice there is an **ALLOW** for all traffic, above a **DENY** for all traffic:



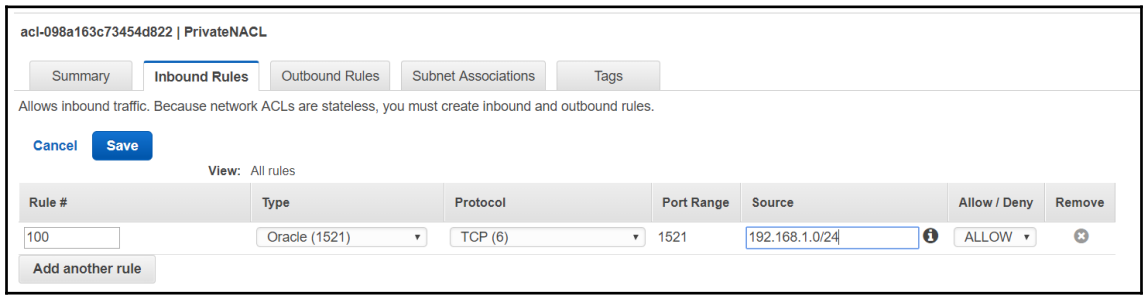
Network ACL rules are processed in order, according to the rule number. Since the **ALLOW** comes before the **DENY**, the **ALLOW** has precedence. Click on the **Subnet Associations** tab, and you will see that the same network ACL is associated with both our private and public subnet:



So let's go ahead and make our private subnet more secure. Click on **Create Network ACL**. Let's give it a name, and select our **VPC**:

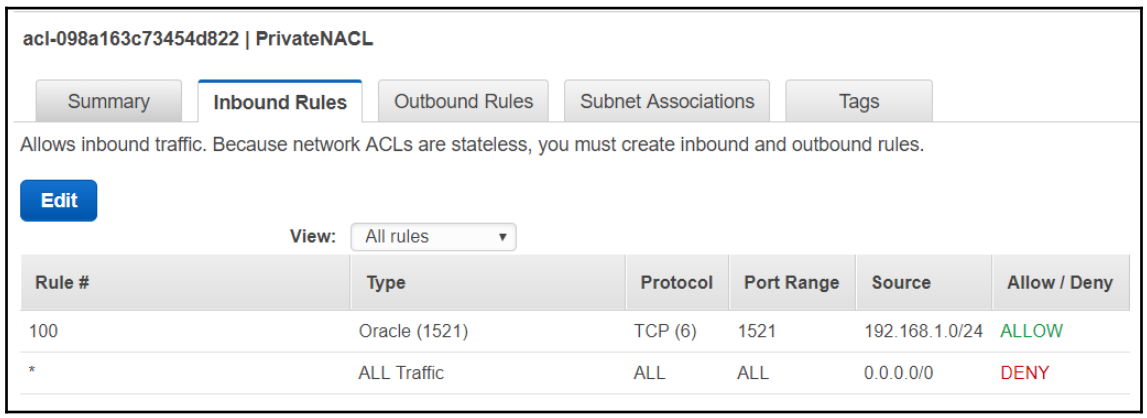


Let's assume that this is in a subnet, we will put in some Oracle databases. So let's add a rule, to allow that traffic from our public subnet, and deny everything else. Click the **Edit** menu in the **Inbound Rules** tab, and then **Add another rule**:



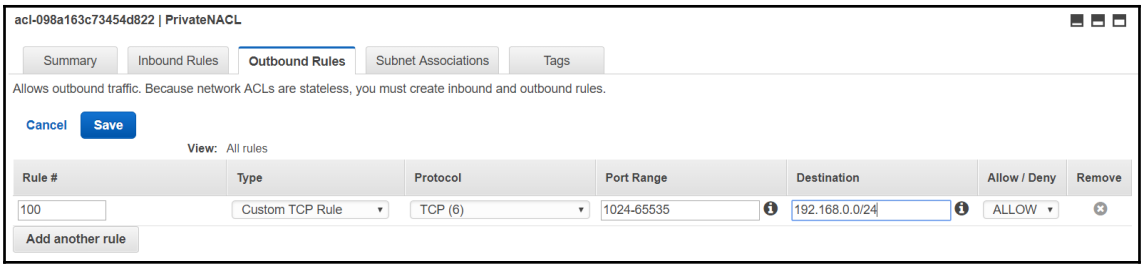
Enter a rule number, such as `100`, and then select **Oracle (1521)**, for the protocol. For the source, enter the IP address for our public subnet. If you're not sure of the IP, go back to the subnets list, and get the IP address for the public subnet. Then, click **Save**.

Note that this will allow inbound traffic on port `1521`, from our public subnet only:

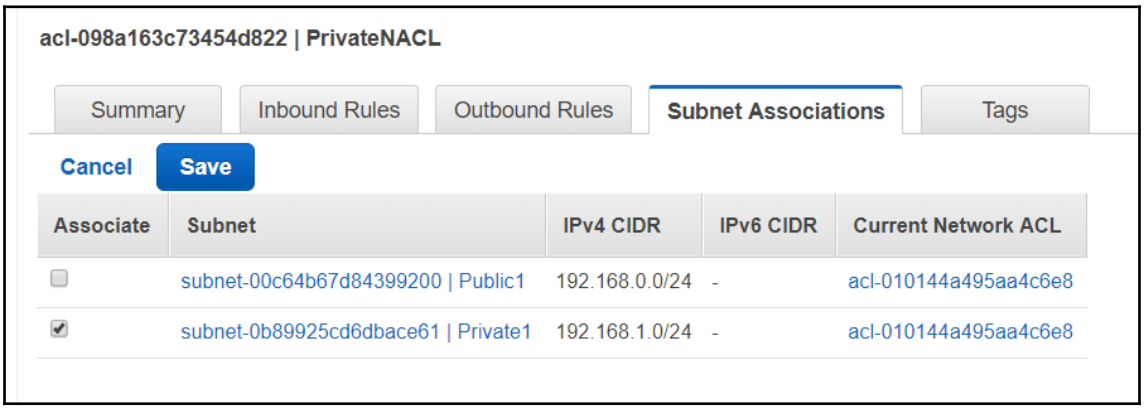


All other traffic will be denied. Unlike security groups, network ACLs are stateless, so we have to configure rules for outbound as well.

For **Outbound Rules**, we need to configure the high ephemeral port range. Click on **Edit | Add another rule**, the port range we should use is `1024-65535`. For the **Destination**, put the IP address of the public subnet, and click **Save**:



Now let's go to the **Subnet Associations** tab, click on **Edit** and add our private subnet:



Now this NACL has replaced a less secure one, that was being shared by the public subnet.

Bastion instances

To follow best security practices, we need to secure our vulnerable instances, such as backend app servers, and databases in private subnets. In a previous section, we mentioned that NAT instances, or an NAT gateway, can be used to give our private instances access to the internet, to download security patches, or to access public AWS services. However, our server administrators will need to be able to connect with our private instances, in order to perform upgrades or security updates. Since there is no direct route to the instances from the internet, we need to launch a Bastion instance, in a public subnet. The administrators can connect to this instance, and from there are able to log in to the private instances. The process is different for connecting to private Linux and Windows instances.

For Linux instances, we launch a Linux instance as a Bastion, and connect to it using SSH, with key forwarding. Once logged in, we can SSH to the private instances. For Windows instances, we launch a Windows instance in a public subnet, and install RD Gateway. With RD Gateway installed, we use our RDP client, and authenticate on the RD Gateway, which then makes an RDP connection to our private instance.

Highly available architectures

In the previous section, we created a secure environment for our applications, by adding additional security to our VPCs. In this section, we will discuss VPC architectures, that can make the environments for our applications highly available. To make our environments highly available, we are going to deploy instances into multiple availability zones, and load balance the requests with elastic load balancing. Finally, we'll add additional resilience to our environment, by automatically scaling horizontally.

