Chapter 13: Scaling out KVM with AWS

Virtualization is a hard problem if you take a look at it up close – it is complicated to emulate complete computers in order to be able to run operating systems on them. For obvious reasons, getting those virtual machines into a cloud is even harder. After that, things really start to get messy. Conceptually, creating clusters of machines that can run on demand is even more complicated, based on the sheer number of machines that must run at the same time. Also, there's a need to create not only emulated computers but also all the networking and infrastructure to support larger deployments. Creating a global cloud – one that not only runs millions of machines but is also almost omnipresent in even the most remote parts of the globe – is a task that few companies have ever tried, and only a couple have succeeded. This chapter will cover those big cloud providers in

general, and then Amazon, as the biggest of them all. Our main idea is to present what makes Amazon tick, how it relates to the rest of the topics covered in this book, and how to use the services Amazon enables in the real world on real machines.

Amazon Web Services (**AWS**) is a unique set of tools, services, and infrastructure that enable cloud services on a really massive scale, a scale that is so huge that it becomes hardly comprehensible. When we are talking about thousands of sites using millions of servers to run billions of applications, even enumerating these things becomes a big problem, and management is something that can easily span not only a single chapter, but probably multiple books. We are going to try to introduce you to the most important services and parts of the AWS cloud, and try to explain what they can be used for and when.

In this chapter, we will cover the following topics:

- Introduction to AWS

- Preparing and converting virtual machines for AWS
- Building hybrid KVM clouds with Eucalyptus

Introduction to AWS

While talking about cloud services, AWS is one that needs almost no introduction, although few people actually understand how big and complex a system the whole Amazon cloud is. What is completely certain is that, at this time, it is unquestionably the biggest and most used service on the planet.

Before we do anything else, we need to talk about how and why AWS is so important, not only in regard to its impact on the internet but also on any task that even remotely tries to provide some kind of scaling.

As we already have done a couple of times in this book, we will start with the basic premise of the AWS cloud – to provide a broadly scalable and distributed solution that will encompass all pos-

sible scenarios for performing any type of workload on the internet.

In almost every other place in this book where we mentioned the cloud, we talked about scaling up, but when we try to describe AWS as being able to scale, we are talking about probably one of the largest providers of capacity and scalability on the planet.

Right now, there are more or less four really big cloud providers on the internet: AWS, Microsoft Azure, Google Cloud Platform, and Alibaba. Since all the numbers are confidential for business reasons, the number of servers and sheer capacity they can provide is something analysts try to estimate, or more frequently guess, but it has to be in the millions.

Approaching the cloud

Although on the surface they are now competing for the same cloud market, all the players came from different backgrounds, and the way they use their infrastructure even now is vastly different. In this market, Amazon was first, and it got a

head start that in IT seems almost unbelievable – roughly 6 *years*. Amazon introduced its Amazon Web Services in 2006 but it started the development of the service a couple of years earlier. There is even a blog post that mentions the service published back in 2004. The idea for AWS was basically conceived once Amazon realized it had a vast infrastructure that was unmatched in the market, and that by expanding it and offering it as a service, it could make a profit. At that point in time, the infrastructure they had was used to provide the Amazon.com service.

This idea was different from anything on the market. People were used to having collocated computers in data centers, and being able to rent a server in the *cloud*, but the concept of renting just the part of the stack they needed instead of the entire hardware infrastructure was something new. The first big service AWS offered was simple, and was even named like that – **Simple Storage Service** (**S3**), along with **Elastic Compute Cloud** (**EC2**). S3 was basically cloud-backed storage that offered almost unlimited storage resources for those who could pay for

them in pretty much the same way it is available even today. EC2 offered computing resources.

Offerings expanded to a **Content Delivery Network** (**CDN**) and much more during the next 6 years, while the competition was still trying to get to grips with what the cloud actually meant.

We'll come back to the services AWS offers in a moment, but only after we mention the competition they eventually got in what has become a market worth hundreds of *billions* of dollars yearly.

Microsoft realized it would need to build up an infrastructure to support itself and its customers sometime in the late 2000s. Microsoft had its own business support infrastructure in place to run the company, but there were no public services offered to the general public at that time. That changed once **Microsoft Azure** was introduced in 2010. Initially, it was called **Windows Azure**, and it was mainly created to run services for both Microsoft and its partners, mainly on Windows. Very quickly, Microsoft realized that

offering just a Microsoft operating system in the cloud was going to cost them a lot of customers, so Linux was also offered as an operating system than could be installed and used.

Azure now runs as a publicly available cloud, but a large portion of it is still used by Microsoft and its services, most notably **Office 365** and a myriad of Microsoft training and marketing solutions.

Google, on the other hand, came to the market in a different way. They also realized the need for a cloud offering but limited their first engagement with the cloud to offering a single service called **App Engine**, in 2008. It was a service targeted at the web developer community, and Google even gave 10,000 free licenses for the usage of the service in a limited way. At its core, this service and almost all the services that came after it came out with the premise that the web needs services that will enable developers to quickly deploy something that may or may not scale and that may or may not work. Therefore, giving it for

free meant that a lot of developers were inclined to use the service just for simple testing.

Google now also has a vast number of services offered, but when you take a look from outside at the actual services and the pricing, it seems that Google has created its cloud as a way to lease out extra capacity it has available in its data centers.

Multi-cloud

Looking a few years back, both Azure and Google Cloud Platform had a viable cloud service, but compared to what AWS was offering, their services were simply not up to par. AWS was the biggest player, both in terms of market share, but also in people's minds. Azure was considered as being more Microsoft oriented, although more than half of the servers running on it are Linux-based, and Google just wasn't perceived as a competitor; their cloud looked more like a side business than a viable proposal to run a cloud.

Then came **multi-cloud**. The idea is simple – do not use a single cloud to deploy your services;

use multiple cloud providers to provide both *data redundancy*, *availability*, and *failover*, and most important – *cost reduction and agility*. It may sound strange, but one of the biggest costs when using a cloud service is getting data out of it. Usually, getting data into the cloud, be it the user uploading data, or you deploying data on a server, is either free or has an extremely low cost, which makes sense, since you are more likely to use more services on this particular cloud if you have a lot of data online. Once you need to extract your data, it becomes expensive. This is intentional, and it keeps users locked into the cloud provider. Once you upload your data, it is much cheaper to just keep it on the servers and not try to work with it offline. But the data is not the only thing that has to be considered when talking about multi-cloud; services are also part of the equation.

Why multi-cloud?

Many companies (and we must stress that multi-cloud users are mostly big companies because of the costs involved) are scared of being locked

into a particular platform or technology. One of the biggest questions is what happens if a platform changes so much that the company has to redo part of its infrastructure? Imagine that you are a multibillion-dollar company running an enterprise application for hundreds of thousands of your own users. You chose the cloud for the usual reasons – to keep capital expenditures down and to be able to scale your services. You decided to go with one of the big providers. Suddenly, your provider decides it is going to change technologies and will phase out some part of the infrastructure. When it comes to shifts like that it usually means that your current setup will slowly become much more expensive, or you are going to lose some part of the available functionalities. Thankfully, these things also typically stretch into years, as no sane cloud provider is going to go through a strategic change overnight.

But a change is a change, and you as a corporation have a choice – stay with the provider and face the consequences in the form of a much higher price for your systems – or redesign the

systems, which will also cost money, and may take years to finish – sometimes decades.

So, a lot of companies decided on a very conservative strategy – to design a system that could run on any cloud, and that means using the lowest common denominator of all available technologies. This also means that the system can be migrated from cloud to cloud in a matter of days. Some of them then decided to even run the system on different clouds at the same time. This is an extremely conservative approach, but it works.

Another big reason to use a multi-cloud strategy is the complete opposite of the one that we just mentioned. Sometimes, the idea is to use a particular cloud service or services that are the best in a very specialized task. This means choosing different services from different providers to perform separate tasks but to do it as efficiently as possible. In the long run, it will also mean having to change providers and systems from time to time, but if the core system that the com-

pany uses is designed with that in mind, this approach can have its benefits.

Shadow IT

There is another way that a company can become a multi-cloud environment without even knowing it, this is usually called **Shadow IT**. If a company does not have a strict security policy and rules, some of the workers might start using services that are not part of the services that they are provided with by the company. It could be a cloud storage container, a videoconferencing system, or a mailing list provider. In bigger companies, it could even be that entire departments start using something from different cloud providers without even realizing it. All of a sudden, there is company data on a server that is outside of the scope that company's IT covers or is able to cover.

One of the better examples of this particular phenomenon was how the usage of video conferencing services changed during the COVID-19 virus worldwide pandemic. Almost all companies had

an established communication system, usually a messaging system that covered the whole company. And then, literally overnight, the pandemic put all workers in their homes. Since communication is the crucial thing in running a company, everyone decided to switch to video and audio conferencing in the span of a week, globally. What happened next can and probably will become a bestselling book theme one day. Most companies tried to stick with their solution but almost universally that attempt failed on the first day, either because the service was too basic or too outdated to be used as both an audio and video conferencing solution, or because the service was not designed for the sheer volume of calls and requests and crashed.

People wanted to coordinate, so suddenly nothing was off the table. Every single video conferencing solution suddenly became a candidate. Companies, departments, and teams started experimenting with different conferencing systems, and cloud providers soon realized the opportunity – almost all the services instantly became free for even sizeable departments, allow-

ing, in some cases, up to 150 people to participate in conferences.

A lot of services crashed due to demand, but big providers mostly were able to scale up to the volume required to keep everything running.

Since the pandemic was global, a lot of people decided they also needed a way to talk to their family. So individual users started using different services at home, and when they decided something worked, they used it at work. In a matter of days, companies became a multi-cloud environment with people using one provider for communication, another for email, a third for storage, and a fourth for backups. The change was so quick that sometimes IT was informed of the change a couple of days after the systems went online and the people were already using them.

This change was so enormous that at the time we are writing this book, we cannot even try to predict how many of these services are going to become a regular part of the company toolset, once

users realize something works better than the company-provided software. These services further prove this point by being able to work continuously through a major disaster like this, so there is only so much a company-wide software usage policy can do to stop this chaotic multi-cloud approach.

Market share

One of the first things everyone mentions as soon as cloud computing companies and services are mentioned is the market share each one of them has. We also need to address this point, since we said that we are talking about the *biggest one* or the *second one*. Before multi-cloud became a thing, market share was divided basically between AWS, with the biggest market share; Azure as a distant second; followed by Google and a big group of *small* providers, such as Alibaba, Oracle, IBM, and such.

Once multi-cloud became a thing, the biggest problem became how to establish who had the biggest actual market share. All the big compa-

nies started using different cloud providers and just simply trying to add up the market share of the providers became difficult. From different polls, it is clear that Amazon is still the leading provider but that companies are slowly starting to use other providers together with Amazon services.

What this means is that, right now, AWS is still by far the cloud provider of choice but the choice itself is no longer about a single provider. People are using other providers as well.

Big infrastructure but no services

Sometimes, trying to divide the market share also has another point of view that must be considered. If we are talking about cloud providers, we usually think that we are talking about companies that have the biggest infrastructure created to support cloud services. Sometimes, in reality, we are actually comparing those companies that have the biggest portfolio of the services on the market. What do we mean by this?

There is a distinct company that has a big cloud presence but uses its own infrastructure almost exclusively to deliver its own content – Facebook. Although it's hard to compare infrastructure sizes in terms of the number of servers, data centers, or any other metric, since those numbers are a closely guarded secret, Facebook has an infrastructure that is in the same order of magnitude in size as AWS. The real difference is that this infrastructure is not going to serve services for third parties, and in reality, it was never meant to do so; everything that Facebook created was tailor-made to support itself, including choosing locations for the data centers, configuring and deploying hardware, and creating software. Facebook is not going to suddenly turn into another AWS; it's too big to do that. Available infrastructure does not always correlate with cloud market share.

Pricing

Another topic we have to cover, if just to mention it, is the one of pricing. Almost every mention of the cloud in this book is technical. We

compared probably every possible metric that made any sense, from **IOPS**, through **GHz**, to **PPS** on the network side, but the cloud is not only a technical problem – when you have to put it in use, someone has to pay for it.

Pricing is a hot topic in the cloud world since the competition is fierce. All the cloud providers have their pricing strategies, and rebates and special deals are almost the norm, and all that turns understanding pricing into a nightmare, especially if you are new to all the different models. One thing is certain, all the providers will say that they are going to charge you only for what you use but defining what they actually mean by that can be a big problem.

When starting to plan the costs of deployment, you should first stop and try to define what you need, how much of it you need, and whether you are using the cloud in the way it is meant to be used. By far the most common mistake is to think that the cloud is in any form similar to using a normal server. The first thing people notice is the price of a particular instance, in a particular

data center, running a particular configuration. The price will usually be either the monthly cost of the instance, and will usually be prorated, so you will pay only for the part that you use, or the price will be given for a different time unit – per day, per hour, or maybe even per second. This should be your first clue: you pay for using the instance, so in order to keep the costs down, do not keep your instances running all the time. This also means that your instances must be designed to be quickly brought up and down on demand so using the *standard* approach of installing a single or multiple servers and running them all the time is not necessarily a good option here.

When choosing instances, the options are literally too numerous to name here. All the cloud providers have their own idea of what people need, so you can not only choose simple things such as the number of processors or the amount of memory but also get an OS preinstalled, and get wildly varied types of storage and networking options. Storage is an especially complicated topic we are just going to quickly scratch the sur-

face of here and only mention later. All the cloud providers offer two things – some sort of storage meant to be attached to instances, and some sort of storage that is meant to be used as a service. What a given provider offers can depend on the instance you are trying to request, the data center you are trying to request it in, and a number of other factors. Expect that you will have to balance three things: capacity, pricing, and speed. Of course, here, we are talking about instance storage. Storage as a service is even more complicated and with that, you have to think about pricing and capacity, but also about other factors like latency, bandwidth, and so on.

For example, AWS enables you to choose from a variety of services that go from database storage, file storage, and long-term backup storage, to different types of block and object storage. In order to use these services optimally, you need to first understand what is being offered, how it's being offered, what the different costs involved are, and how to use them to your advantage.

Another thing that you will notice quickly is that when the cloud providers said that everything is a service, they really meant it. It is completely possible to have a running application without having a single server instance. Tasks can be accomplished by stitching different services together, and this is by design. This creates an enormously flexible infrastructure, one that scales quickly and easily, but requires not only a different way of writing code but a completely different mindset when designing the solution you need. If you have no experience, find an expert, since this is the fundamental problem with your solution. It has to run on the cloud, not be running on your virtual machines that happen to be in the cloud.

Our advice to you is simple – *read a lot of documentation*. All the providers have excellent resources that will enable you to understand what their service provides, and how, but what the thousands of pages will not tell you is how it compares to the competition, and more importantly, what is the optimal way of connecting the services together. When paying for cloud ser-

vices, expect that you will make a mistake once in a while and pay for it. This is why it's useful to use a pay-as-you-go option when getting services deployed – if you make a mistake, you will not run up a huge bill; your infrastructure will simply stop.

The other thing to mention when talking about pricing is that everything costs a little, but the composite price for a given configuration can be huge. Any additional resource will cost money. An *internal link between servers*, *external IP address*, *firewall*, *load balancer*, *virtual switch*, these are all the things that we usually don't even think about when designing infrastructure, but once we need them in the cloud, they can become expensive. Another thing to expect is that some of the services have different contexts – for example, network bandwidth can have a different price if you are transferring data between instances or to the outside world. The same goes for storage – as we mentioned earlier in this chapter, most providers will charge you different prices when storing and getting data out of the cloud.

Data centers

A couple of times in this chapter, we have mentioned **data centers**, and it is important that we talk a little bit about them. Data centers are at the core of the cloud infrastructure, and in more ways than you may think of. When we talked about a lot of servers, we mentioned that we usually group them into racks, and put the racks into data centers. As you are probably aware, a data center is in its essence a group of racks with all the infrastructure that servers need to function optimally, both in terms of power and data, but also when it comes to cooling, securing, and all the other things required to keep the servers running. They also need to be logically divided into **risk zones** that we usually call **fault domains**, so that we can avert various risks associated with the *we deployed everything on one rack* or *we deployed everything on one physical server* scenarios.