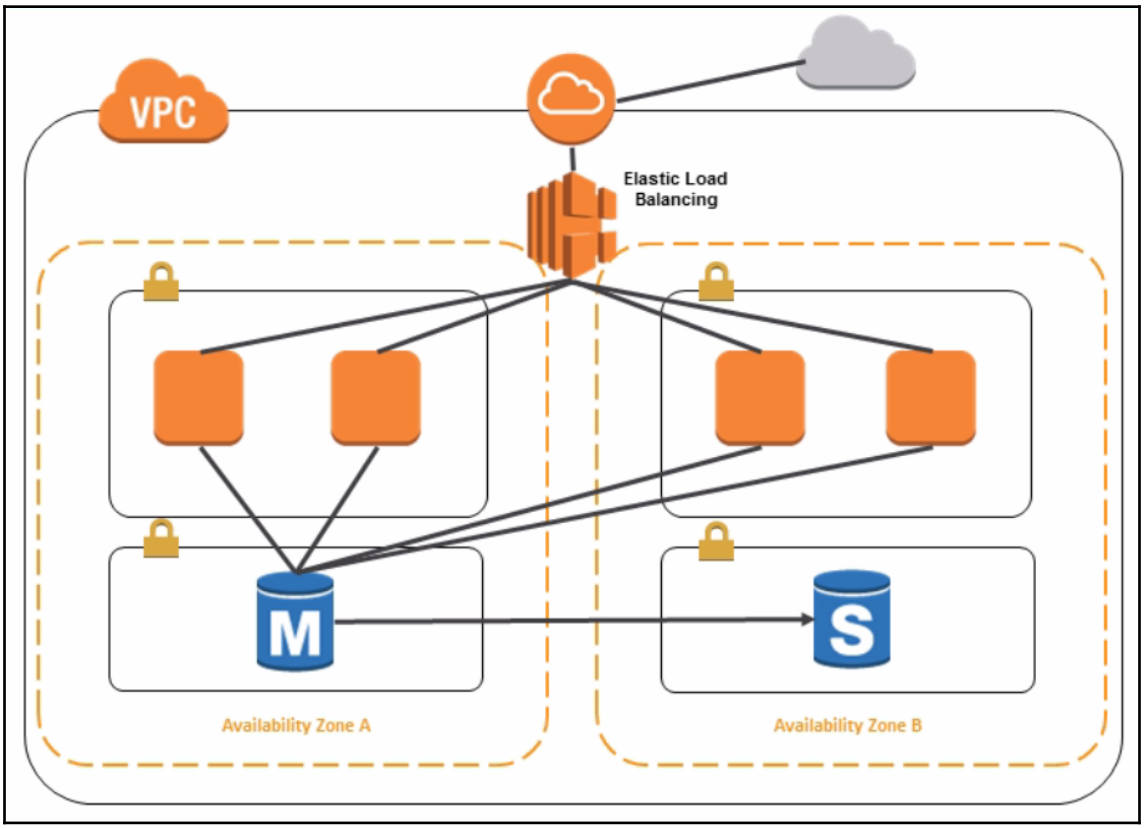
Availability zones

AWS is built on a global infrastructure, located in more than a dozen regions, including several countries in Europe and Asia, as well as Australia, the United States, Canada, and Brazil.

Each region is divided up into separate physical areas, in what AWS calls availability zones, or just AZs. The data centers are built in small, fault isolated clusters, in these availability zones, which are typically tens of miles apart. This makes a regionwide outage less likely, in the case of a natural disaster. Availability zones within a region, are connected with low-latency private connections. Each region has a minimum of two AZs, but the largest currently has five. Although VPCs must be contained in a single region, they can span multiple availability zones. This allows us to deploy EC2 instances in multiple availability zones, and load balance the traffic.

Thus, if a single availability zone has an outage, the instances in the healthy availability zones, will continue to respond to the requests. Of course, we can only be truly highly available, if we eliminate single points of failure, such as a relational database. Most relational databases allow you to configure a hot standby, or slave, that can take over if the master is unreachable. In AWS, we can make our database highly available, by deploying the slave in a different availability zone from the master.

Here is a highly available VPC architecture, where we have created subnets for our instances, in two different AZs:



Note that a single subnet cannot span availability zones. We use an elastic load balancer, a load balancing service, managed by AWS, to distribute the request to the instances in different AZs.

We have also separated the master and slave databases into AZs.

Elastic load balancer

An elastic load balancer, or ELB for short, has a number of noteworthy features, including the ability to perform health checks on instances. If an instance fails a health check, it automatically stops routing any traffic to it. Originally, AWS offered a single type of ELB, which is now called a classic load balancer. You can configure it to listen for TCP, SSL, HTTP and HTTPS.

For HTTP and HTTPS, it uses a least outstanding requests algorithm for routing. For other protocols, it uses simple round-robin.

The newer type of ELB is called an application load balancer. This allows you to route requests, based upon the content of the request. For example, a request for different microservices, could go to different backend instances.

In addition to HTTP and HTTPS, it has support for the new faster version of HTTP/2. For both types of load balancers, you can terminate HTTPS at the ELB, by uploading a certificate. Either type of load balancer can be used to make your environment highly available. However, not every software application is ready to be put in a load balanced environment.

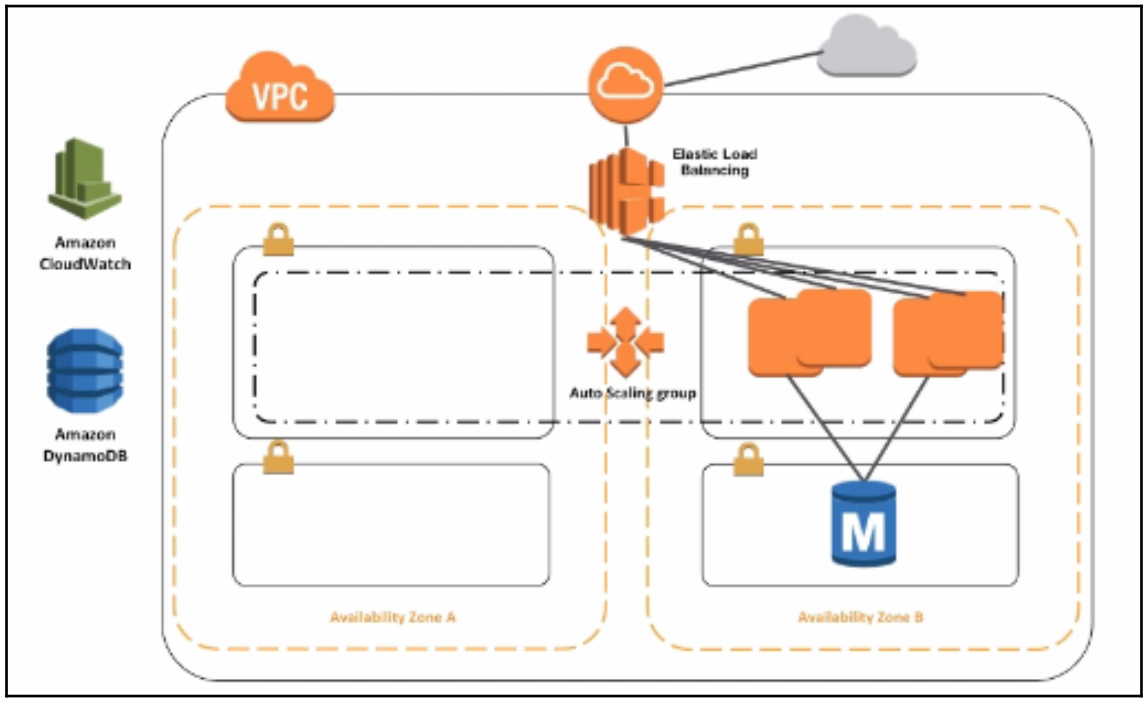
Load balancing stateful applications

Many applications store some type of session or state data, to know for example, that a user has logged in or added an item to their shopping cart. If they store this data at the local filesystem level, then this data will be kept only on one instance, and the other instances won't have a copy. So, if the ELB routes a subsequent request to a different instance, the application will lose the session state and the user may have to log back in again, or have to add their item back into the shopping cart.

One way to cope with this, is through replication of the state data across all the instances. But this can be prone to failure, of the sinking application. Another option, which is supported by ELBs, is to enable sticky sessions. In this case, the ELB uses a cookie, to keep track of the instance ID that the users request was directed to, and to continue to send other requests to the same instance. A much better option, is to store the state data in a separate storage system, that is accessible by all the instances. Some good choices for this are DynamoDB, a NoSQL database, or ElastiCache, which uses Memcached or Redis.