A data center is a complex infrastructure element in any scenario since it requires a combination of things to be efficient, secure, and re-

dundant. Putting a group of servers in a rack is easy enough, but providing *cooling* and *power* is not a simple task. Add to that the fact that the cooling, power, and data all have to be *redundant* if you want your servers to work, and that all that needs to be secure, both from fires, floods, earthquakes and people, and the cost of running a real data center can be high. Of course, a data center running a couple of hundred servers is not as complex as the one running thousands or even tens of thousands, and the prices rise with the size of the facility. Add to that that having multiple data centers creates additional infrastructure challenges in connecting them so costs add up.

Now multiply that cost by a hundred since this is the number of data centers each of the cloud providers keep around the world. Some of the centers are small, some are huge but the name of the game is simple – *networking*. In order to be a truly global provider, all of them have to have a data center, or a couple of servers at least, as close to you as possible. If you are reading this in one of the bigger cities in almost any big country

in the world, chances are there is an AWS, Microsoft, or Google-owned server in a radius of 100 miles from you. All the providers try to have at least one data center in every big city in every country since that can enable them to offer a range of services extremely quickly. This concept is called **Point of Contact** (**POC**) and means that when connecting to the provider's cloud, you just need to get to the nearest server, and after that, the cloud will make sure your services are as quick as possible.

But when we are talking about data centers that actually belong to Amazon or the others, we are still dealing with a large-scale operation. Here, numbers are in the hundreds, and their location is also a secret, mainly for security reasons. They all have a few things in common. They are a highly automated operation situated somewhere in the vicinity of a major power source, a major source of cooling, or a major data hub. Ideally, they would be placed in a spot that has all those three things, but that is usually not possible.

Placement is the key

Different companies have different strategies since choosing a good place to build a data center can mean a lot of cost savings. Some even go to extremes. Microsoft, for instance, has a data center completely submerged in the ocean to facilitate cooling.

When providing a service for a particular user, your main concern is usually speed and latency, and that in turn means that you want your server or your service to run in the data center that is closest to the user. For that purpose, all cloud providers divide their data centers geographically, which in turn enables administrators to deploy their services in the optimal part of the internet. But at the same time, this creates a typical problem with resources – there are places on the planet that have a small number of available data centers but are heavily populated, and there are places that are quite the opposite. This in turn has a direct influence on the price of resources. When we talked about pricing, we mentioned different criteria; now we can add another one – location. The location of a data center is usually given as a **region**. This means

that AWS, or any other provider for that matter, is not going to give you the location of their data center, but instead will say *users in this region would be best served by this group of servers*. As a user, you have no idea where the particular servers are, but instead, you only care about the region as given to you by the provider. You can find the names of service regions and their codes here:

Name	Code
US East (Ohio)	`us-east-2`
US East (N. Virginia)	`us-east-1`
US West (N. California)	`us-west-1`
US West (Oregon)	`us-west-2`
Asia Pacific (Hong Kong)	`ap-east-1`
Asia Pacific (Mumbai)	`ap-south-1`
Asia Pacific (Osaka-Local)	`ap-northeast-3`
Asia Pacific (Seoul)	`ap-northeast-2`
Asia Pacific (Singapore)	`ap-southeast-1`
Asia Pacific (Sydney)	`ap-southeast-2`
Asia Pacific (Tokyo)	`ap-northeast-1`
Canada (Central)	`ca-central-1`
Europe (Frankfurt)	`eu-central-1`
Europe (Ireland)	`eu-west-1`
Europe (London)	`eu-west-2`
Europe (Paris)	`eu-west-3`
Europe (Stockholm)	`eu-north-1`
Middle East (Bahrain)	`me-south-1`
South America (São Paulo)	`sa-east-1`

Figure 13.1 – Service regions on AWS with the names used in the configuration

Choosing a service provided by a region that is in heavy demand can be expensive, and that same service can be much cheaper if you choose a server that is somewhere else. This is the beauty

of the cloud – you can use the services that suit you and your budget.

Sometimes price and speed are not the most important things. For example, legal frameworks such as **GDPR**, a European regulation on personal data collection, processing, and movement, basically states that companies from Europe must use a data center in Europe since they are covered by the regulation. Using a US region in this case could mean that a company could be legally liable (unless the company running this cloud service is a part of some other framework that allows this – such as **Privacy Shield**).

AWS services

We need to talk a little bit about what AWS offers in terms of services since understanding services is one thing that will enable you to use the cloud appropriately. On AWS, all the available services are sorted into groups by their purpose. Since AWS has hundreds of services, the AWS management console, the first page you will see once you log in, will at first be a daunting place.

You will probably be using the AWS Free Tier for learning purposes, so the first step is to actually open an AWS Free account. Personally, I used my own personal account. For the Free account, we need to use the following URL: **https://aws.amazon.com/free/**, and follow along with the procedure. It just asks for a couple of pieces of information, such as email address, password, and AWS account name. It will ask you for credit card info as well, to make sure that you don't abuse the AWS account.

After signing up, we can log in and get to the AWS dashboard. Take a look at this screenshot:

Figure 13.2 – Amazon services

Every single thing here is a link, and they all point to different services or pages with subservices. Moreover, this screenshot shows only about a third of all available services. There is no point in covering them all in this book; we are just going to use three of all these to show how AWS connects to our KVM infrastructure, but once you get the hang of it, you will slowly begin to understand how everything connects and what to use in a particular moment. What really

helps is that AWS has great documentation, and that all the different services have provisioning wizards that help you find the thing you are looking for.

In this particular chapter, we are going to use these three services: **IAM**, **EC2**, and **S3**.

All of these are of course abbreviations, but other services just use project names, such as **CloudFront** or **Global Accelerator**. In any case, your first point of order should be to start using them, not just read about them; it is much easier to understand the structure once you put it to good use.

In this chapter, we used a Free account, and almost everything we did was free, so there is no reason for you not to try to use the AWS infrastructure yourself. AWS tries to be helpful there as much as it can, so if you scroll down on the console page, you will find these helpful icons:

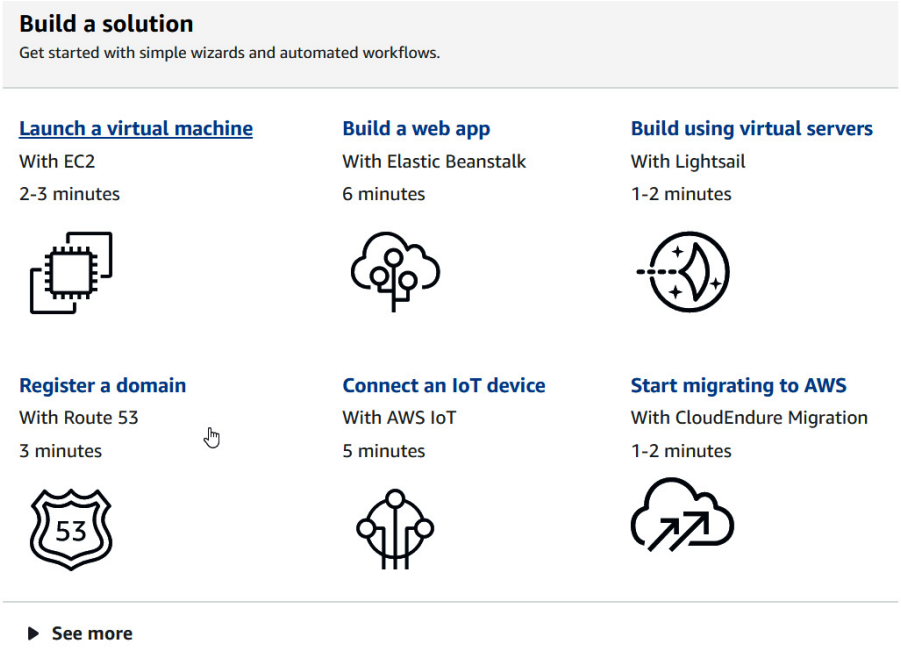

Figure 13.3 – Some AWS wizards, documentation, and videos – all very helpful

All of these are simple scenarios that will get you up and running in a couple of minutes, for free. Amazon realizes that first-time users of the cloud are overwhelmed with all the choices, so they try to get your first machine running in a couple of minutes to show you how easy it is.

Let's get you acquainted with the services we are going to use, which we're going to do by using a scenario. We want to migrate a machine that was running in our local KVM installation into Amazon AWS. We are going to go through the whole process step by step, but we first need to understand what we need. The first thing, obvi-

ously, is the ability to run virtual machines in the cloud. In the AWS universe, this is EC2 or Amazon Elastic Compute Cloud in full.

EC2

EC2 is one of the few real core services that basically runs everything there is to run in the AWS cloud. It is a scalable computing capacity provider for the whole infrastructure. It enables running different instances or virtual computing environments, using various configurations of storage, memory, CPU, and networking, and it also provides everything else those instances need, including security, storage volumes, zones, IP addresses, and virtual networks. Some of these services are also available separately in case you need more complex scenarios, for example, a lot of different storage options exist, but the core functionality for the instances is provided by EC2.

S3

The full name of this service is actually Amazon Simple Storage Service, hence the name *Amazon*

S3. The idea is to give you the ability to store and retrieve any amount of data, anytime you need it, using one or more of the methods offered. The most important concept we are going to use is an *S3 bucket*. A bucket is a logical storage element that enables you to group objects you store. Think of it as a name for a storage container you will later use to store things, whatever those things may be. You can name your buckets however you want, but there is a thing we must point out – the names of buckets have to be *globally unique*. This means that when you name a bucket, it must have a name that is not repeated anywhere else in any of the regions. This makes sure that your bucket will have a unique name, but it also means that trying to create a generic-sounding name such as **`bucket1`** or **`storage`** is probably not going to work.

Once you create a bucket, you can upload and download data from it using the web, a CLI, or an API. Since we are talking about a global system, we must also point out that data is stored in the region you specify when creating the bucket,

and is kept there unless you specify you want some form of multi-region redundancy. Have that in mind when deploying buckets, since once you start using the data in the bucket, your users or your instances need to get the data, and latency can become a problem. Due to legal and privacy concerns, data never leaves your dedicated region unless you explicitly specify otherwise.

A bucket can store any number of objects, but there is a limit of 100 buckets per account. If that is not enough, you can request (and pay) to have that limit raised to 1,000 buckets.

Also, take a close look at other different options for storing and moving data – there are different types of storage that may or may not fit your needs and budget, such as, for example, S3 Glacier, which offers much cheaper options for storing large amounts of data, but is expensive if you need to get the data out.

IAM

AWS **Identity and Access Management (IAM)** is the service we need to use since it enables access management and permissions for all the objects and services. In our example, we are going to use it to create policies, users, and roles necessary to accomplish our task.

Other services

There is simply no way to mention all the services AWS offers in simple form. We mentioned only the ones that were necessary and tried to point you in the right direction. It is up to you to try and see what your usage scenario is, and how to configure whatever satisfies your particular needs.

So far, we have explained what AWS is and how complex it can become. We have also mentioned the most commonly used parts of the platform and started explaining what their functions are. We are going to expand on that as we actually migrate a machine from our local environment into AWS. This is going to be our next task.

Preparing and converting virtual machines for AWS

If you search for it on Google, migrating machines from KVM to AWS is easy, and all that is required is to follow the instructions at this link: **https://docs.amazonaws.cn/en_us/vm-import/latest/userguide/vm-import-ug.pdf**

If you actually try to do it, you will quickly understand that, given *basic* knowledge of the way AWS works, you will not be able to follow the instructions. This is why we choose to do this simple task as an example of using AWS to quickly create a working VM in the cloud.

What do we want to do?

Let's define what we are doing – we decided to migrate one of our machines into the AWS cloud. Right now, our machine is running on our local KVM server, and we want it running on AWS as soon as possible.

The first thing we must emphasize is that there is no live migration option for this. There is no simple tool that you can point to the KVM machine and move it to AWS. We need to do it step by step, and the machine needs to be off. After quickly consulting the documentation, we created a plan. Basically, what we need to do is the following:

1. Stop our virtual machine.
2. Convert the machine to a format that is compatible with the import tool used in AWS.
3. Install the required AWS tools.
4. Create an account that will be able to do the migration.
5. Check whether our tools are working.
6. Create an S3 bucket.
7. Upload the file containing our machine into the bucket.
8. Import the machine to EC2.
9. Wait for the conversion to finish.
10. Prepare the machine to start.
11. Start the machine in the cloud.

So, let's start working on that:

1. A good place to start is by taking a look at our machines on our workstation. We will be migrating the machine named **`deploy-1`** to test our AWS migration. It's a core installation of CentOS 7 and is running on a host using the same Linux distribution. For that, we obviously need to have privileges:

```
[cloud@workstation ~]$ virsh list
 Id    Name                           State
----------------------------------------------------

[cloud@workstation ~]$ sudo su
[sudo] password for cloud:
[root@workstation cloud]# virsh list
 Id    Name                           State
----------------------------------------------------
 1     deploy-1                       running
```

   Figure 13.4 – Selecting a VM for our migration process

   The next thing to do is to stop the machine – we cannot migrate machines that are running since we need to convert the volume that the machine is using in order to make it compatible with the import tool on EC2.

2. The documentation available at **https://docs.aws.amazon.com/vm-import/latest/userguide/vmimport-image-import.html** states that:

   "When importing a VM as an image, you can import disks in the following formats: Open Virtualization Archive (OVA), Virtual Machine

Disk (VMDK), Virtual Hard Disk (VHD/VHDX), and raw. With some virtualization environments, you would export to Open Virtualization Format (OVF), which typically includes one or more VMDK, VHD, or VHDX files, and then package the files into an OVA file."

In our particular case, we are going to use the **`.raw`** format, since it is compatible with the import tool, and is fairly simple to convert from the **`.qcow2`** format KVM uses, into this format. Once our machine has been stopped, we need to do the conversion. Find the image on disk and use **`qemu-img`** to do the conversion. The only parameter is the files; the converter understands what it needs to do by detecting the extensions:

```
[root@workstation deploy-1]# ls
centos1.qcow2  deploy-1-cidata.iso  meta-data  user-data
[root@workstation deploy-1]# qemu-img convert centos1.qcow2 deploy1.raw
[root@workstation deploy-1]# ls
centos1.qcow2  deploy-1-cidata.iso  deploy1.raw  meta-data  user-data
[root@workstation deploy-1]#
```

Figure 13.5 – Converting a qcow2 image to raw image format

We only need to convert the image file, containing the disk image for the system; other data was left out of the installation of the VM.

We need to have in mind that we are converting to a format that has no compression so your file size can significantly increase:

```
[root@workstation deploy-1]# ls -al
total 941644
drwxr-xr-x. 2 root root        107 Apr 11 01:43 .
drwx--x--x. 6 root root        158 Jan 13 16:52 ..
-rw-r--r--. 1 root root   43188224 Apr 11 01:27 centos1.qcow2
-rw-r--r--. 1 qemu qemu     374784 Jan 12 18:51 deploy-1-cidata.iso
-rw-r--r--. 1 root root 8589934592 Apr 11 01:44 deploy1.raw
-rw-r--r--. 1 root root         26 Jan 12 16:27 meta-data
-rw-r--r--. 1 root root        629 Jan 12 17:42 user-data
[root@workstation deploy-1]#
```

Figure 13.6 – The conversion process and the corresponding capacity change

We can see that our file increased from 42 MB to 8 GB just because we had to remove the advanced features **`qcow2`** offers for data storage. The free tier offers only 5 GB of storage, so please make sure to configure the raw image size correspondingly.

Our next obvious step is to upload this image to the cloud since the conversion is done there. Here, you can use different methods, either GUI or CLI (API is also a possibility but is way too complicated for this simple task).

3. AWS has a CLI tool that facilitates working with services. It's a simple command-line tool compatible with most, if not all, the operating systems you can think of:

```
[root@workstation ~]# curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/in  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0st100 31.1M  100 31.1M
[root@workstation ~]#
[root@workstation ~]# unzip awscliv2.zip
```

Figure 13.7 – Downloading and uncompressing AWS CLI

We're using **`curl`** to download a file, and its **`-o`** option to say what the name of the output file is going to be. Obviously, we need to unzip the ZIP file so that we can use it. The installation process of the tool is also referenced in the documentation. We are talking about a simple download, after which we have to extract the tool. Since there is no installer, the tool will not be in our path, so from now on, we need to reference it by the absolute path.

Before we can use the AWS CLI, we need to configure it. This tool has to know how it is going to connect to the cloud, which user it's going to use, and has to have all the permissions granted in order for the tool to be able to get the data uploaded to AWS, and then imported and converted into the EC2 image. Since we do not have that configured, let's switch to the GUI on AWS and configure the things we need.

Important note

From now on, if something looks edited in the screenshots, it probably is. To enable things to work seamlessly, AWS has a lot of personal and account data on the screen.

4. We will go into **Identity and Access Management** or IAM, which looks like the following screenshot. Go to **Services** | **Security**, click on **Identity & Compliance**, and click on **IAM**. This is what you should see:

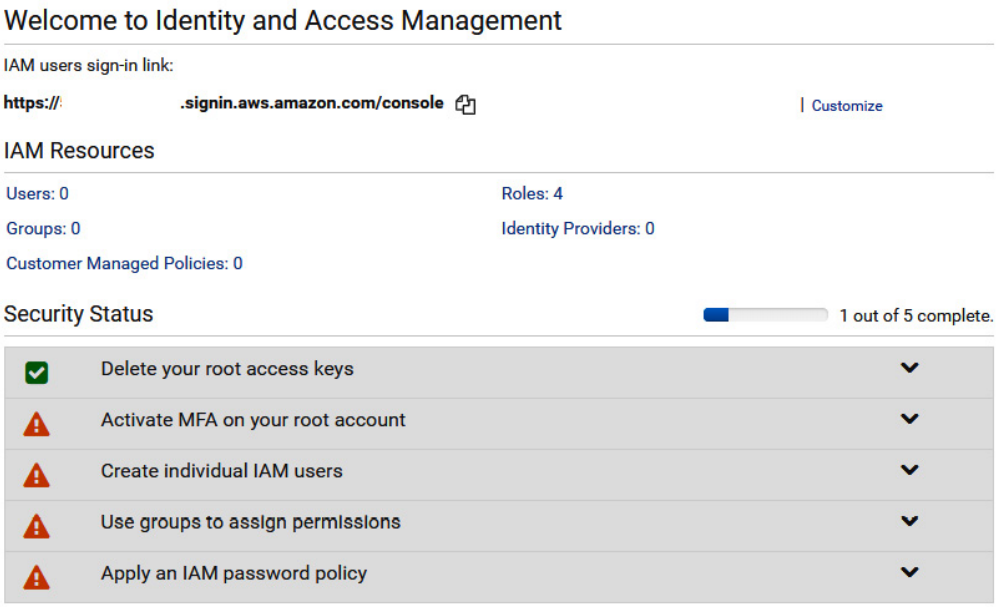

Figure 13.8 – IAM console

We need to choose users on the left side of the screen. This will give us access to the user console. We will create a user named **`Administrator`**, a group named **`administrators`**, and apply appropriate permissions to them. Then we are going to join the user to the

group. On the first screen, you can choose both options, **Programmatic access** *and* **AWS Management Console access**. The first one enables you to use the AWS CLI, and the second one enables the user to log in to the management console if we need this account to configure something. We choose only the first one but will add the API key later:

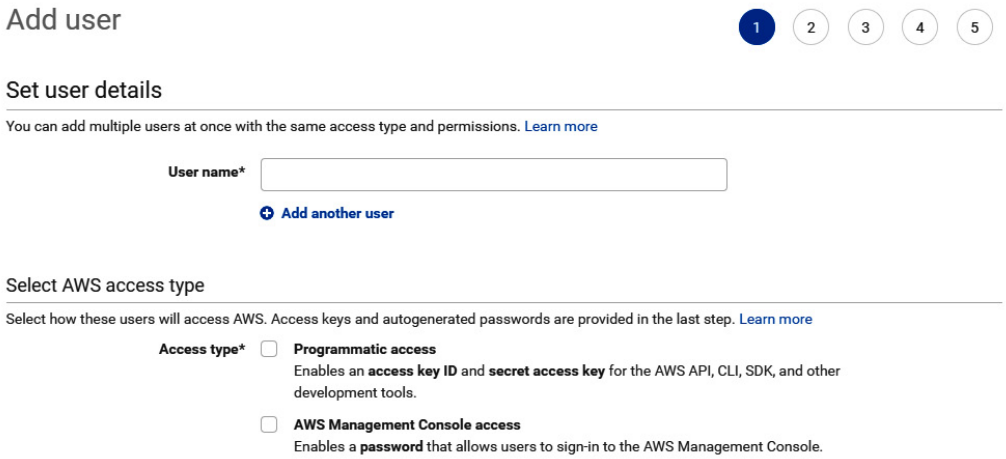

Figure 13.9 – Configuring user permissions

After clicking on the appropriate option, we can set the initial password for the user. This user will have to change it as soon as they log in:

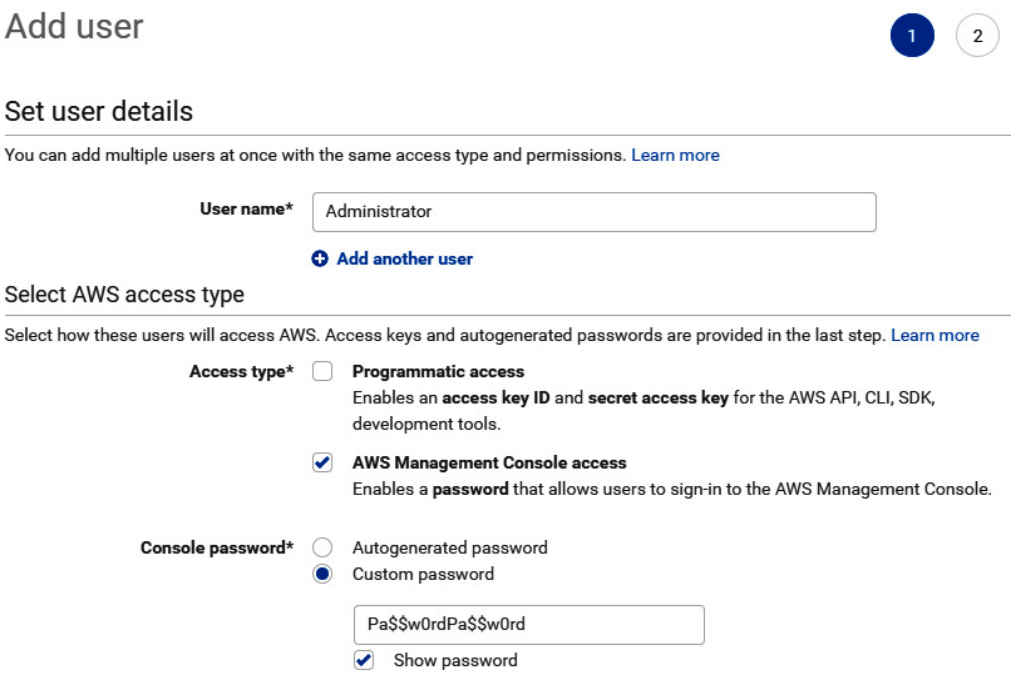

Figure 13.10 – Setting an initial password to be changed later

We will also create a group for this user. Do that by choosing the appropriate button in the upper part of the screen:

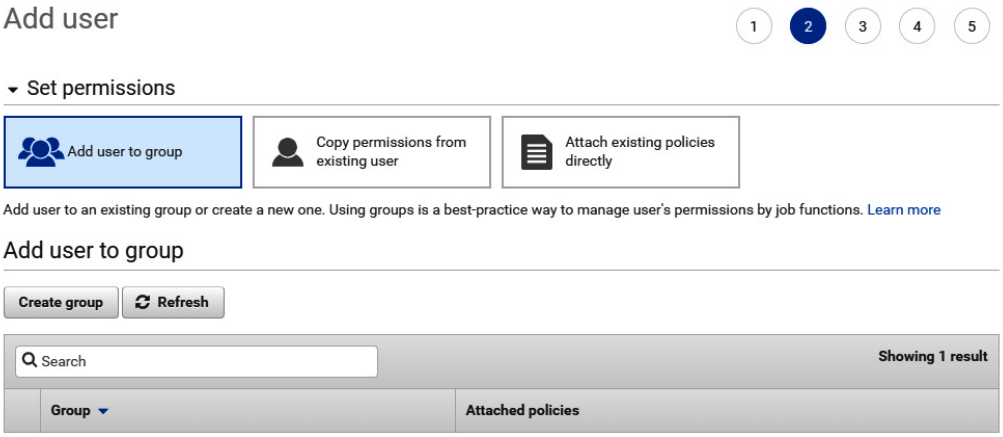

Figure 13.11 – Creating a group for our user

We can assign appropriate policies directly to the user, but having policies assigned to groups, and then assigning users to appropriate groups is a better option, saving a lot of

time when we need to remove some permissions from users. Once you click the **Create group** button on the left, you will be able to name and create the group. Below the name box are a lot of predefined policies that we can use to configure a strict user policy. We can also create custom policies, but we are not going to do that:

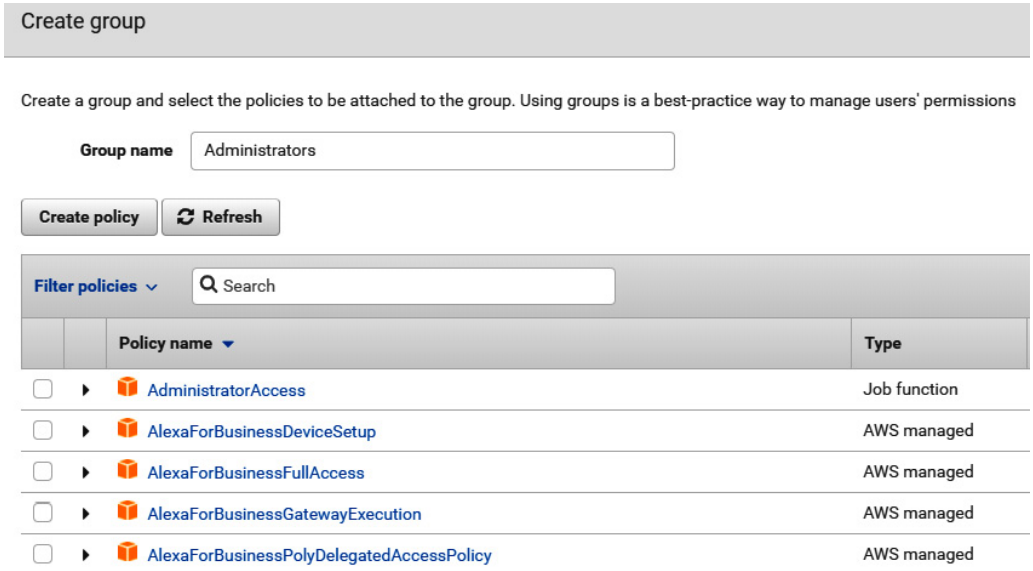

Figure 13.12 – Create a group wizard and policies

For this to work, we are going to create a group with permissions that are way over the top for this task. We are essentially giving the user all the permissions across the cloud. Filter the policies by **AWS managed – job function**:

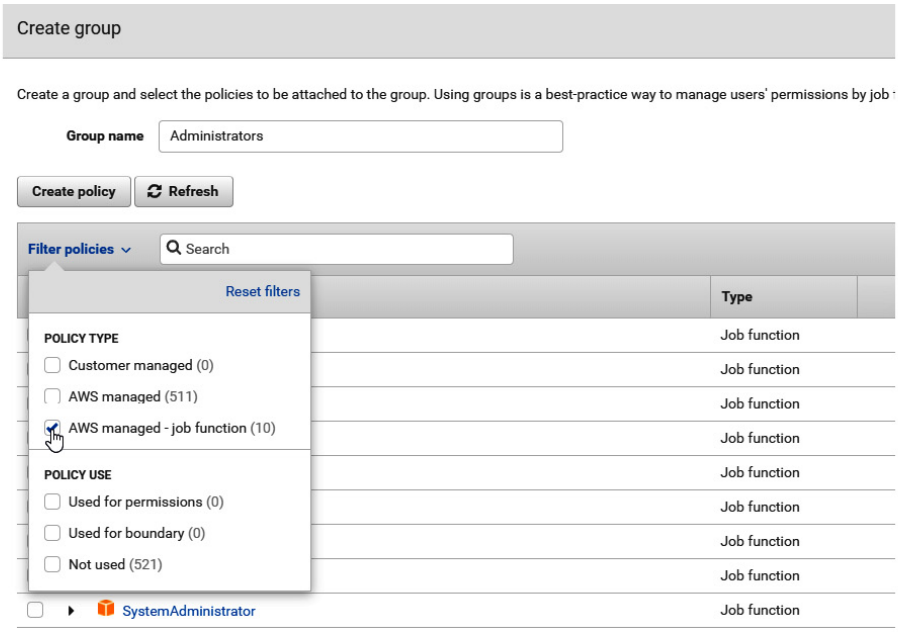

Figure 13.13 – Filtering policies

We are going to use the **`AdministratorAccess`** policy for this example. This policy is very important, as it allows us to give all available permissions to the **`Administrators`** group that we're creating. Now select **AdministratorAccess** and click **Create group** in the lower right of the screen:

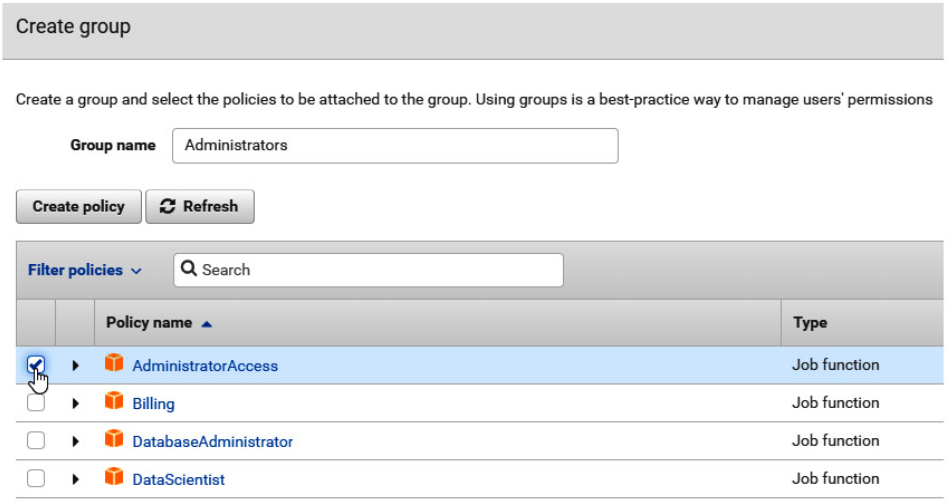

Figure 13.14 – Selecting a policy for our group

The next step is the tags: you can create different attributes or **`tags`** that can be used later for identity management.