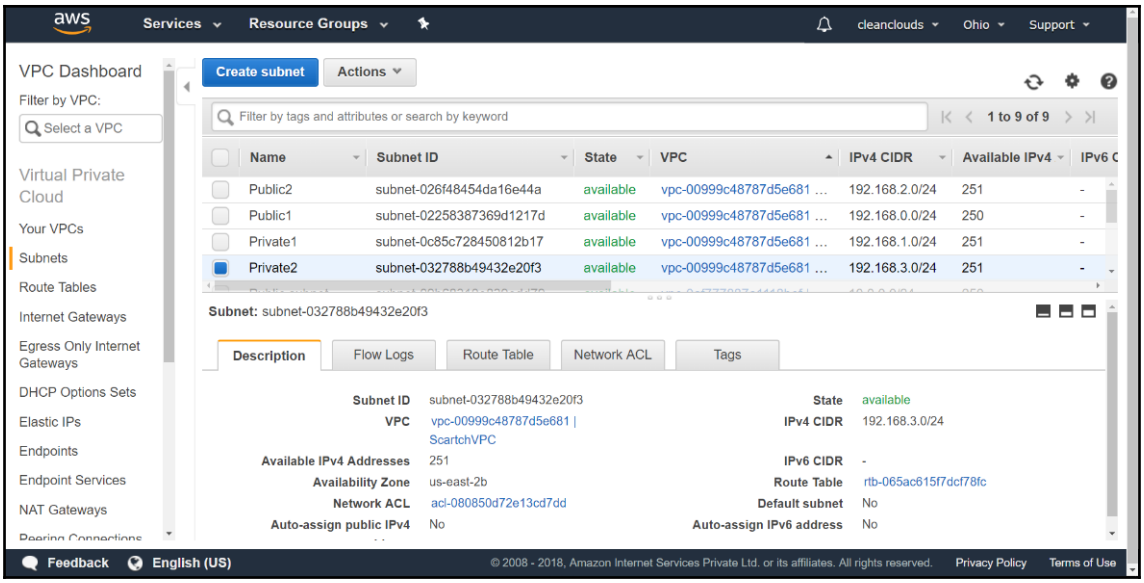
If we go back to our architecture diagram for a second, you can see that if AWS experiences an AZ outage, the load balancer will send all the requests to only one AZ, and we may not have enough capacity for that AZ to handle the increased request volume. The solution for this, is to automatically launch more instances in the working AZ.

For this, we add two more services for our VPC, auto scaling and CloudWatch:



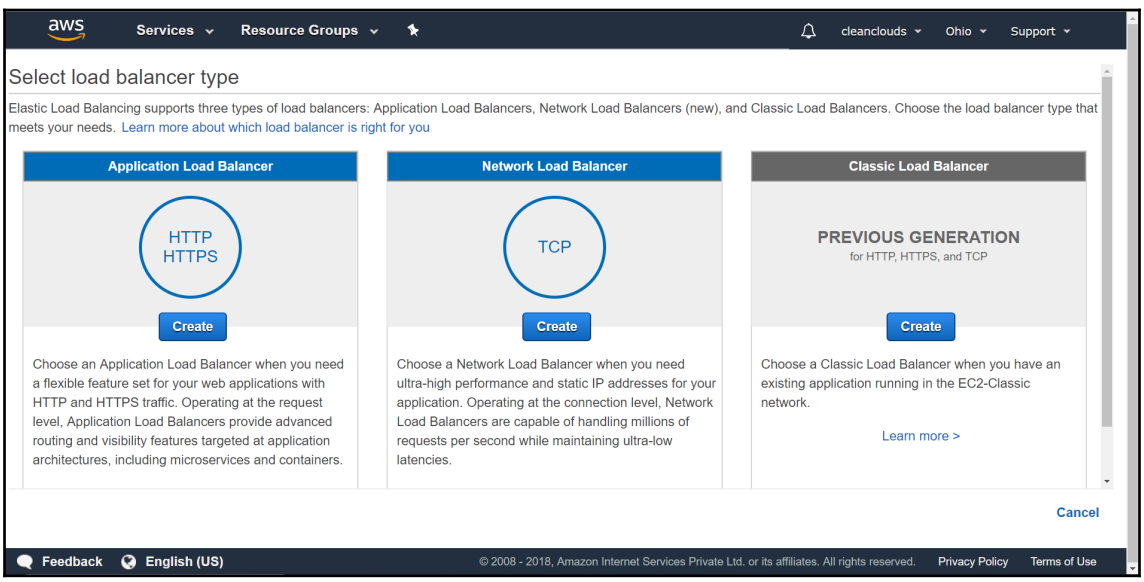
Auto scaling

Auto scaling allows us to horizontally scale, by launching and terminating instances in response to the request to load. So if our application gets a lot of traffic during the day, perhaps we will have ten instances getting requests from the ELB. But overnight, we don't have a lot of users, so maybe we'll go down to a minimum of two. We can specify the minimum and maximum number of instances for our auto scaling group. To maintain our high availability, we should probably never specify a minimum of less than two. When you create an auto scaling group, you will also have to define the launch configuration. This is simply all of the attributes of the EC2s that the auto scaling group will launch, including the AMI, instance size and type, security group, and so on. The way that auto scaling knows when it's time to launch or terminate instances, is through the CloudWatch monitoring service CloudWatch. You configure alarms based upon default metrics, such as average CPU utilization, for the instances in the end or average response latency measured at the ELB. You could also use your own custom metrics, such as memory or thread utilization by pushing metrics to CloudWatch from your EC2 instances, using the CloudWatch API, or by streaming log files to CloudWatch, using the CloudWatch logs agent. Auto scaling launches and terminates instances, when notified by CloudWatch that an alarm has been triggered. An example of an alarm, would be average CPU utilization above 70% for three consecutive measurements. You define a scaling policy, that determines what action to take for a particular alarm. So for this high CPU utilization alarm, we can have auto scaling launch two instances, or maybe add 50% more instances. We could scale in, by configuring an alarm for low CPU utilization, such as below 20%, and to find a scaling policy to terminate two instances. Let's go through a real world example, to help you understand the concepts. Here, you can see we've created some new subnets in the VPC. You can see now we have subnets in both a and b availability zones:



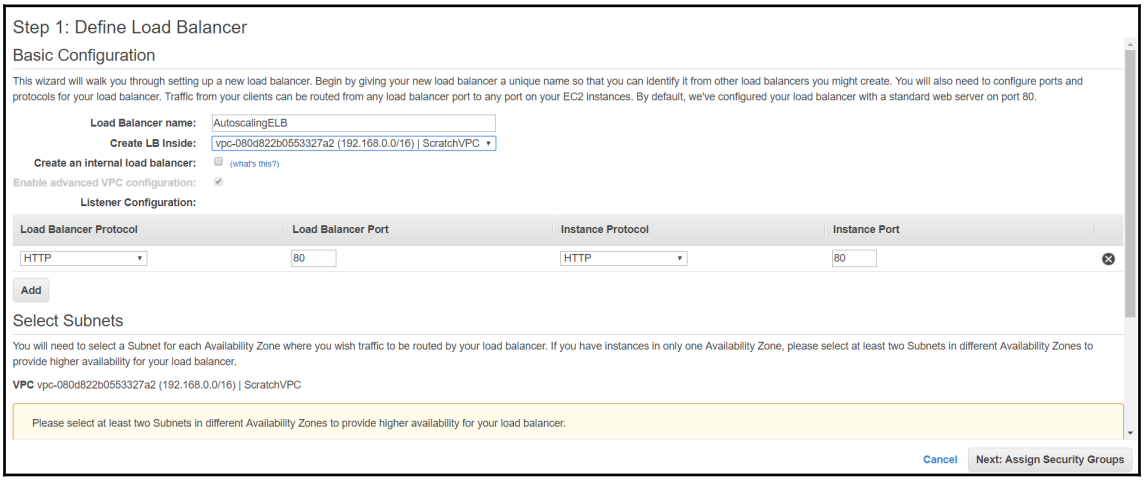
The next thing we need to add to our VPC, is an elastic load balancer. You can just go to **EC2s**, and click **Load Balancers**, and then **Create Load Balancer.**

You will see that you have the option of an **Application Load Balancer**, a **Network Load Balancer**, or a **Classic Load Balancer**:



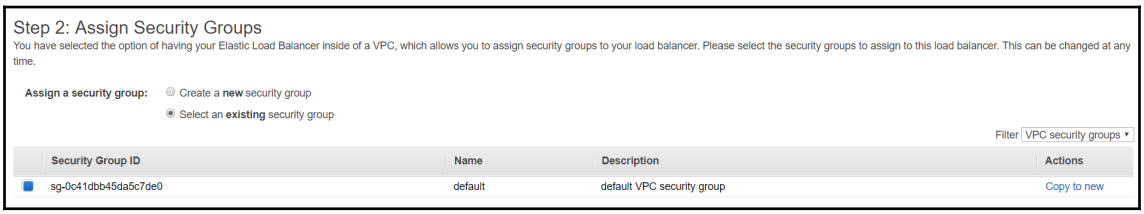
For our purposes, we're not going to need to do content based routing, so let's just choose the simpler **Classic Load Balancer**. We need to give it a name, and select our VPC.