5. Tagging can be done by using almost anything – name, email, job title, or whatever you need. We are going to leave this empty:

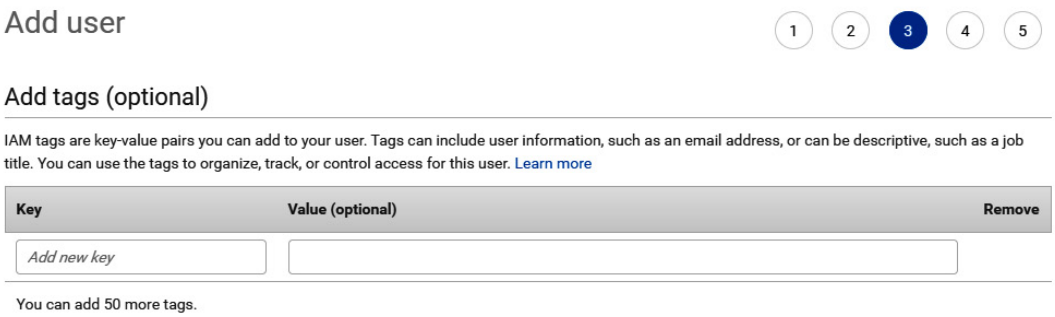

Figure 13.15 – Adding tags

Let's review what we have configured so far. Our user is a member of the group we just created, and they have to reset the password as soon as they log in:

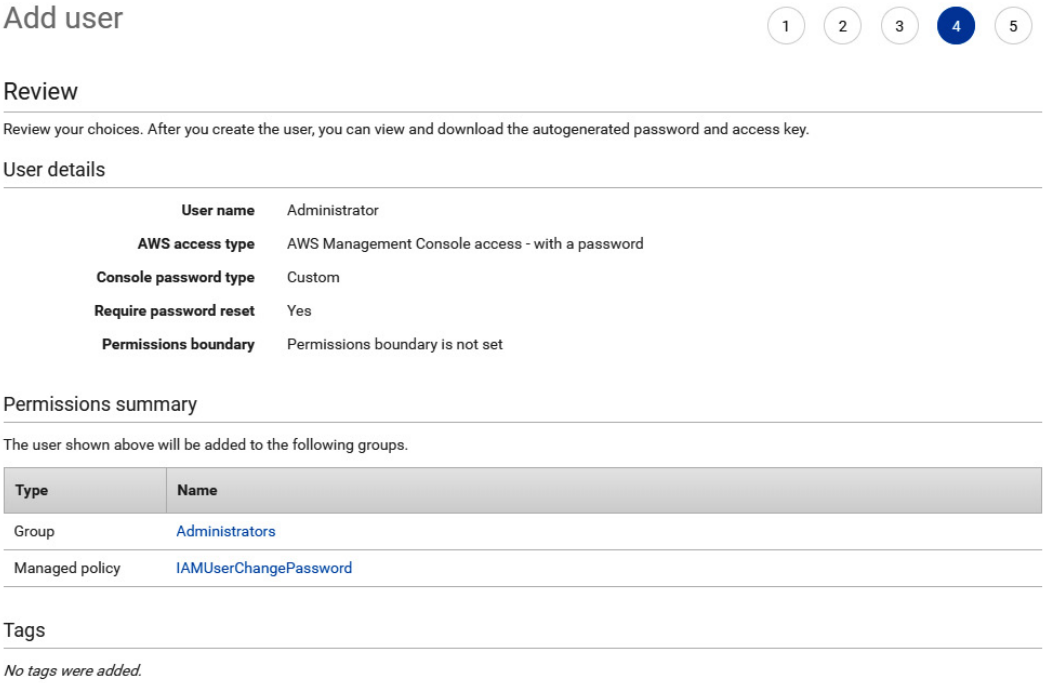

Figure 13.16 – Reviewing the user configuration with group, policy, and tag options

Accept these and add the user. You should be greeted with a reassuring green message box that will give you all the relevant details about what just happened. There is also a direct link to the console for management access, so you can share that with your new user:

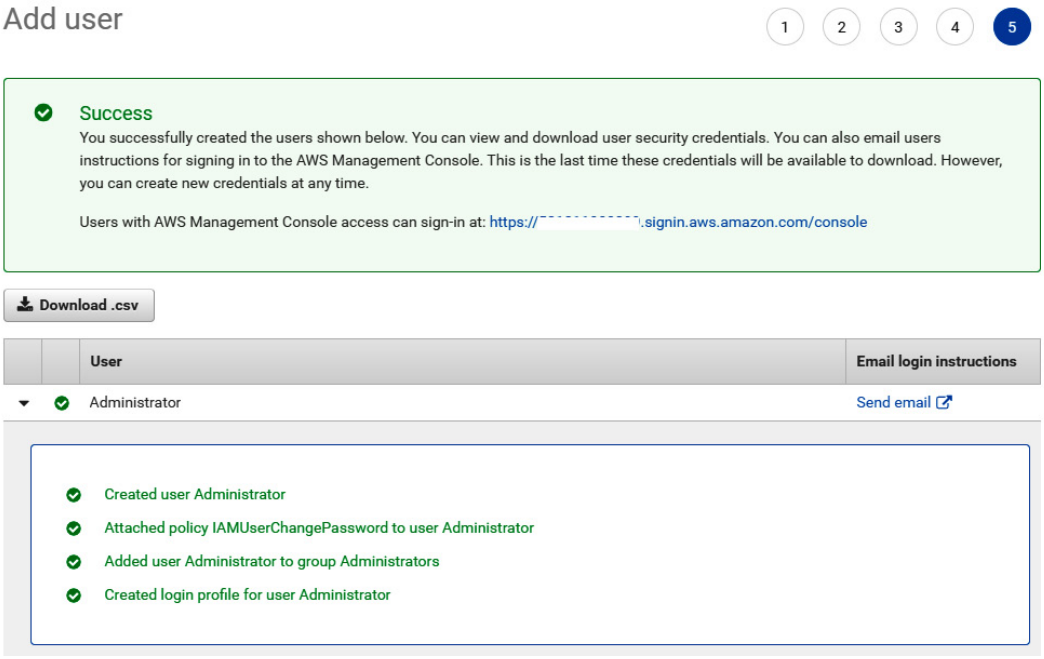

Figure 13.17 – User creation was successful

Once the user has been created, we need to enable their **`Access Key`**. This is a normal concept in using different command-line utilities. It enables us to provide a way for an application to do something as a given user, and not give the application the username or the password. At the same time, we can give each application its own key, so when we want to revoke access, we can simply disable the key.

Click on **Create access key** in the middle of the screen:

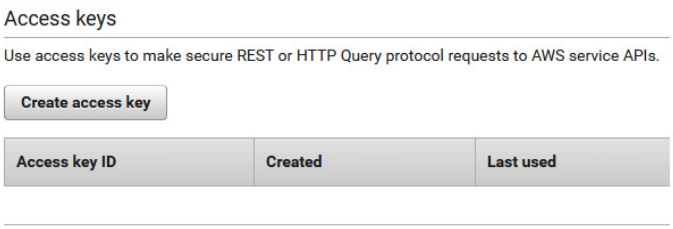

Figure 13.18 – Creating an access key

A couple of things need to be said about this key. There are two fields – one is the key itself, which is **`Access key ID`**, the other is the secret part of the key, which is **`Secret access key`**. In regard to security, this is completely the same as having a username and password for a particular user. You are given only one opportunity to see and download the key, and after that, it is gone. This is because we are dealing with hashed information here, and AWS is *not* storing your keys, but hashes of them. This means there is no way to retrieve a key if you didn't save it. It also means if somebody grabs a key, let's say by reading it off a screenshot, they can identify themselves as the user that has the key assigned. The good thing is that you can create as many keys as you want and revoking them is only a question of deleting them here. So, save the key somewhere safe:

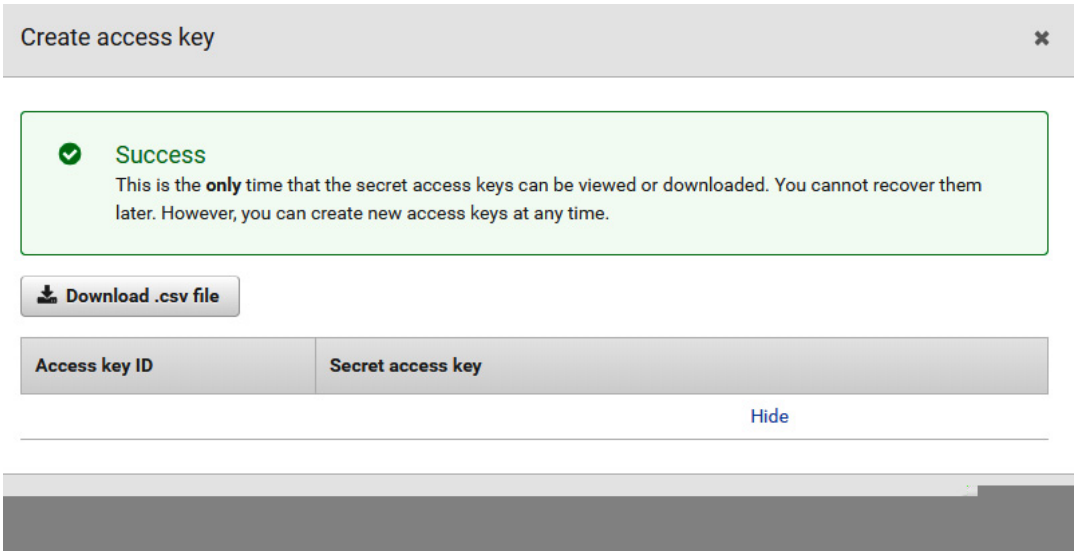

Figure 13.19 – Access key was created successfully

We are finished with the GUI for now. Let's go back and install the AWS CLI:

1. We just need to start the installation script and let it finish its job. This is done by starting the file called **`install`** in the **`aws`** directory:

```
 inflating: aws/dist/botocore/data/appconfig/2019-10-09/paginators-1.json
  creating: aws/dist/botocore/data/events/2015-10-07/
 inflating: aws/dist/botocore/data/events/2015-10-07/service-2.json
 inflating: aws/dist/botocore/data/events/2015-10-07/examples-1.json
 inflating: aws/dist/botocore/data/events/2015-10-07/paginators-1.json
  creating: aws/dist/botocore/data/comprehendmedical/2018-10-30/
 inflating: aws/dist/botocore/data/comprehendmedical/2018-10-30/service-2.json
 inflating: aws/dist/botocore/data/comprehendmedical/2018-10-30/paginators-1.json
root@workstation ~]#
root@workstation ~]# sudo ./aws/install
ou can now run: /usr/local/bin/aws --version
root@workstation ~]# aws
ash: aws: command not found...
root@workstation ~]# /usr/local/bin/aws --version
ws-cli/2.0.7 Python/3.7.3 Linux/3.10.0-1062.el7.x86_64 botocore/2.0.0dev11
root@workstation ~]#
root@workstation ~]#
root@workstation ~]#
root@workstation ~]# /usr/local/bin/aws configure
WS Access Key ID [None]:
WS Secret Access Key [Nor
efault region name [None]: us-west-2
efault output format [None]: table
root@workstation ~]#
```

Figure 13.20 – Installing AWS CLI

Remember what we said about absolute paths? The **`aws`** command is not in the user path; we need to call it directly. Use **`configure`** as a parameter. Then, use the two parts of the key we saved in the previous step. From now on, every command we give using the AWS CLI is interpreted as having been run as the user **`Administrator`** that we just created on the cloud.

The next step is to create a bucket on S3. This can be done in one of two ways. We can do it through our newly configured CLI, or we can use the GUI. We are going to take the "pretty" way and use the GUI in order to show how it looks and behaves.

2. Select S3 as the service in the console. There is a button at the top right labeled **Create bucket** – click it. The following screen will appear.

   Now create a bucket that is going to store your virtual machine in its raw format. In our case, we labeled the bucket **`importkvm`** but choose a different name. Make sure that you take note of the **`region`** pull-down menu – this is the AWS location where your resource will be cre-

ated. Remember that the name has to be unique; if everybody who bought this book tried to use this name, only the first one would succeed. Fun fact: if by the time you read this, we haven't deleted this bucket, nobody will be able to create another with the same name, and only those of you reading this exact sentence will understand why. This wizard is quite big in terms of screen estate and might not fit on a single book page, so let's split it into two parts:

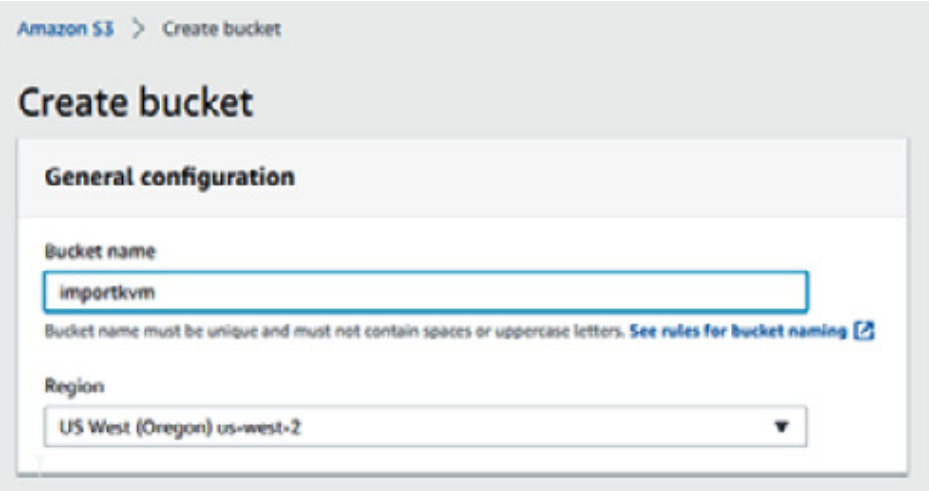

Figure 13.21 – Wizard for creating an S3 bucket – selecting bucket name and region

The second part of this wizard is related to settings:

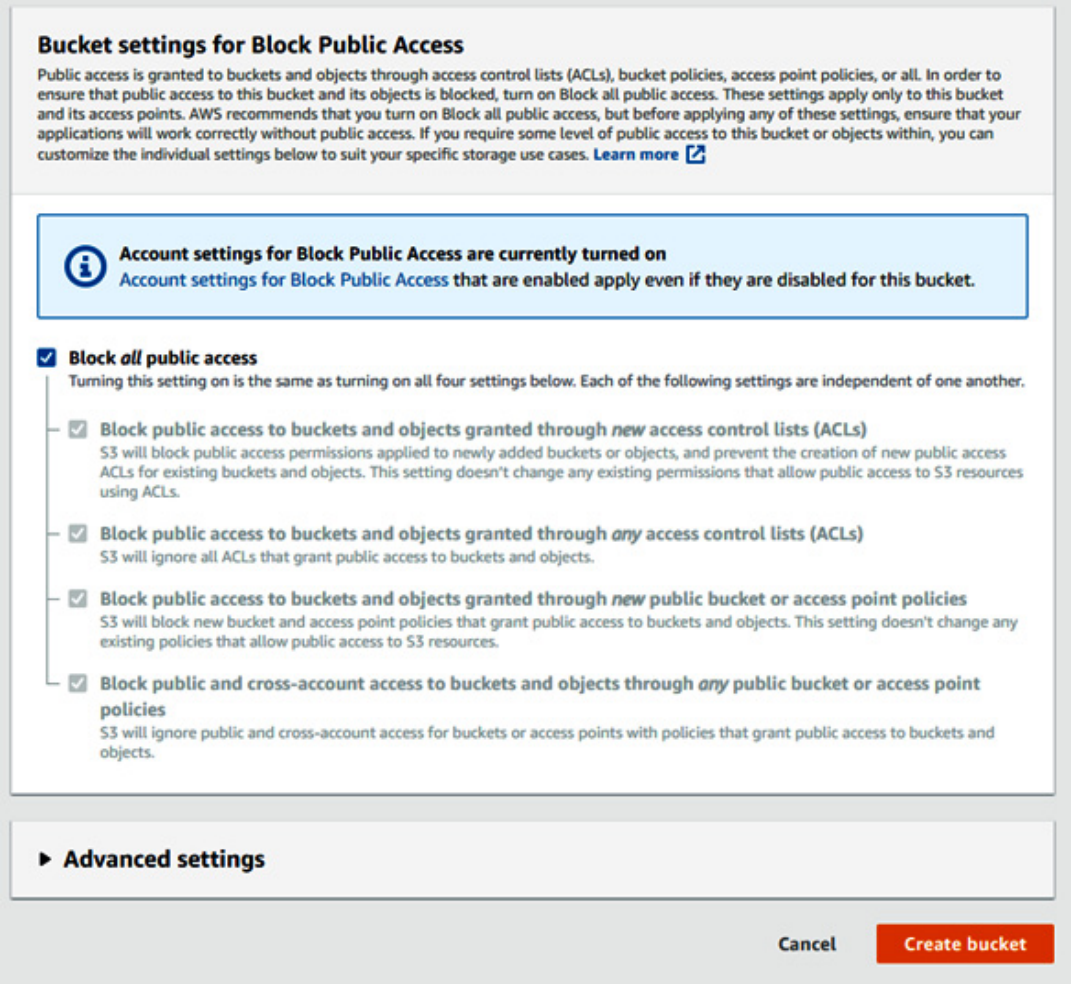

Figure 13.22 – Bucket settings

Do not change the access to public – there is really no need to; nobody but you is ever going to need to access this particular bucket and the file in it. By default, this option is pre-selected, and we should leave it as it is. This should be the end result:

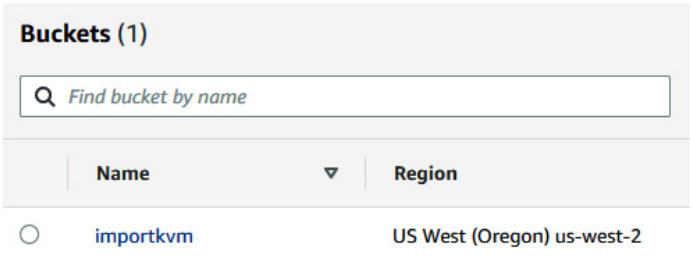

Figure 13.23 – S3 bucket created successfully

OK, having done that, it's time for some waiting on the next command to finish. In the next step, we are going to use our AWS CLI to copy the `.raw` file onto S3.