We can leave the listener as **HTTP**:

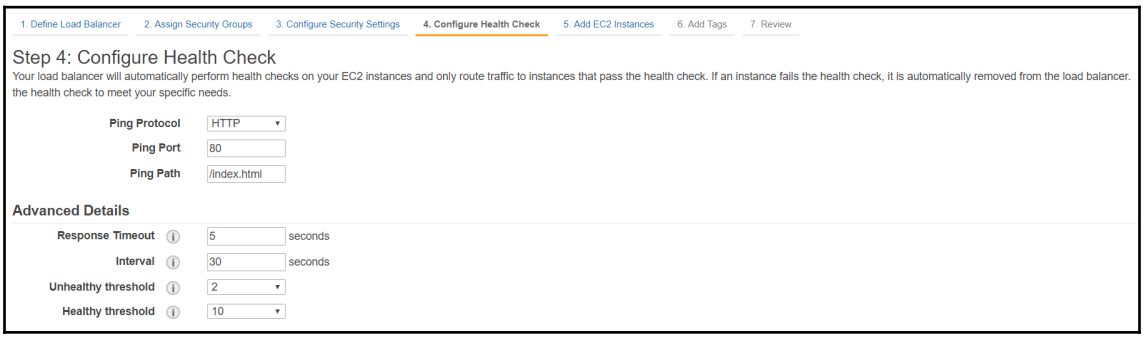


And since this is an internet-facing load balancer, we need to add our two public subnets, and then click **Next: Assign Security Groups**.

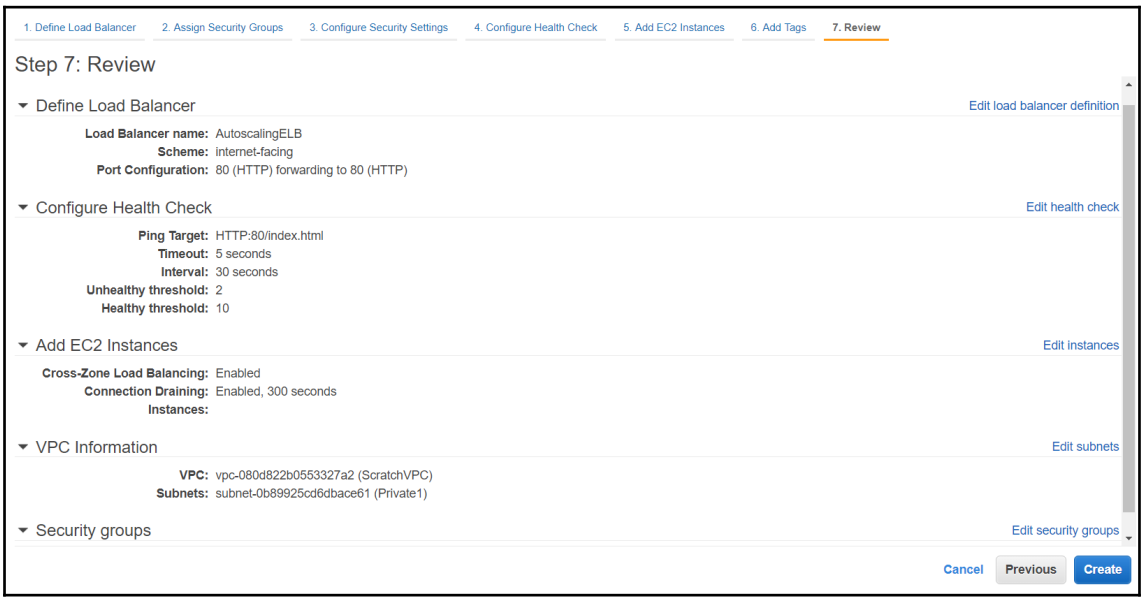
We already have a security group defined for the ELB, which has HTTP port `80` open to the internet:



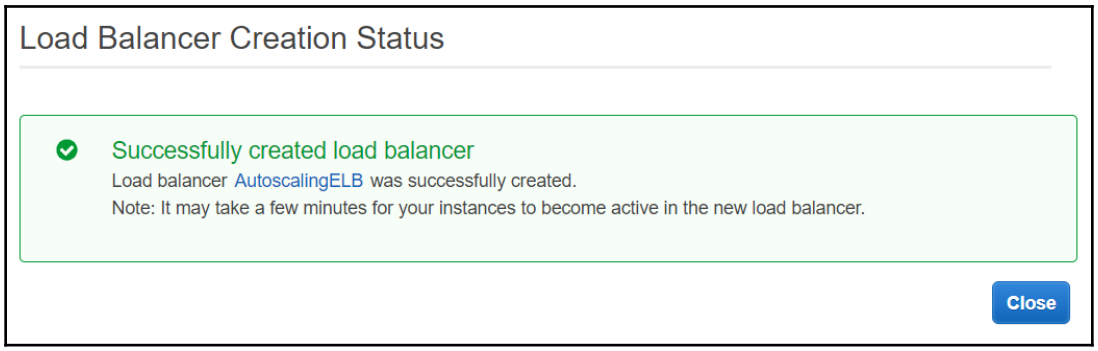
Click **Next: Configure Health Check**, and we'll leave the health check as it is, which is to check the index page of our web server:



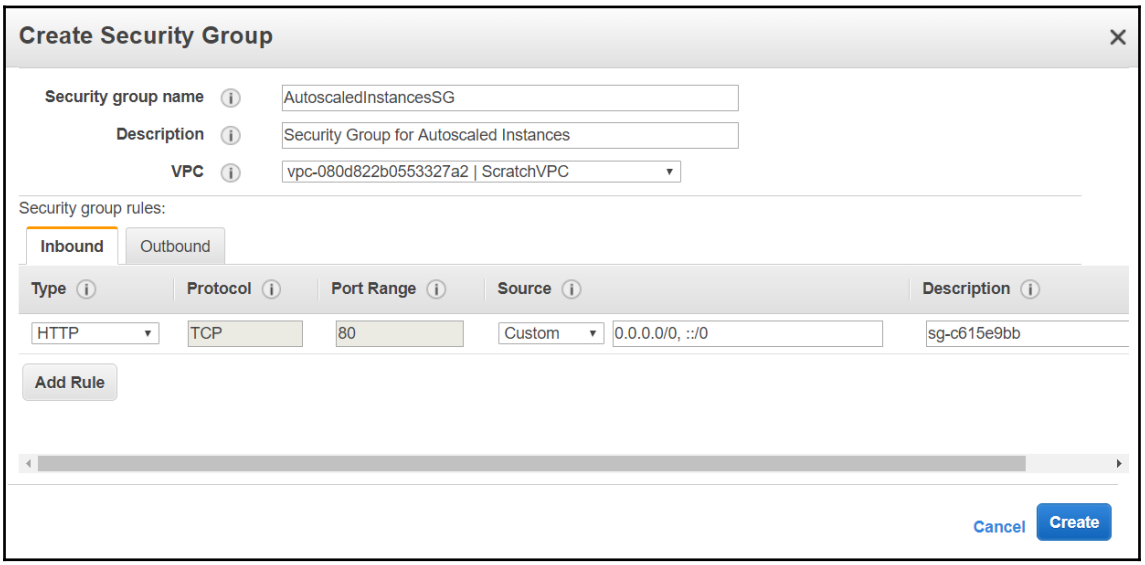
We are not going to add any EC2 instances, because we're going to allow the auto scaling group to do it, so just click **Next: Add Tags**, and then **Review and Create**, and finally **Create**:



You will see that the load balancer has been successfully created:



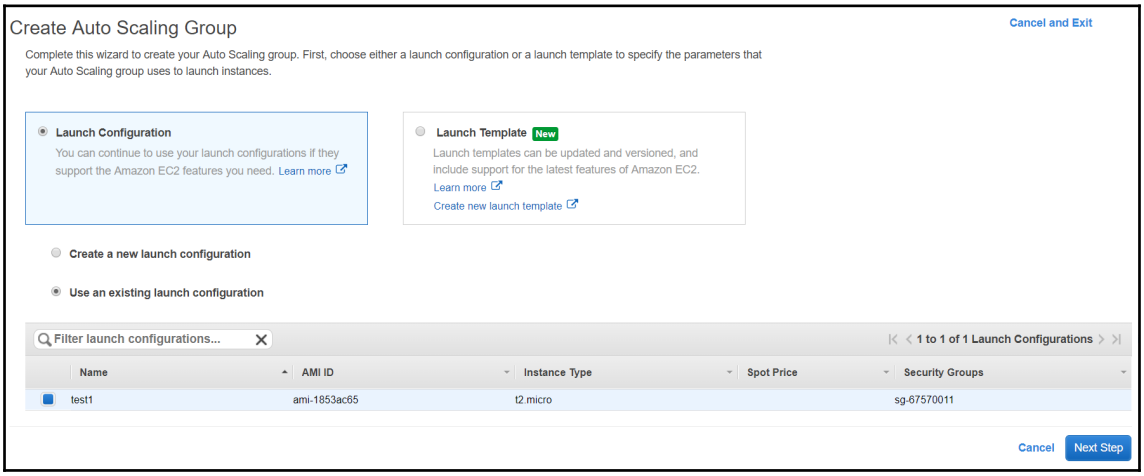
Next, let's create a security group for the instances and our auto scaling group. Click on **Create Security Group**, give this a name and description, select our **VPC**, and for the **Inbound** rule, we're going to allow **HTTP**:



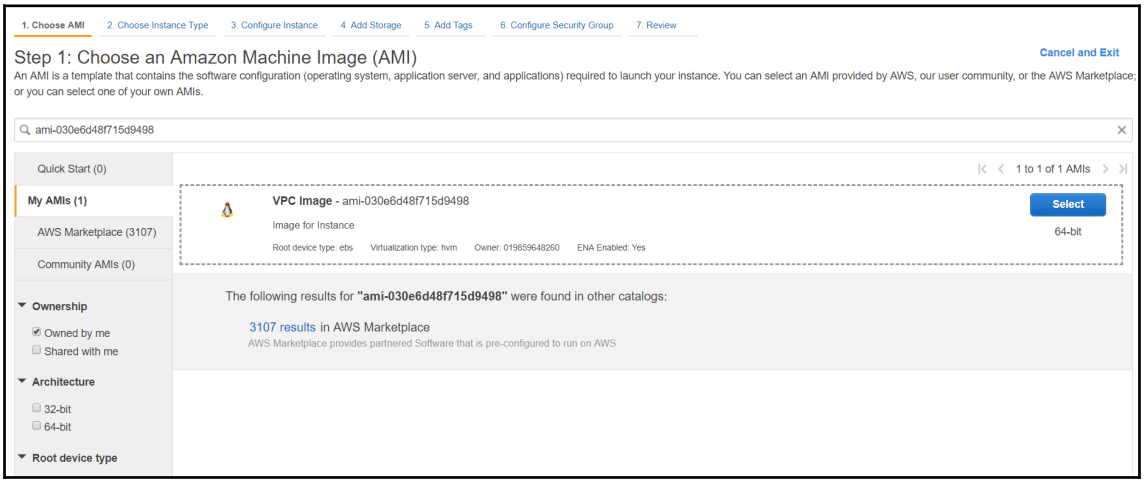
However, we're going to restrict it to the security group, for the load balancer. So type in `sg`, and then select **ELB security group**, and then **Create**.

Now let's go and create our auto scaling group.

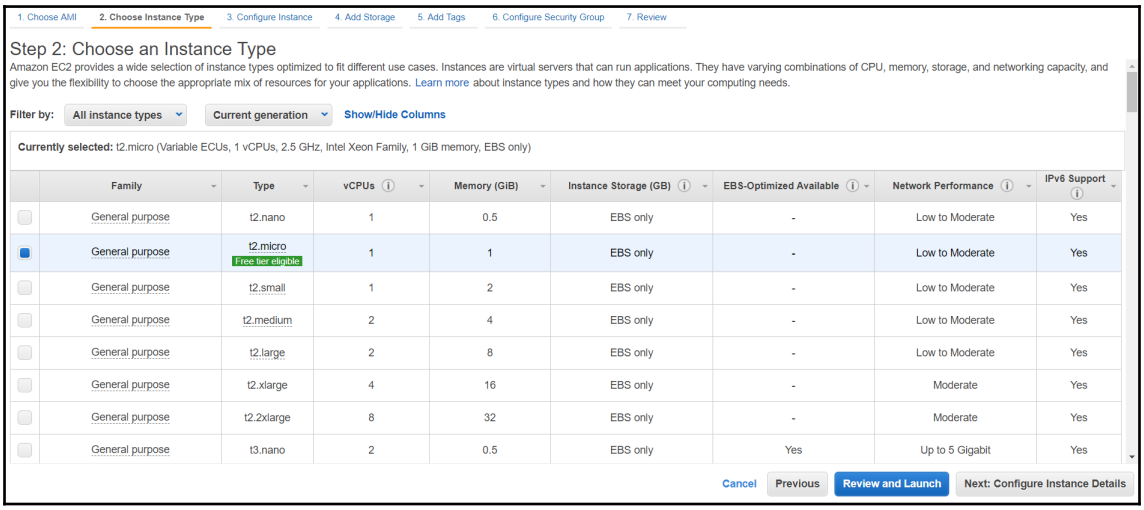
Click on **Auto Scaling Groups | Create Auto Scaling group | Use an existing launch configuration**:



We will then pick one of the AMIs, with just a basic web page:

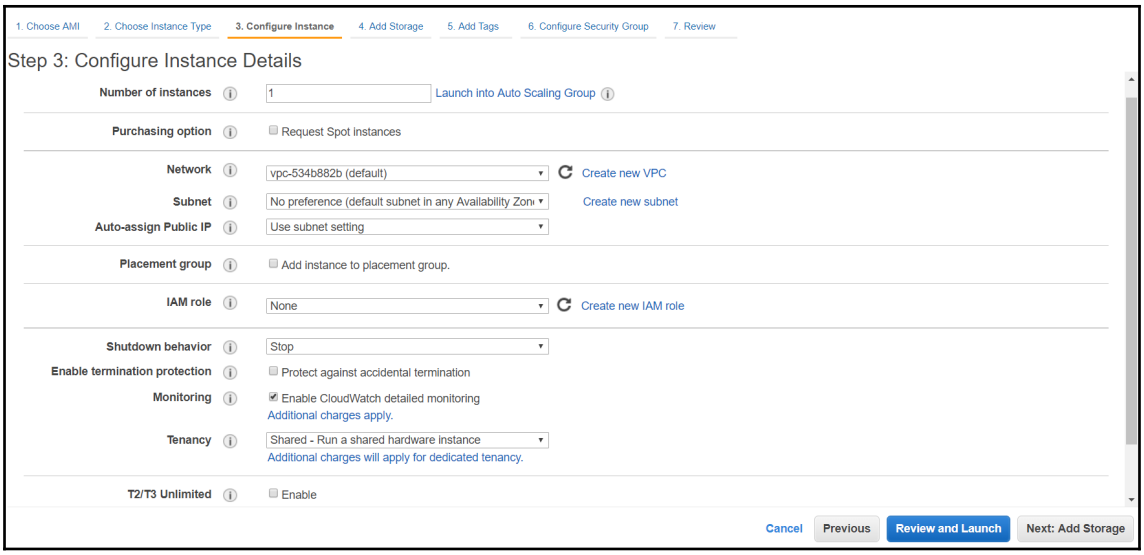


We'll leave the default instance type, **t2.micro**, because this is on the free tier:

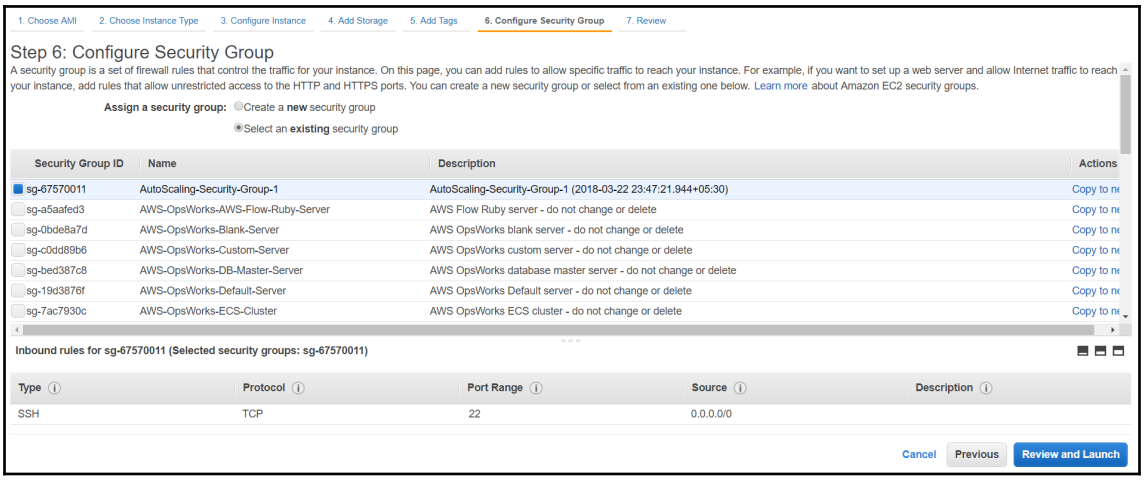


However, in production, you would not want to auto scale these instances, because the CPU is burstable. So in that case, you would choose something like an m3 or an m4. Then, we'll give our launch configuration a name.