Important note

Depending on the type of account, from this point on, it is possible that we will have to pay for some of the services that we create since they may over-draft the free tier enabled on your account. If you do not enable anything expensive, you should be fine, but always take look at your **Cost management** *dashboard, and check that you are still in the black.*

Uploading an image to EC2

The next step is to upload an image to EC2 so that we can actually run that image as a virtual machine. Let's start the upload process – this is why we installed the AWS CLI utility in the first place:

1. Use the AWS CLI with the following parameters:

```
[root@workstation deploy-1]# /usr/local/bin/aws s3 cp deploy1.raw s3://importkvm
upload: ./deploy1.raw to s3://importkvm/deploy1.raw
[root@workstation deploy-1]#
```

Figure 13.24 – Using the AWS CLI to copy a virtual machine raw image to an S3 bucket

That's the end result. Since we are talking about 8 GB of data, you will have to wait for some time, depending on your upload speed. The syntax for the AWS CLI is pretty straightforward. You can use most Unix commands that you know, both **`ls`** and **`cp`** do their job. The only thing to remember is to give your bucket name in the following format as the destination: **`s3://<bucketname>`**.

2. After that, we do an **`ls`** – it will return the bucket names, but we can list their contents by using the bucket name. In this example, you can also see it took us something like 15 minutes to transfer the file from the moment we created the bucket:

```
[root@workstation deploy-1]# /usr/local/bin/aws s3 ls
2020-04-11 01:34:14 importkvm
[root@workstation deploy-1]# /usr/local/bin/aws s3 ls importkvm
2020-04-11 01:48:59 8589934592 deploy1.raw
[root@workstation deploy-1]#
```

Figure 13.25 – Transferring the file

And now starts the fun part. We need to import the machine into EC2. To do that, we need to do a few things before we will be able to do the conversion. The problem is related to permissions – AWS services are unable to talk to each other by default. Therefore, you have to give explicit permission to each of them to do the importing. In essence, you have to let EC2 talk to S3 and get the file from the bucket.

3. For upload purposes, we will introduce another AWS concept – **`.json`** files. A lot of things in AWS are stored in **`.json`** format, including all the settings. Since the GUI is rarely used, this is the quickest way to communicate data and settings, so we must also use it. The first file we need is **`trust-policy.json`**, which we are using to create a role that will enable the data to be read from the S3 bucket:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
```

```
            "Service":"vmie.amazona
ws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "sts:ExternalId":"vm
import"
            }
         }
      }
   ]
}
```

Just create a file with the name **trust-policy.json**, and get the preceding code typed in. Do not change anything. The next one up is the file named **role-policy.json**. This one has some changes that you have to make. Take a closer look inside the file and find the lines where we mention our bucket name (**im-portkvm**). Delete our name and put the name of your bucket instead:

```
{
   "Version":"2012-10-17",
```

```
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetBucketLocation",
                 "s3:ListBucket",
                 "s3:GetObject"
         ],
         "Resource":[
            "arn:aws:s3:::importkvm
"
                 "arn:aws:s3:::impo
rtkvm/*"
         ],
      },
      {
         "Effect":"Allow",
         "Action":[
            "s3:GetObject",
                 "s3:GetBucketLocat
ion",
                 "s3:ListBucket",
                 "s3:GetBucketAcl",
                 "s3:PutObject"
```

```
            ],
            "Resource":[
               "arn:aws:s3:::importkvm
"
                     "arn:aws:s3:::impo
rtkvm/*"
            ],
         },
         {
            "Effect":"Allow",
            "Action":[
               "ec2:ModifySnapshotAttr
ibute",
               "ec2:CopySnapshot",
               "ec2:RegisterImage",
               "ec2:Describe*"
            ],
            "Resource":"*"
         }
      ]
}
```

Now it's time to put it all together and finally upload our virtual machine to AWS.

4. Execute these two commands, disregard whatever happens in the formatting – both of them are one-liners, and the filename is the last part of the command:

```
/usr/local/bin/aws iam create-role
--role-name vmimport --assume-role-
policy-document  file://trust-
policy.json
/usr/local/bin/aws iam put-role-
policy --role-name vmimport --
policy-name vmimport --policy-
document file://role-policy.json
```

You should get a result something like this:

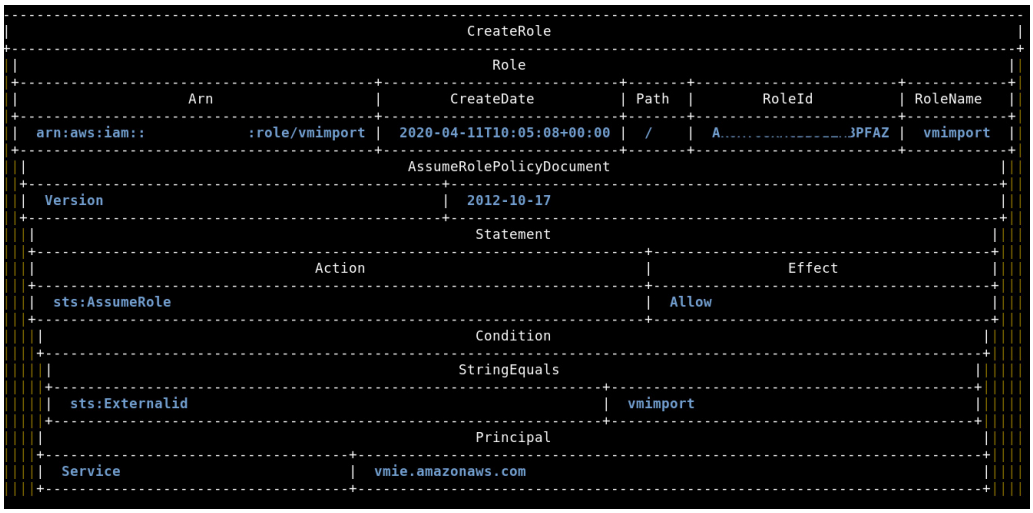

Figure 13.26 – Result of createrole

This confirms that the role was given the permissions it needs. The second command should not return any output.

We're almost done. The last step is to create yet another **.json** file that will describe to EC2 what we are actually importing and what to do with it.

5. The file we're creating needs to look like this:

```
[
{
"Description": "Test deployment",
"Format": "raw",
"Userbucket": {
"S3Bucket": "importkvm",
"S3Key": "deploy1.raw"
}
]
```

As you can see, there is nothing special in the file, but when you create your own version, pay attention to use your name for the bucket and the disk image that is stored inside the bucket. Name the file whatever you want, and use that name to call the import process:

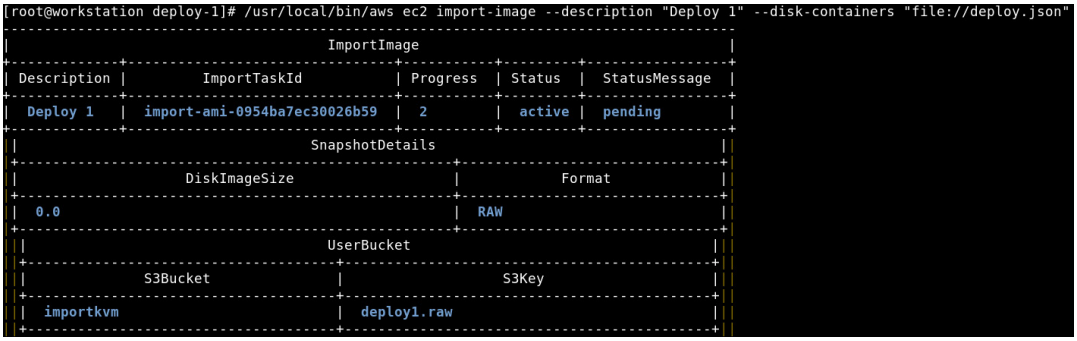

Figure 13.27 – Final step – virtual machine deployment to AWS

Now you wait for the process to finish. What happens in this step is both the import and conversion of the image and the operating system you uploaded. AWS is not going to run your image as is; the system is going to change quite a few things to make sure your image can run on the infrastructure. Some users will also receive some changes, but more on that later.

The task will run in the background, and will not notify you when it completes; it is up to you to check on it. Luckily, there is a command that can be used in the AWS CLI called **`describe-import-image-tasks`** and this is the output:

```
|                                                DescribeImportImageTasks                                                 |
+-------------------------------------------------------------------------------------------------------------------------+
||                                                ImportImageTasks                                                       ||
|+--------------+-------------+----------------------+-----------------------------+-------------+-----------+-----------+|
|| Architecture | Description |       ImageId        |        ImportTaskId         | LicenseType | Platform  |  Status   ||
|+--------------+-------------+----------------------+-----------------------------+-------------+-----------+-----------+|
||  x86_64      |  Deploy 1   | ami-06487af0f1c31c829 | import-ami-0954ba7ec30026b59 |  BYOL       |  Linux    | completed ||
|+--------------+-------------+----------------------+-----------------------------+-------------+-----------+-----------+|
||                                                SnapshotDetails                                                        ||
||+--------------------+---------------+-------------------+-----------+---------------------------+-----------------+||
|||     Description    |  DeviceName   |  DiskImageSize    |  Format   |        SnapshotId         |     Status      |||
||+--------------------+---------------+-------------------+-----------+---------------------------+-----------------+||
|| Test deployment     | /dev/sda1     | 8589934592.0      | RAW       | snap-0e3cce75ff798ad46    | completed       |||
||+--------------------+---------------+-------------------+-----------+---------------------------+-----------------+||
|||                                                UserBucket                                                         |||
|||+-----------------------------------------------+-------------------------------------------------------------+|||
|||                  S3Bucket                     |                            S3Key                            |||
|||+-----------------------------------------------+-------------------------------------------------------------+|||
|||  importkvm                                    |  deploy1.raw                                                |||
|||+-----------------------------------------------+-------------------------------------------------------------+|||
[root@workstation deploy-1]#
```

Figure 13.28 – Checking the status of our upload process

What this means is that we successfully imported our machine. Great! But the machine is still not running. Now it has become something called an **Amazon Machine Image** (**AMI**). Let's check how to use that:

1. Go to your EC2 console. You should be able to find the image under **AMIs** on the left side:

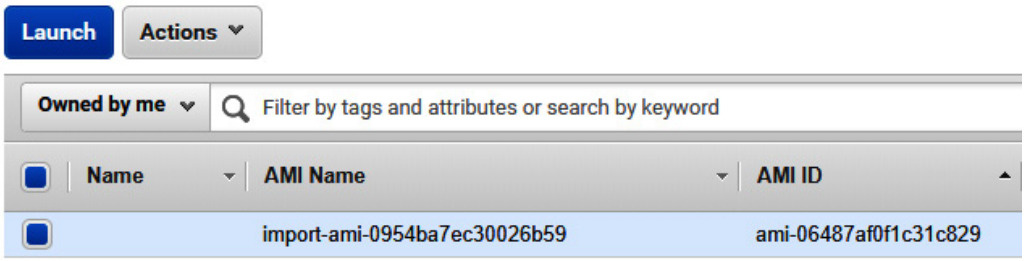

Figure 13.29 – Our AMI has been uploaded successfully and we can see it in the EC2 console

Now click the big blue **Launch** button. There are a couple of steps you need to finish before your instance is running, but we are almost there. First, you need to choose your instance

type. This means choosing what configuration fits your needs, according to how much of everything (CPU, memory, and storage) you need.

2. If you are using a region that is not overcrowded, you should be able to spin a *free tier* instance type that is usually called **`t2.micro`** and is clearly marked. In your free part of the account, you have enough processing credits to enable you to run this machine completely free:

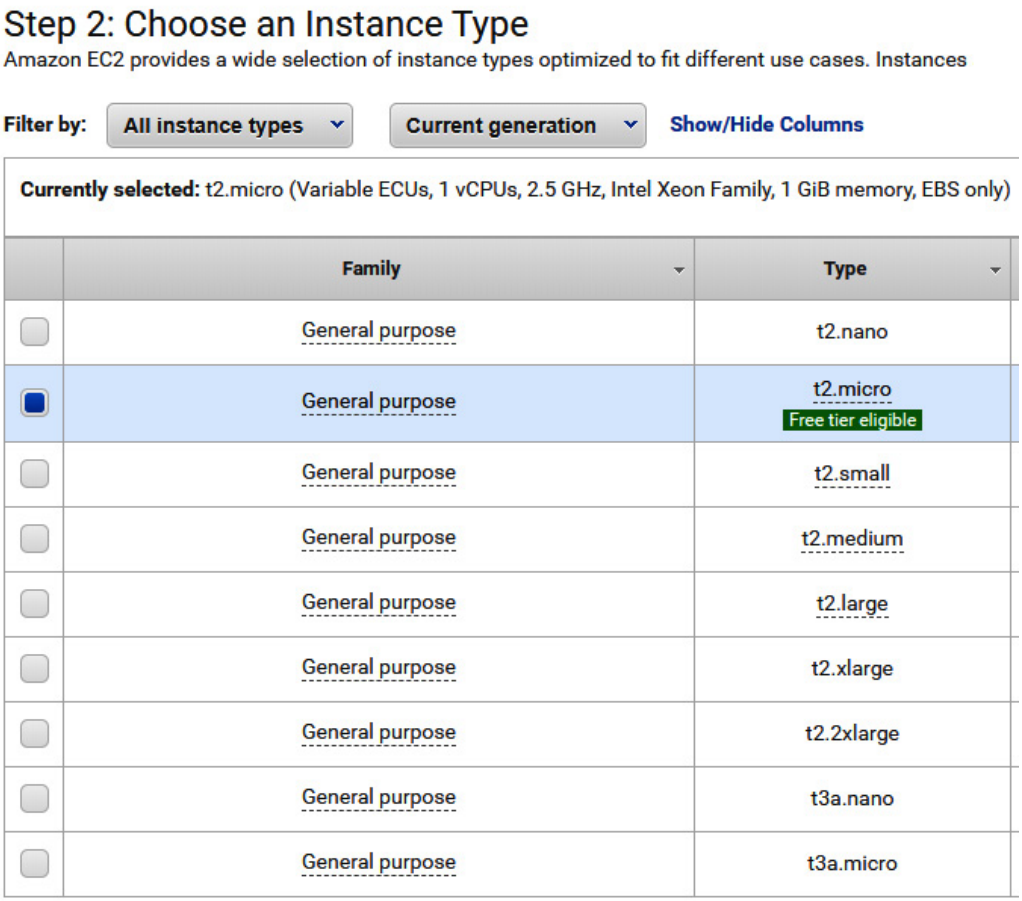

Figure 13.30 – Selecting an instance type

And now for some security. Amazon changed your machine and has implemented password-

less login to the administrator account using a key pair. Since we don't have a key yet, we will also need to create the key pair.

3. EC2 is going to put this key into the appropriate accounts on the machine you are just creating (all of them), so you can log in without using the password. A key pair is generated if you choose to do so, but Amazon will not store it – you have to do that:

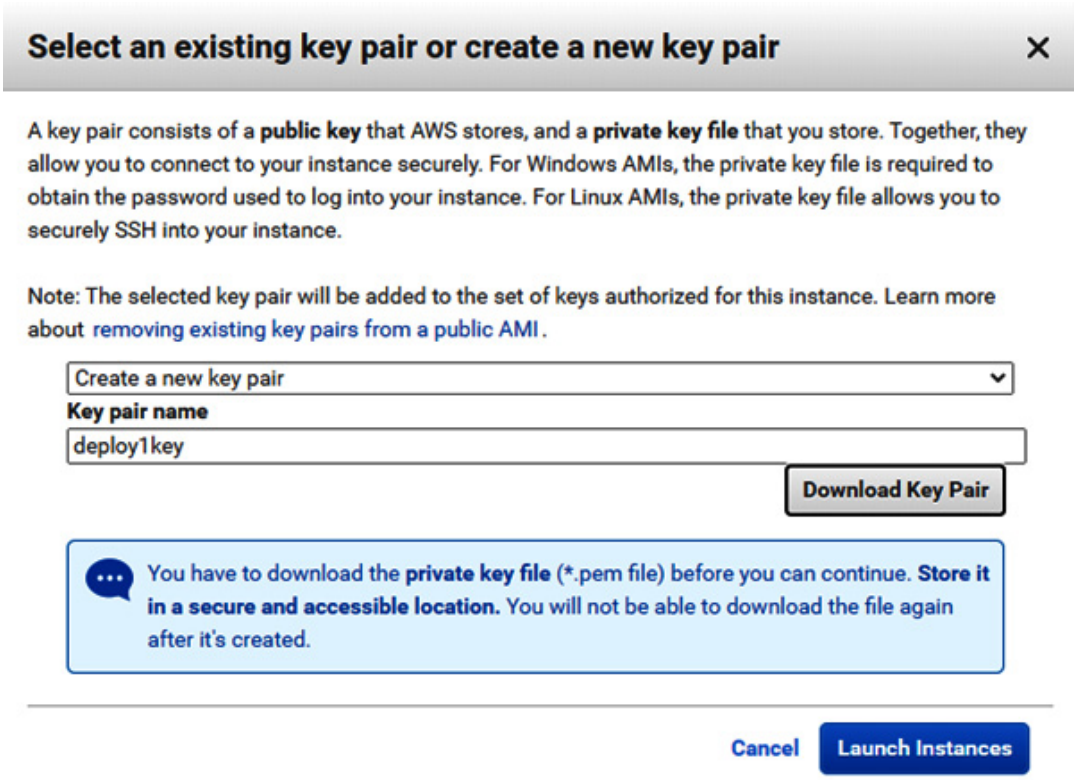

Figure 13.31 – Selecting an existing key or creating a new one

That's it, your VM should now take a couple of minutes to launch. Just wait for the confirmation

window. Once it is ready, connect to it using the context menu. You will get to the list of instances by clicking **View Instances** at the bottom right.

To connect, you need to use the key pair provided to you, and you need an **`ssh`** client. Alternatively, you can use the embedded **`ssh`** that AWS provides. In any case, you need the outside address of the machine, and AWS also provides that, along with simple instructions:

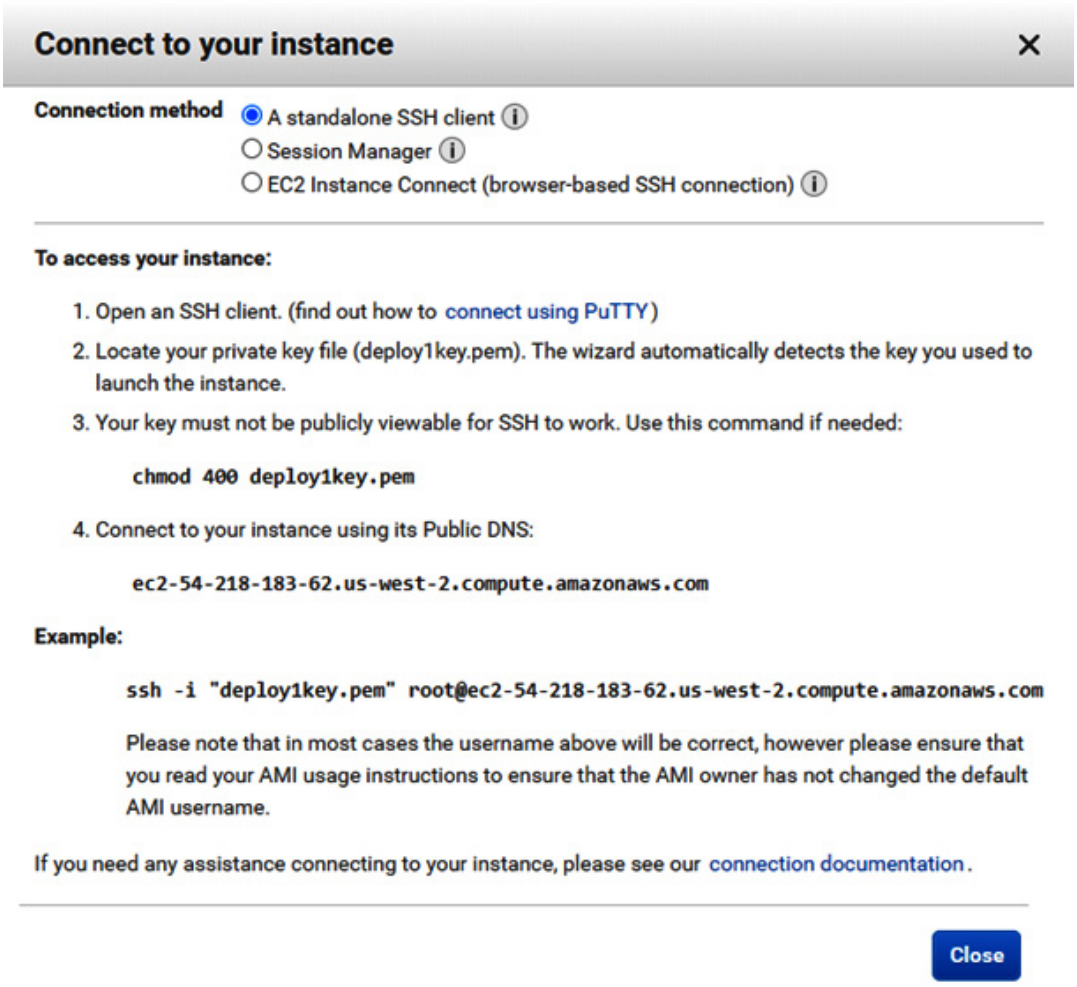

Figure 13.32 – Connect to your instance instructions

So, going back to our workstation, we can use the **ssh** command mentioned in the previous screen-shot to connect to our newly started instance:

```
[root@workstation ~]# ssh -i deploy1key.pem centos@ec2-54-218-183-62.us-west-2.compute.amazonaws.com
The authenticity of host 'ec2-54-218-183-62.us-west-2.compute.amazonaws.com (54.218.183.62)' can't be established.
ECDSA key fingerprint is SHA256:WRucACtXNTbAlvwfuynPEgqo6FjJoLas6bLKymPJrEQ.
ECDSA key fingerprint is MD5:44:b5:04:e8:87:ad:24:19:01:a3:e9:8d:a7:0e:42:34.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-54-218-183-62.us-west-2.compute.amazonaws.com,54.218.183.62' (ECDSA) to the list of known hosts.
Last login: Sat Apr 11 11:03:17 2020 from
[centos@ip-172-31-21-125 ~]$
```

Figure 13.33 – Connecting to our instance via SSH

That's it. You have successfully connected to your machine. You can even keep it running. But be aware, if you have accounts or services that are on by default or have no password – you have, after all, pulled a VM out of your safe, home sandbox and stuck it on the big, bad internet. And one last thing: after you have your VM running, delete the file in the bucket to save you some resources (and money). After conversion, this file is no longer needed.

The next topic on our list is how to extend our local cloud environments into hybrid cloud environments by using an application called Eucalyptus. This is a hugely popular process that

a lot of enterprise companies go through as they scale their infrastructure beyond their local infrastructure. Also, this offers benefits in terms of scalability when needed – for example, when a company needs to scale its testing environment so an application that its employees are working on can be load-tested. Let's see how it's done via Eucalyptus and AWS.

Building hybrid KVM clouds with Eucalyptus

Eucalyptus is a strange beast, and by that, we do not mean the plant. Created as a project to bridge the gap between private cloud services and AWS, Eucalyptus tries to recreate almost all AWS functionalities in a local environment. Running it is almost like having a small local cloud that is compatible with AWS, and that in turn uses almost the same commands as AWS. It even uses the same names for things as AWS does, so it works with buckets and all of that. This is on purpose, and with consent from Amazon. Having an environment like this is a great thing for ev-

erybody since it creates a safe space for developers and companies to deploy and test their instances.

Eucalyptus consists of several parts:

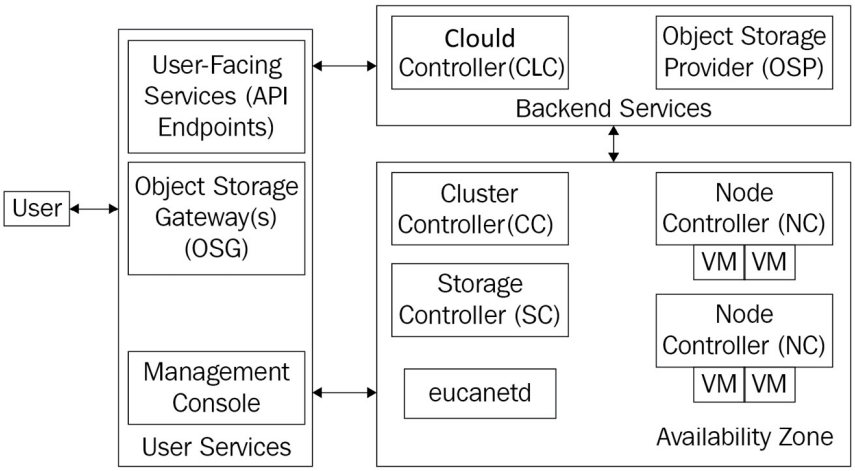

Figure 13.34 – Eucalyptus architecture (http://eucalyptus.cloud, official documentation)

As can be seen from the diagram, Eucalyptus is highly scalable.

An **availability zone** is one segment that can hold multiple nodes controlled by a cluster controller. **Zones** are then combined into the cloud itself, and this is controlled by the **Cloud Controller**. Connected to all this is the user services part that enables interaction between the user and the whole Eucalyptus stack.

All in all, Eucalyptus uses five components that are sometimes referred to by their names from the diagram, and sometimes by their project names, much like OpenStack does:

- **Cloud Controller** (**CLC**) is the central point of the system. It provides both the EC2 and the web interfaces and routes every task to itself. It is there to provide scheduling, allocation of resources, and accounting. There is one of these per cloud.
- **Cluster Controller** (**CC**) is the part that manages each individual node and controls VMs and their execution. One is running in each availability zone.
- **Storage Controller** (**SC**) is there to provide block-level storage, and to provide support for instances and snapshots but within the cluster. It is similar to EBS storage from AWS.
- **Node Controller** (**NC**) hosts instances and their endpoints. One is running for each node.
- **eucanetd** is a service Eucalyptus uses to manage cloud networking, as we are talking about extending your local networks to the AWS cloud, at the end of the day.

When you get to know Eucalyptus, you'll notice that it has a vast array of capabilities. It can do the following:

- Work with volumes, instances, key pairs, snapshots, buckets, images, network objects, tags, and IAM.
- Work with load balancers.
- Work with AWS as an AWS integration tool.

These are just some of the features worth mentioning at the start of your Eucalyptus journey. Eucalyptus has an additional command-line interface called **Euca2ools**, available as a package for all the major Linux distributions. Euca2ools is an additional tool that provides full API and CLI compatibility between AWS and Eucalyptus. This means that you can use a single tool to manage both and to perform *hybrid cloud* migrations. The tool is written in Python, so it is more or less platform-independent. If you want to learn more about this interface, make sure that you visit **https://wiki.debian.org/euca2ools**.

How do you install it?

Installing Eucalyptus is easy, if you are installing a test machine and *following the instructions*, as we'll describe in the last chapter of the book, ***Chapter 16***, *Troubleshooting Guideline for the KVM Platform*, which deals with KVM troubleshooting. We are going to do just that – install a single machine that will hold all the nodes and part of the whole cloud. This is, of course, not even close to what is needed for a production environment, so on the Eucalyptus website, there are separate guides for this single-machine-does-all situation, and for installing production-level clouds. Make sure that you check the following link: **https://docs.eucalyptus.cloud/eucalyptus/4.4.5/install-guide-4.4.5.pdf**.

Installation is simple – just provide a minimally installed CentOS 7 system that has at least 120 GB of disk space and 16 GB of RAM. These are the minimums. If you go below them, you will have two kinds of problems:

- If you try to install on a machine that has less than 16 GB of RAM, the installation will proba-

bly fail.

- The installation will, however, succeed on a machine with a smaller disk size than the minimum recommended, but you will almost immediately run out of disk space as soon as you start getting the deployment images installed.

For production, everything changes – the minimums are 160 GB for the storage, or 500 GB of storage for nodes that are going to run Walrus and SC services. Nodes must run on bare metal; nested virtualization is not supported. Or, to be more precise, it will work but will negate any positive effect that the cloud can provide.

Having said all that, we have another point to make before you start installing – check for the availability of a new version, and have in mind that it is quite possible that there is a newer release than the one that we are working on in this book.

Important note

At the time of writing, the current version was 4.4.5, with version 5 being actively worked on and

close to being released.

Having installed your base operating system – and it has to be a core system without a GUI, it's time to do the actual Eucalyptus installation. The whole system is installed using **FastStart**, so the only thing we have to do is to run the installer from the internet. The link is helpfully given on the front page of the following URL for the project – **https://eucalyptus.cloud**.

There are some prerequisites for successful Eucalyptus installation:

- You have to be connected to the internet. There is no way to do a local installation this way, because everything is downloaded on the fly.
- You also have to have some IP addresses available for the system to use when installed. The minimum is 10, and they will get installed along with the cloud. The installer will ask for the range and will try to do everything without intervention.
- The only other prerequisites are a working DNS and some time.

Let's start the installation by using the following command:

```
# bash <(curl -Ls
https://eucalyptus.cloud/install)
```

The installation looks strange if you're seeing it for the first time. It kind of reminds us of some text-based games and services that we used in the 1990s (MUD, IRC):

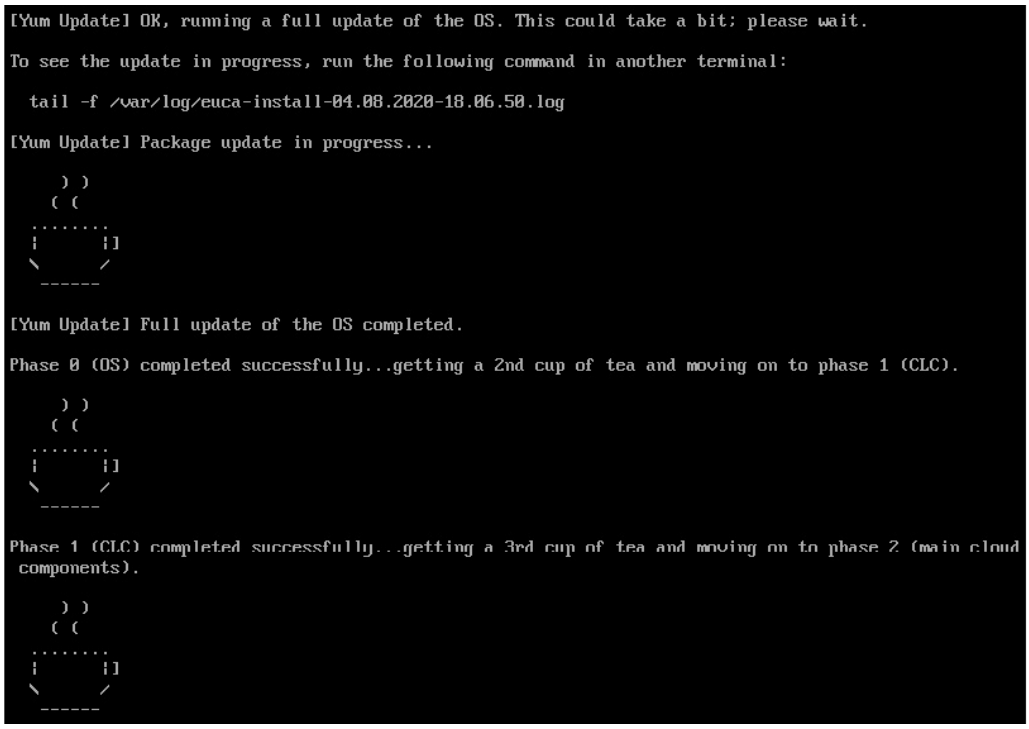

Figure 13.35 – Eucalyptus text-mode installation

The information on the screen will tell you which log to follow if you want to see what is actually happening; otherwise, you can look at the installer and wait for the tea on the screen to get

cold. In all honesty, on a decent machine, the installation will probably take around 15 minutes, or 10 minutes more if you install all the packages.

Once installed, Eucalyptus will provide you with a default set of credentials:

- **Account name**: `eucalyptus`
- **Username**: `admin`
- **Password**: `password`

  Important note

  In the event that the current installer breaks, the solutions to the subsequent problems are in the CiA video here: <video_URL>. There are known bugs, and may or may not be solved before this book hits the stores. Make sure that you check **https://eucalyptus.cloud** *and documentation before installation.*

The information is case sensitive. Having finished the installation, you can connect to the machine using a web browser and log in. The IP address you are going to use is the IP address of the machine you just installed:

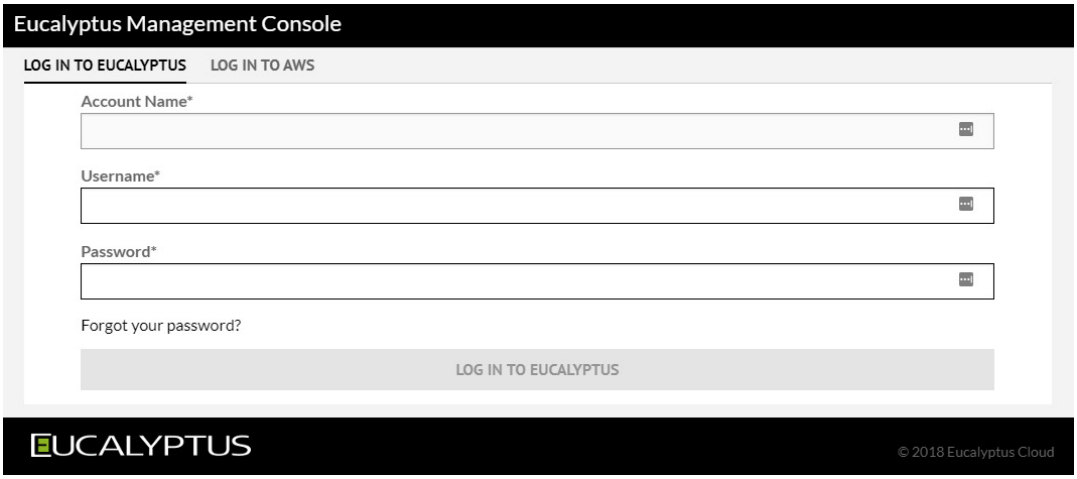

Figure 13.36 – Eucalyptus login screen

Once the system has finished installing, on the console of the newly installed system, you are going to be met with an instruction that will say to run the master tutorial contained on the system. The tutorial itself is a great way to get to know how the system looks, what the key concepts are, and how to use the command line. The only problem you may have is that the tutorial is a set of scripts that have some information hard-coded. One of the things you will notice straight away is that the links to the cloud versions of the image templates will not work unless you fix them – the links point to expired addresses. This is easy to solve but will catch you off guard.

On the other hand, by the time you read this, maybe the problem will be fixed. The tutorial on

how to do this and all of its parts is offered in plain text mode on the machine Eucalyptus is running on. It's not available in the GUI:

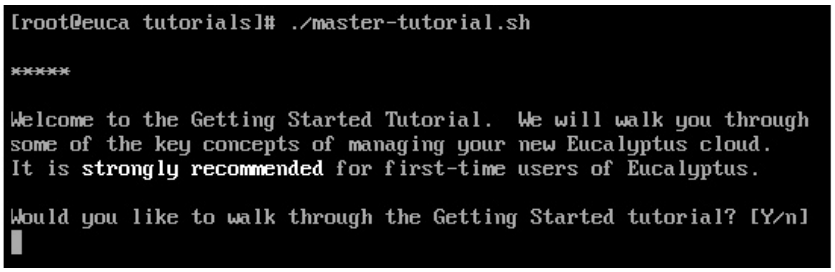

Figure 13.37 – Starting a text-mode Eucalyptus master tutorial

The tutorial is extremely rudimentary in its appearance, but we liked it because it gave us a short but important overview of everything that Eucalyptus offers:

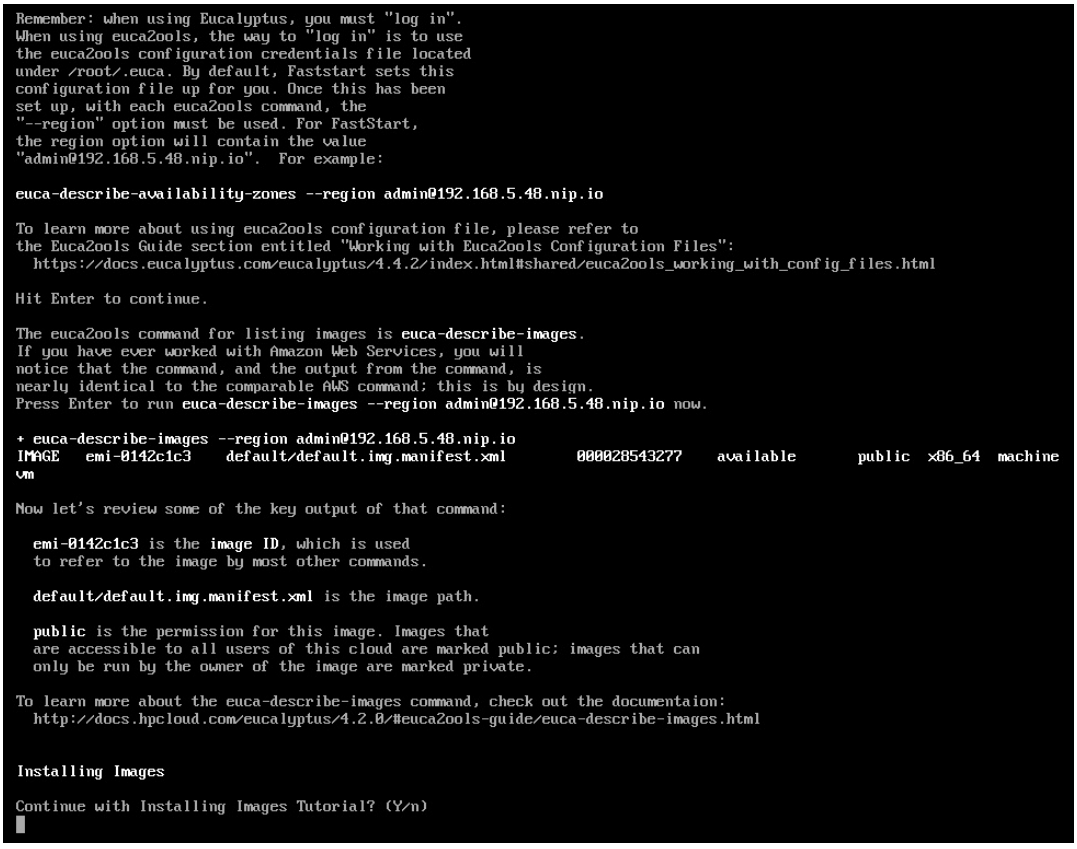

Figure 13.38 – Using the master tutorial to learn how to configure Eucalyptus

As you can see, everything is explained in detail, so you can really learn key concepts in a short amount of time. Go through the tutorial – it is well worth it. Another thing you can do from the command line as soon as you start up the system is to download a couple of new template images. The script for this is also started from the web, and is written in big letters on the official site, literally on the landing page located at the following URL (make sure that you scroll down a bit) – **https://www.eucalyptus.cloud/**:

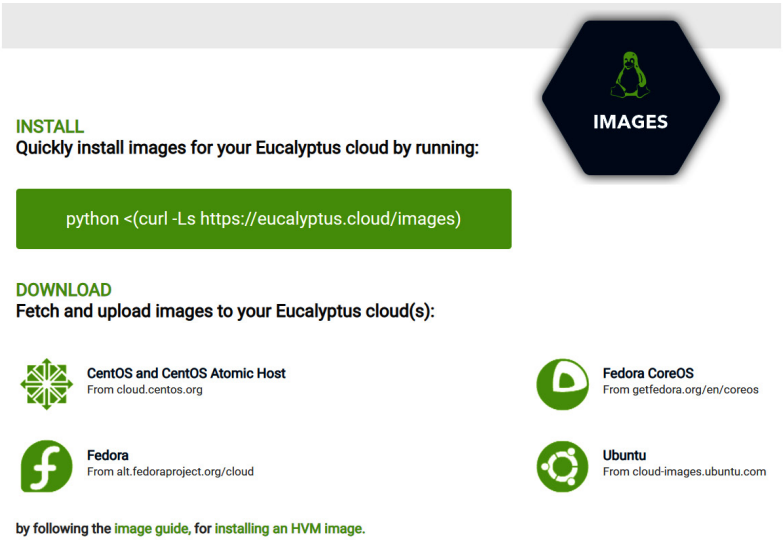

Figure 13.39 – Downloading images to our Eucalyptus cloud

Copy-paste this into the root prompt and, shortly, there will be a menu that will enable you to download images you may use. This is one of the simplest and most bullet-proof installations of templates we have ever seen, short of them being included in the initial download:

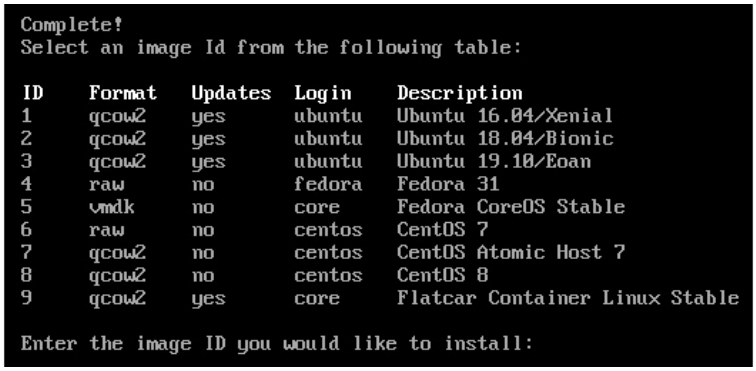

Figure 13.40 – Simple menu asking us to select which image we want to install

Choose one at a time, and they will get included in the image list.

Now let's switch to the web interface to see how it works. Log in using the credentials written above. You will be greeted with a well-designed dashboard. On the right, there are groups of functionalities that are most commonly used. The left part is reserved for the menu that holds links to all the services. The menu will autohide

as soon as you move your mouse away from it, leaving only the most essential icons:

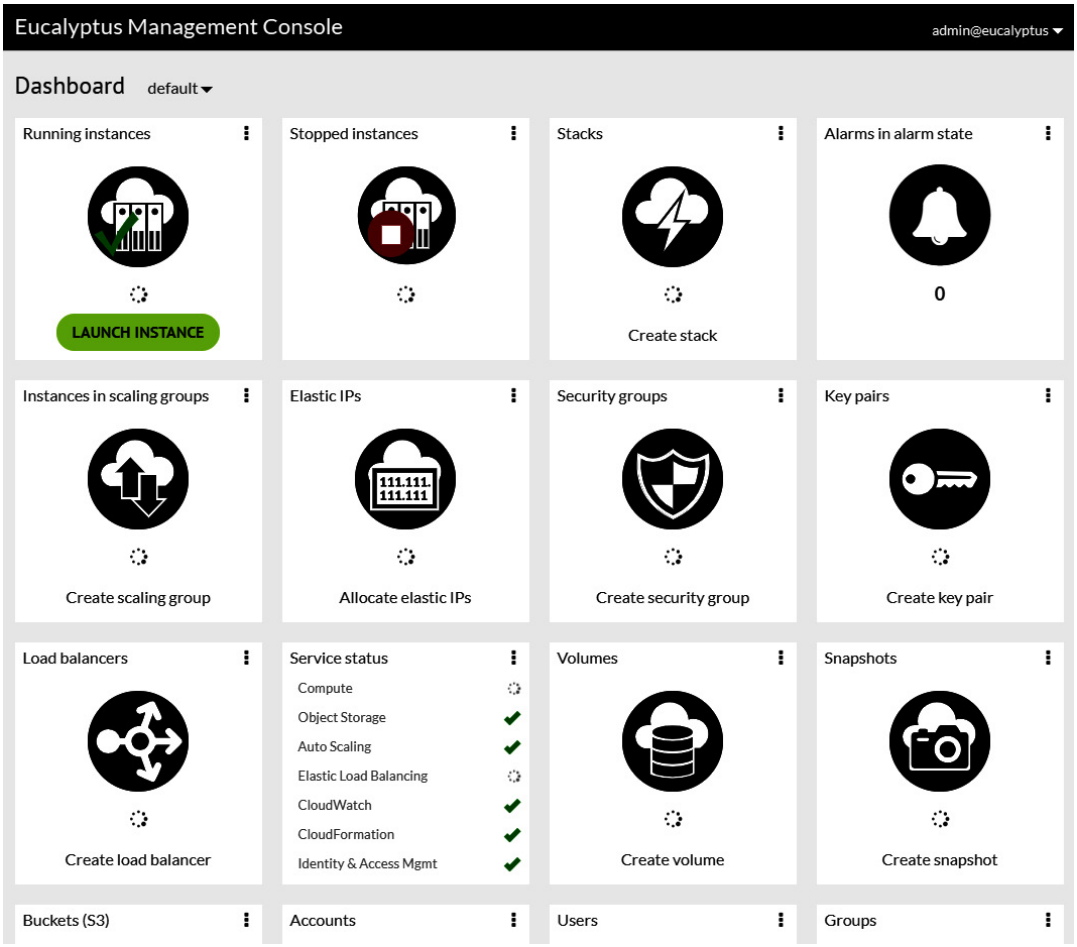

Figure 13.41 – Eucalyptus management console

We already discussed most of the things on this page – just check the content in this chapter related to AWS and you'll be in very familiar territory. Let's try using this console. We are going to launch a new instance, just to get a feel for how Eucalyptus works:

1. In the left part of the stack of services, there is an inviting green button labeled **Launch in-**

stance – click on it. A list of the images that are available on the system will appear. We already used the script to grab some cloud images, so we have something to choose from:

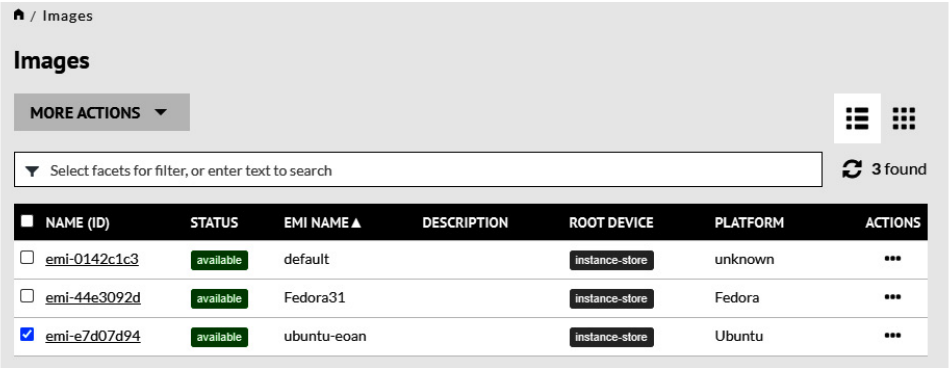

Figure 13.42 – Selecting an image to run in the Eucalyptus cloud

We chose to run Ubuntu from the cloud image. Choose **Launch** from the drop-down menu after you have selected the image you want. A new window opens, permitting you to create your virtual machine. In the dropdown or the instance type, we chose a machine that looked powerful enough to run our Ubuntu, but basically, any instance with over 1 GB of RAM will do fine. There is not much to change since we are preparing just one instance:

Figure 13.43 – Launch a new instance wizard

The next configuration screen is related to security.

2. We have a choice of using the default key pair that was created on the Eucalyptus cloud or creating a new one. Only the public part of the key is stored in Eucalyptus, so we can use this key pair for authentication only if we downloaded the keys when we installed them. The process of creating keys is completely identical to the one used for AWS:

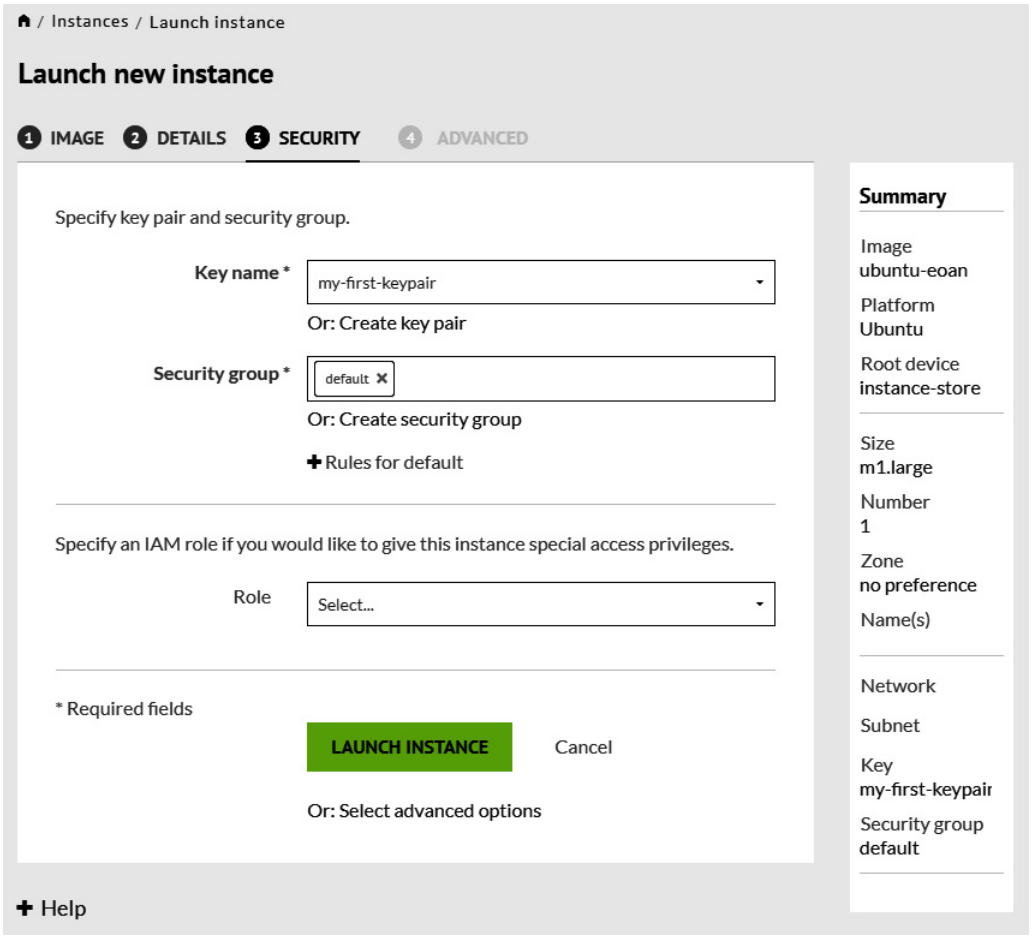

Figure 13.44 – Security configuration – selecting keys and an IAM role

After clicking the **LAUNCH INSTANCE** button, your machine should boot. For testing purposes, we already started another machine earlier, so right now we have two of them running:

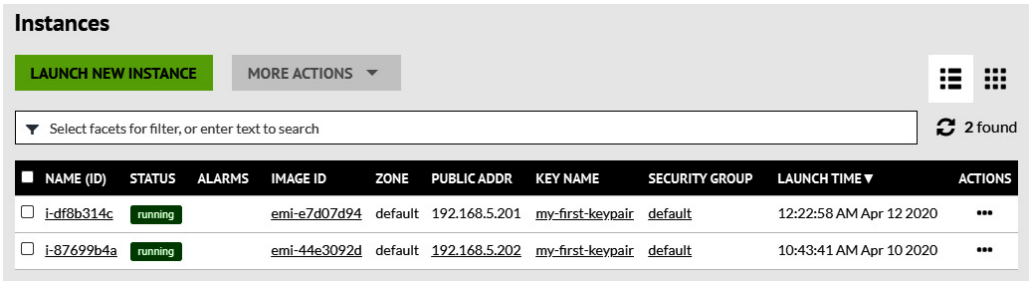

Figure 13.45 – A couple of launched instances in the Eucalyptus cloud

The next step is trying to create a storage bucket.

3. Creating a storage bucket is easy, and looks very similar to what AWS enables you to do since Eucalyptus tries to be as similar to AWS as possible:

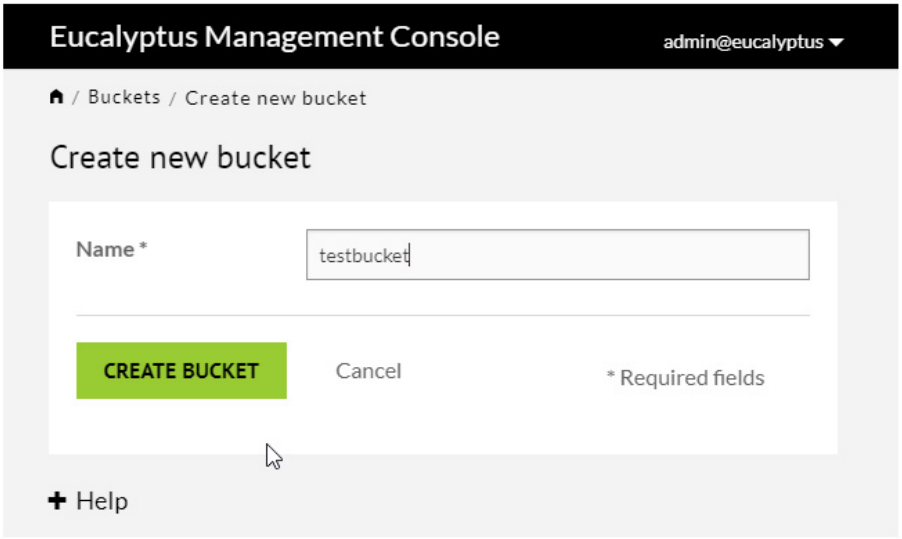

Figure 13.46 – Creating a bucket

Since Eucalyptus is not as complex as AWS, especially in regard to policy and security, the security tab for the bucket is smaller, but has some very powerful tools, as you can see in the following screenshot:

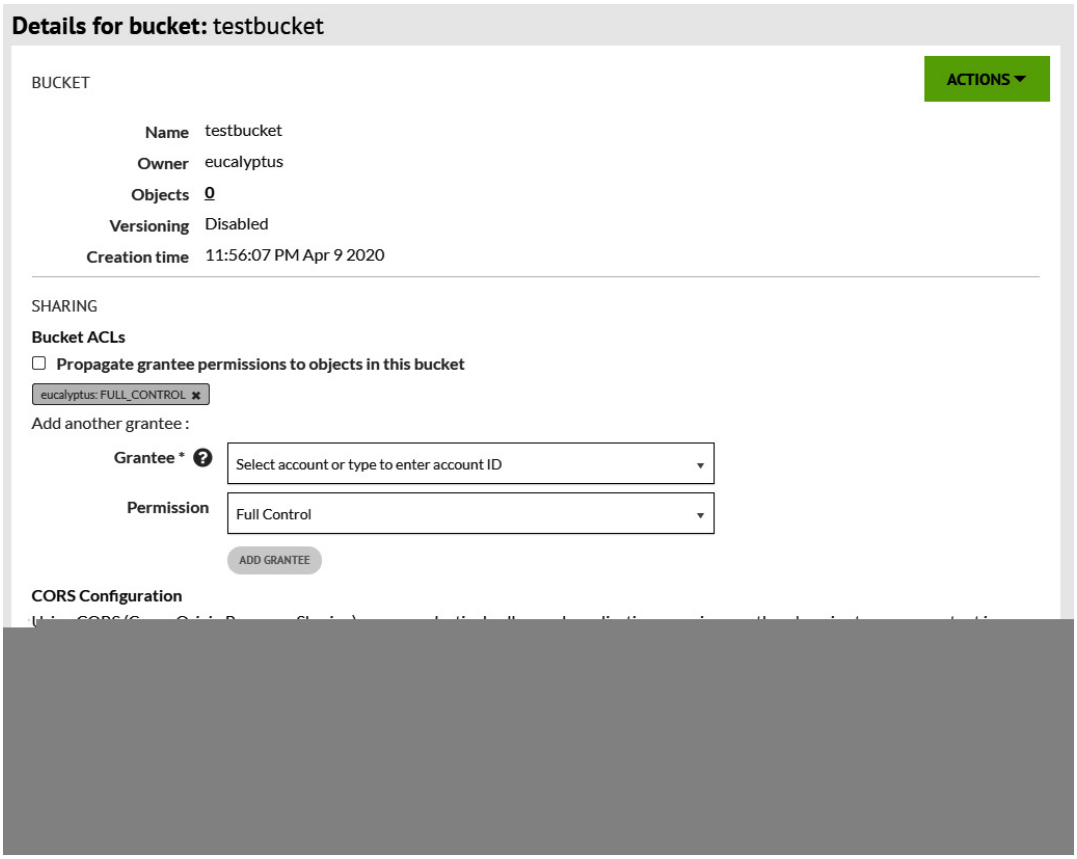

Figure 13.47 – Bucket security configuration

Now that we have installed, configured, and used Eucalyptus, it's time to move on to the next topic of our chapter, which is scaling out our Eucalyptus-based cloud to AWS.

Using Eucalyptus for AWS control

Do you remember the initial screen that showed you the login credentials, and we mentioned that you can also log in to AWS? Log out of your Eucalyptus console and get to the login screen. Click on **LOG IN TO AWS** this time:

Figure 13.48 – Log on to AWS via Eucalyptus

Try and use the auth key we created in the *Uploading an image to EC2* section, or create a new one for the **`Administrator`** user in AWS IAM. Copy and paste it into the AWS credentials, and you will have a fully working interface connected to your AWS account. Your dashboard will look almost the same, but will show the status of your AWS account:

Figure 13.49 – Eucalyptus Management Console for AWS

Let's check we can see our buckets:

Figure 13.50 – Checking our AWS buckets

Notice that we not only see the bucket we used to test the AWS KVM import, but we also see the region we are running in, in the upper-right corner. Your account is given by its key name, not the actual user; this is simply because we are in fact logged in *programmatically*. Everything we click gets translated to API calls, and the returned data is then parsed and displayed to the user.

Our currently stopped instance is also here, but keep in mind that you will see it only if you choose the region you initially imported your instance into. In our case, it was **`US West`**, so our instance is there:

Figure 13.51 – Checking our AWS instances

As you have probably noticed, Eucalyptus is a multi-faceted tool that's able to provide us with hybrid-cloud services. Basically, one of the key

points of Eucalyptus is the fact that it gets you to an AWS-compatible level. So, if you start using it as a private solution and sometime in the future start thinking about moving to AWS, Eucalyptus has you covered. It's a de-facto standard solution for KVM-based virtual machines for that purpose.

We will stop here with the AWS integration. The point of this chapter was, after all, to get you to see how Eucalyptus connects to AWS. You might see that this interface lacks functionality that AWS has, but can at the same time be more than enough to control a basic mid-size infrastructure – buckets, images, and instances from one place. Having tested the 5.0 beta 1 version, we can definitely tell you that the full 5.0 version should be quite a substantial upgrade when it comes out. The beta version already has many additional options and we're rather excited to see when the full release comes out.

Summary

In this chapter, we covered a lot of topics. We introduced AWS as a cloud solution and did some cool things with it – we converted our virtual machine so that we can run in it, and made sure that everything works. Then we moved to Eucalyptus, to check how we can use it as a management application for our local cloud environment, and how to use it to extend our existing environment to AWS.

The next chapter takes us into the world of monitoring KVM virtualization by using the ELK stack. It's a hugely important topic, especially as companies and infrastructures grow in size – you just can't keep up with the organic growth of IT services by manually monitoring all possible services. The ELK stack will help you with that – just how much, you'll learn in the next chapter.

Questions

1. What is AWS?
2. What are EC2, S3, and IAM?
3. What is an S3 bucket?
4. How do we migrate a virtual machine to AWS?

5. Which tool do we use to upload a raw image to AWS?
6. How do we authenticate ourselves to AWS as a user?
7. What is Eucalyptus?
8. What are the key services in Eucalyptus?
9. What are availability zones? What are fault domains?
10. What are the fundamental problems of delivering Tier-0 storage services for virtualization, cloud, and HPC environments?

Further reading

Please refer to the following links for more information regarding what was covered in this chapter:

- Amazon AWS documentation: **https://docs.aws.amazon.com/**
- Amazon EC2 documentation: **https://docs.aws.amazon.com/ec2/?id=docs_gateway**
- Amazon S3 documentation: **https://docs.aws.amazon.com/s3/?**

id=docs_gateway

- Amazon IAM documentation: https://docs.aws.amazon.com/iam/?id=docs_gateway
- Eucalyptus installation guide: https://docs.eucalyptus.cloud/eucalyptus/4.4.5/install-guide/index.html
- Eucalyptus administration guide: https://docs.eucalyptus.cloud/eucalyptus/4.4.5/admin-guide/index.html
- Eucalyptus console guide: https://docs.eucalyptus.cloud/eucalyptus/4.4.5/console-guide/index.html
- Euca2ools guide: https://docs.eucalyptus.cloud/eucalyptus/4.4.5/euca2ool guide/index.html