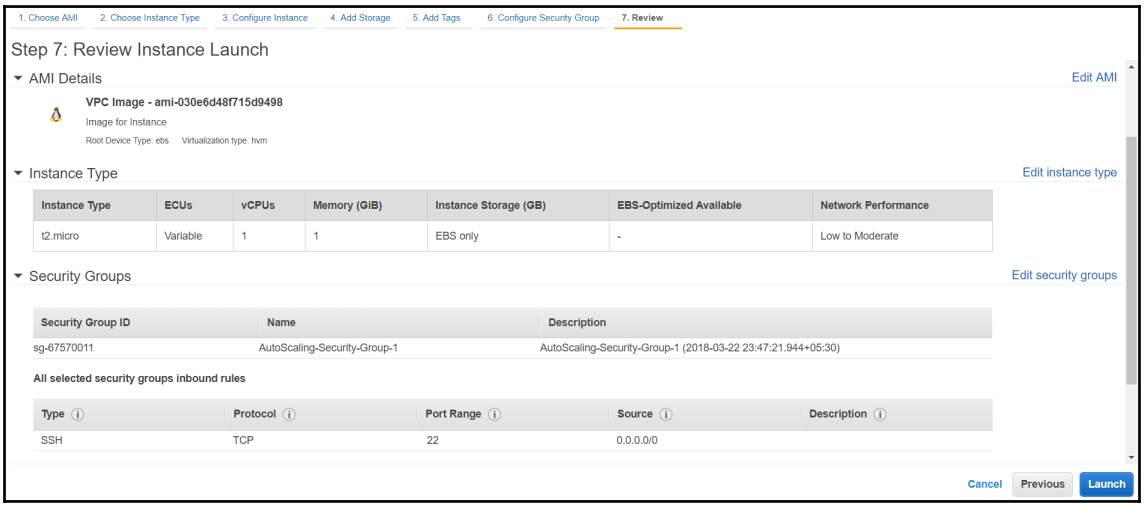
We're going to enable **CloudWatch detailed monitoring**, which will allow CloudWatch to record metrics at one minute intervals, instead of the default five minute intervals. Next, click **Next: Add Storage**:



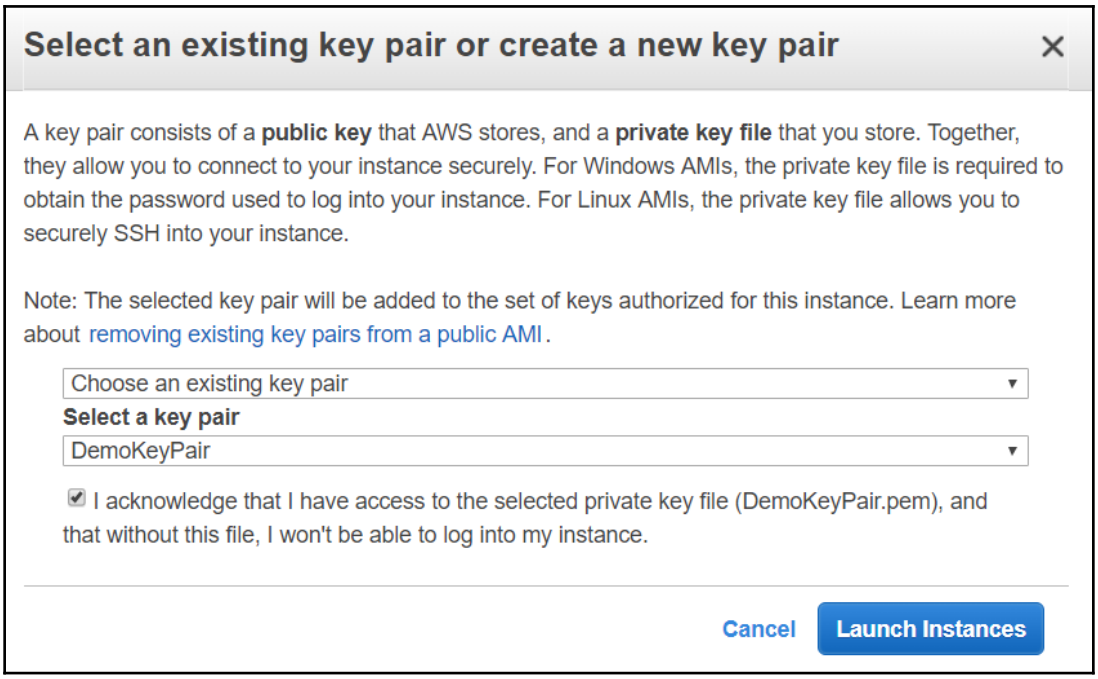
In **Configure Security Group**, let's choose the security group that we just created:



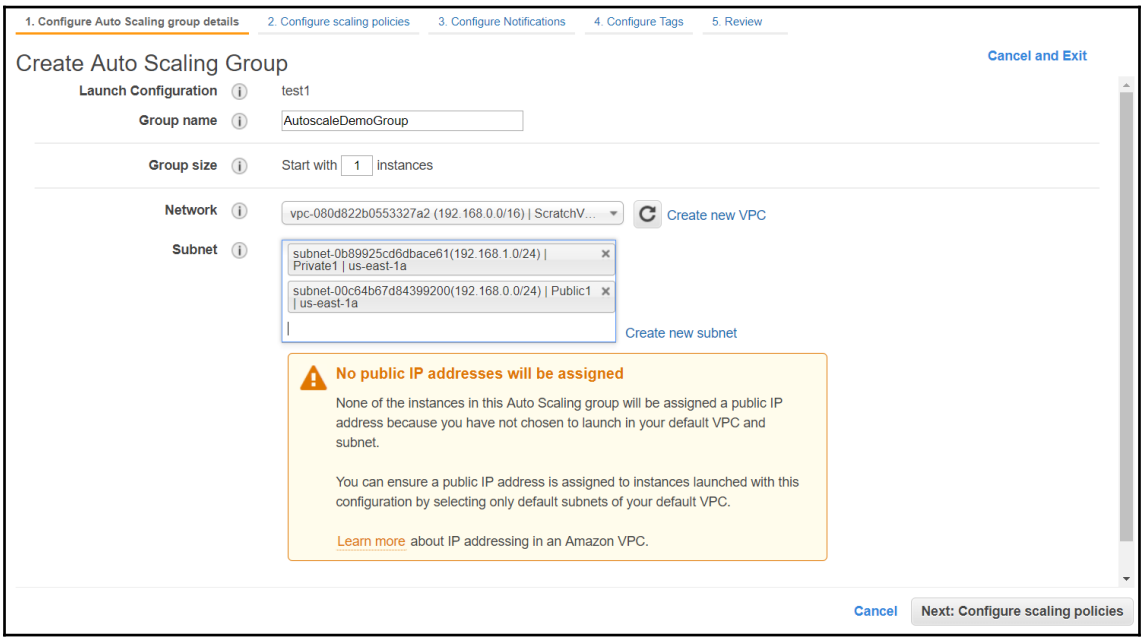
Click **Review and Launch**, and then **Launch**:



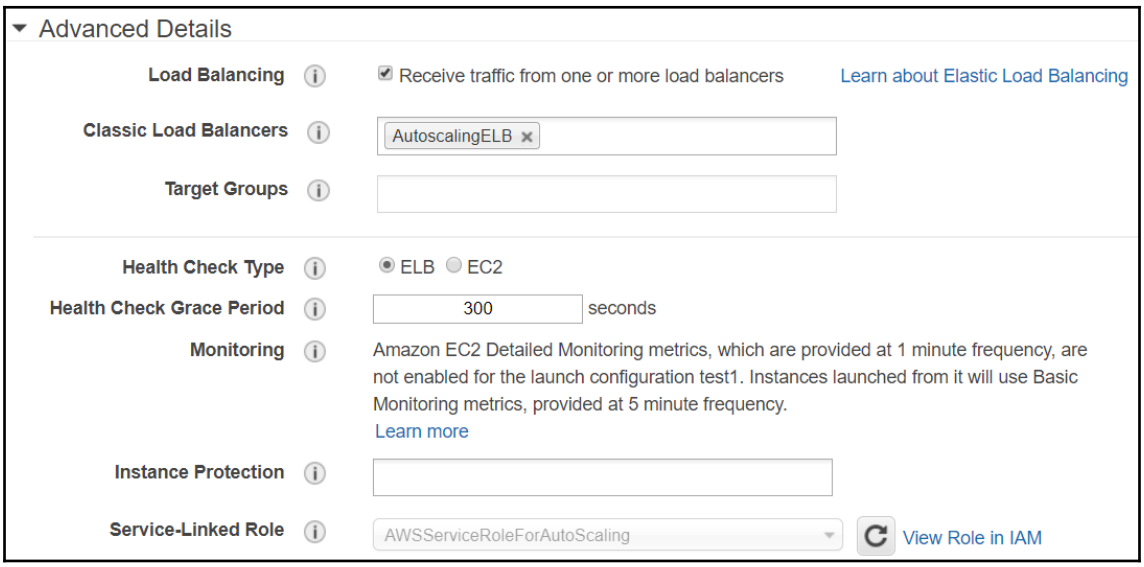
Finally, let's just choose one of our key pairs, then check the box that we acknowledge we have access to the key, and create the launch configuration:



Now we have finished the launch configuration, let's give the group a name; select our VPC and our two private subnets:

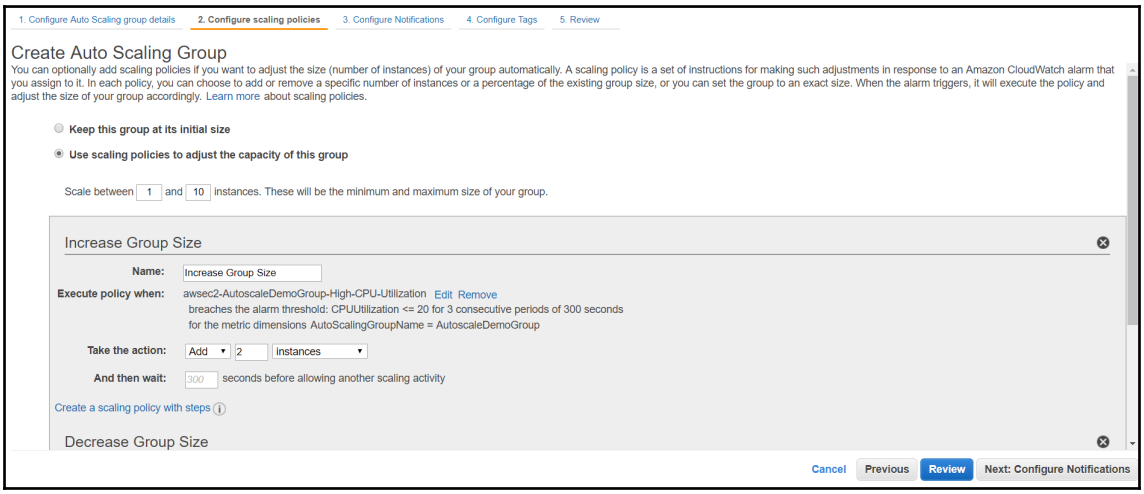


Now let's scroll down to the **Advanced Details**, click the arrow, and check the **Receive traffic from one or more load balancers** box, and select our load balancer:

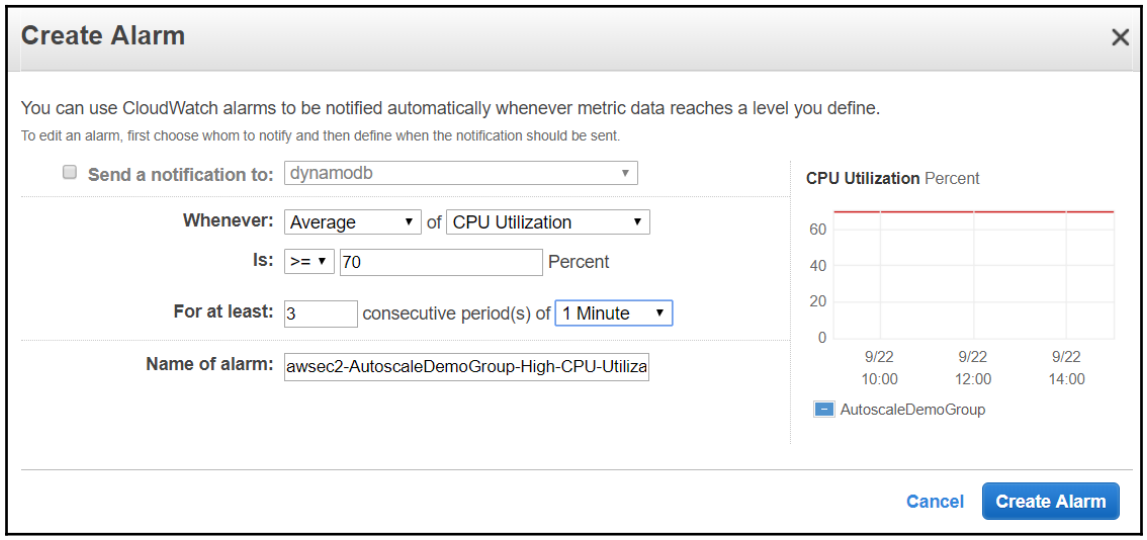


For the **Health Check Type**, select **ELB**, and then click **Next: Configure scaling policies**.

Check the radio button, **Use scaling policies to adjust the capacity of this group**, and let's scale between a minimum of `1` instances, and a maximum of `10`:

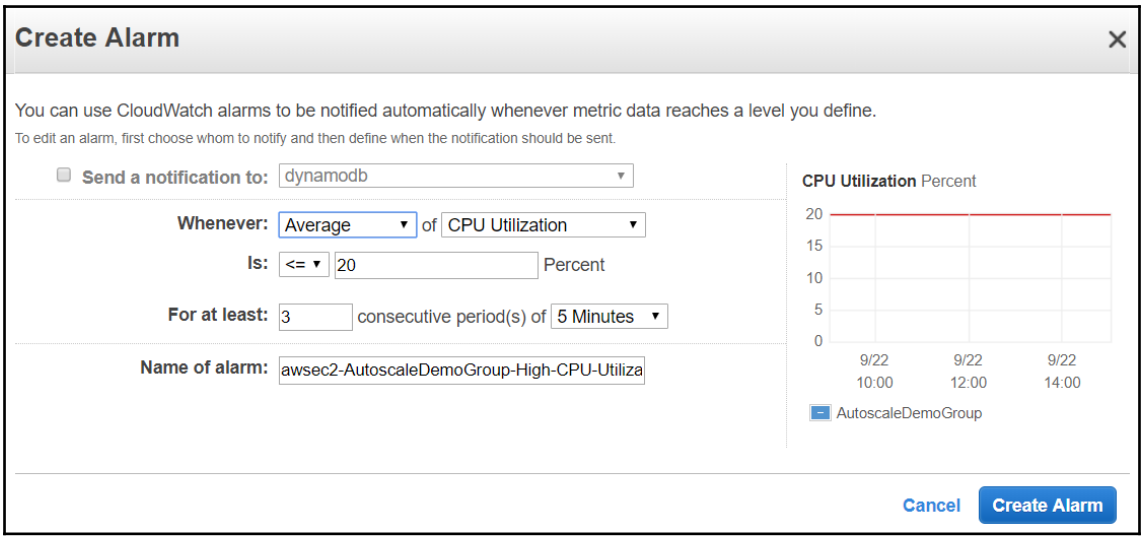


Here, we're going to set the scaling policy and the alarm. So for the first policy, which is going to be increasing the group size, we need to configure an alarm. Click on **Add new alarm**. We can uncheck the **Send a notification to**, although we could send ourselves a notification however, if we like. And then let's go with creating an alarm whenever the average of CPU utilization is greater than or equal to, and let's put `70`%, and for at least three consecutive periods, of one minute. We'll just leave the **Name of alarm** as it is, and click **Create Alarm**:



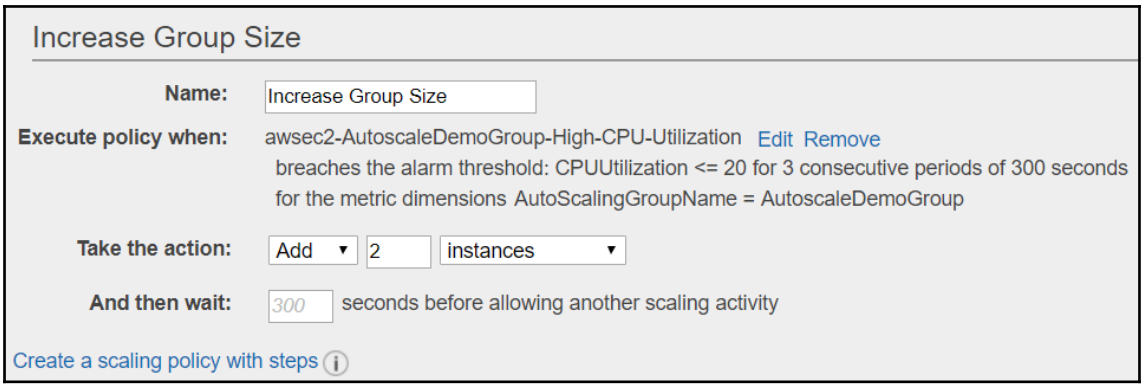
Now we need to do the same thing for decreasing the group size. Create an alarm, let's uncheck the **Send a notification to**, and change the rule to whenever the average of CPU utilization is less than or equal to `20`%, for `3` consecutive periods of **5 Minutes**.

Let's just call this, low CPU utilization. Then finally, click **Create Alarm**:

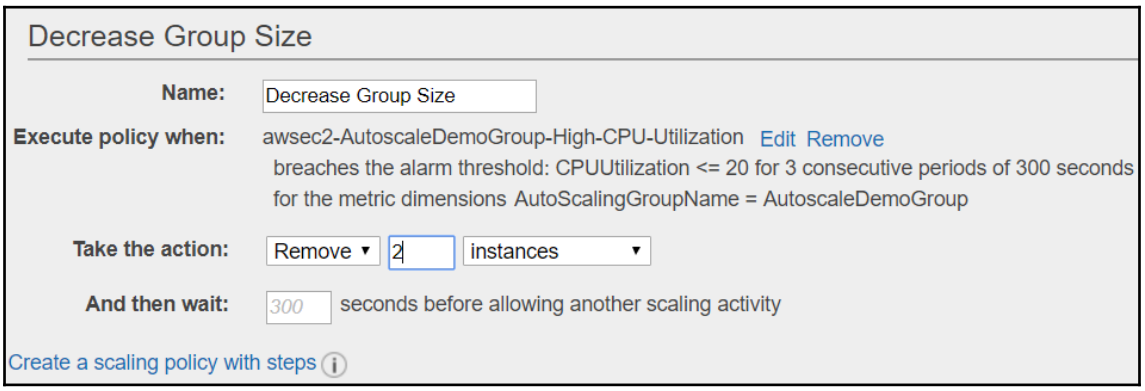


Let's go back to our first group, for the increased group size, and select **Create a simple scaling policy**.

Let's just say that we're going to **Add**, 2 instances, whenever this alarm goes off:



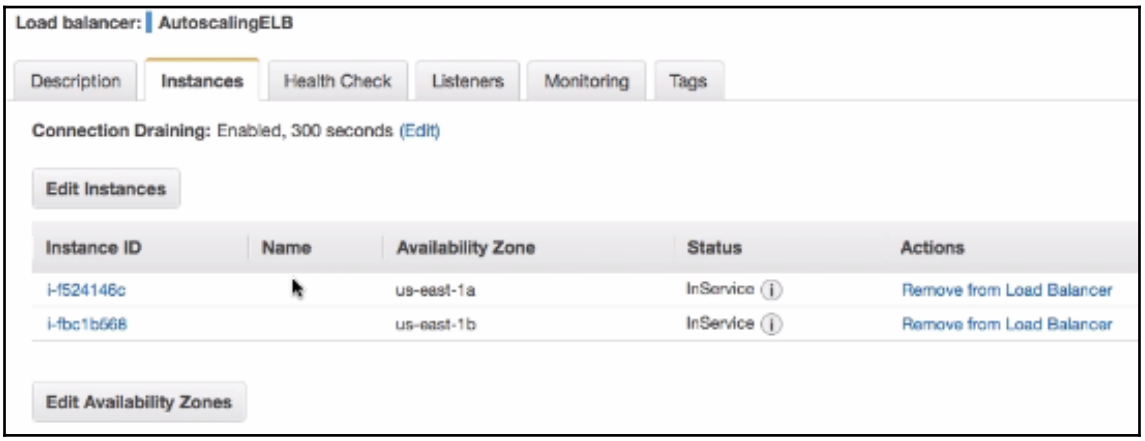
Let's go down to the same link and, under **Decreased Group Size**, create a simple scaling policy, and let's say we remove two instances, whenever this alarm goes off:



Finally, let's just click on **Review**, and then **Create Auto Scaling group**, and then **Close**.

We now have our auto scaling group. If we go check our **EC2s**, there should be two more launching.

Now let's go into our **Load balancers**, and after a few minutes, you should be able to click on the **Instances** tab, and see the two instances in service, behind the load balancer:



Summary

In this chapter, we discussed classic and VPC EC2 instances, and how to cope with a mixed environment. We also described the default VPC, which for simple public applications and proof of concepts may be all that you need. However, it will require a lot of modifications to make it suitable for most production workloads and provide the required security. We created a simple VPC, first by using the VPC Wizard, and then we created one from scratch. We created the VPC, attached an internet gateway, created private and public subnets and route tables, and launched an NAT gateway in our public subnet. We discussed accessing the internet through virtual private gateways, VPN connections, direct connect, and VPC peering. We talked about how to make your VPCs secure, by using network access control lists, and Bastion instances.

By way of some subsequent steps, I suggest getting some practice with your free AWS account. Create some VPCs and launch applications on EC2 instances in them. Don't forget to implement best security practices and make your application highly available. You may want to go back and review the demonstration chapters in this book, pausing as needed. Good luck using your new AWS skills.

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:

Expert AWS Development
Atul V. Mistry

ISBN: 978-1-78847-758-1

- Learn how to get up and running with AWS Developer Tools.
- Integrate the four major phases in the Release Processes. Source, Build, Test and Production.
- Learn how to integrate Continuous Integration, Continuous Delivery, and Continuous Deployment in AWS.
- Make secure, scalable and fault tolerant applications.
- Understand different architectures and deploy complex architectures within minutes

AWS Administration - The Definitive Guide - Second Edition
Yohan Wadia

ISBN: 978-1-78847-879-3

- Take an in-depth look at what's new with AWS, along with how to effectively manage and automate your EC2 infrastructure with AWS Systems Manager
- Deploy and scale your applications with ease using AWS Elastic Beanstalk and Amazon Elastic File System
- Secure and govern your environments using AWS CloudTrail, AWS Config, and AWS Shield
- Learn the DevOps way using a combination of AWS CodeCommit, AWS CodeDeploy, and AWS CodePipeline
- Run big data analytics and workloads using Amazon EMR and Amazon Redshift
- Learn to back up and safeguard your data using AWS Data Pipeline
- Get started with the Internet of Things using AWS IoT and AWS Greengrass

Leave a review - let other readers know what you think

Please share your thoughts on this book with others by leaving a review on the site that you bought it from. If you purchased the book from Amazon, please leave us an honest review on this book's Amazon page. This is vital so that other potential readers can see and use your unbiased opinion to make purchasing decisions, we can understand what our customers think about our products, and our authors can see your feedback on the title that they have worked with Packt to create. It will only take a few minutes of your time, but is valuable to other potential customers, our authors, and Packt. Thank you!

Index

F

G

H

I

K

L

M

N

P